Logical Representation of the Timed Automata in PVS*

Qingguo XU[1,2], Huaikou MIA0[1]

[1]*School of Computer Engineering and Science, Shanghai University, Shanghai 200072, P. R. China,*
[2]*State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, P. R. China, {qgxu,hluniao}@shu.edu.en*

Abstract. This paper gives us a representation of timed automata in PVS (Prototype Verification System) specification. So the real time system can be formally verified in a different method from model checking techniques because this kind of representation of timed automata is implemented in a logical view. Finally, a toy case study is model and verified in this representation.

Keywords: real-time system, Logical representation, Timed Automata, PVS

1 Introduction

Timed Automata(TA) were introduced by R. Alur and D. L. Dill in 1994 **[1]** as an extension of Biichi-automata (which are again an extension of finite automata), to model the behaviour of real-time systems over time. Researchers have proposed many innovative formal methods for developing real-time systems using TA model [2-4]. Such methods can give system developers and customers greater confidence that the real-time systems satisfy their requirements, especially their critical requirements.

This paper will introduces how we have built upon the modeling system PVS to support formal specification of real-time systems modeled as TA. We first give the theories about clock manipulations in PVS, and then a generic timed automaton theory template for modeling real-time system by importing the clocks theory, as well as the framework used for verifying some properties under the aid of the clock manipulation theories. Finally a case study is investigated.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries about timed automata, which are consisted of formal syntax and semantics for clock constraints. As a main body, section 3 will present the logical expression for clock manipulations and semantics of the TA formalism. Section 4 constructs the TA theory's PVS specification based on the ideas in section 3, a toy case study using the logical representation is investigated in section 5. Finally, the conclusions and the future works under consideration are sketched out.

2 Timed Automata Model

The Timed Automata[1], which are used to model real-time system, was invented by Rajeev Alur and David L. Dill. In order to model timed behaviors, the clock constraints should be firstly defined first.

*Supported by the National Natural Science Foundation of China under Grant No.60970007, 61073050, 61170044 ; State Key Laboratory of Software Engineering (SKLSE) under Grant No. SKLSE2010-08-26;the National Grand Basic Research Program of China under grant No. 2007CB310800; Science and Technology Commission of Shanghai Municipality under Grant No. 10510704900; the Shanghai Leading Academic Discipline Project of China under Grant No.J50103.

Definition 1 (Guards, Invariants, Clock Constraints[5]) Let C be a finite set of real-valued variable, called clocks. A guard cp in the set *$O_G(C)$* can be defined by BNF-grammar

$v:=x—clx—y—cltruelfalselv_i nv_2 Iv_i vq;0_2 1--iv_i$

while an invariant cp in the set *OK)* can be defined by BNF-grammar

$:=x—clx—y—cltruelfalselv_i nv_2$

where x and *y* are clocks, *c* is a rational constant and —E {<,5_,=,,>}. •

The restriction to rational time constants is needed for decidability of emptiness and reachability of timed automata.

We say that a clock interpretation v for C satisfies a clock constraint cp over C if and only if cp evaluates to true according to the values given by v. For Se 9t, v+gdenotes the clock interpretation which maps every clock x to the value v(x) + g For *YcC, v[Y:=t]* denotes the clock interpretation for *C* that assigns t to each xe Y, and agrees with v over the rest of the clocks.

Definition 2 Timed Automaton

A timed automaton is a tuple<L, *1°, E, C, Inv, Gad, Rst>* with

- *L,* a finite set of locations with initial locations set PE *L*
- *EcLxL,* a set of edges
- *C,* a finite set of clocks
- *Inv: L—g1) (C) ,* a function that assigns to each locations *lE L* an invariant *Inv())*
- *Gad: E—>O_G (C),* a function that labels each edge *eEE* with a clock constraint *Gad(e)* (called *guard)* in 1G *(C)* over *C,* and
- *Rst: E—>.2^c,* a function that assigns to each edge *eeE* a set of clocks *Rst*

(e). LX2^cx(D_G(C) xL is a set of switches. A switch *< s'V'A's >* represents a transition from location *s* to location *s'* on symbol *a,* if *e=(s, s')* is in *E, q.' = Inv(s)* A *Gad (e)* is enabled, and X, = *Rst (e).* ■

Definition 2 is different from that given in [6] to some extent. Using our definition here, it is easy to generate cp automatically for every switch > using the conjunction of *Gad* and *Inv.* The label set is omitted because only one automaton is investigated and all the labels can be consider as one internal action.

The semantics of a TA *A* is defined by associating a transition system [6] *S_A* = < n0, with it. There are two types of transitions in *SA:*

- State can change due to elapse of time: for a state *(s, v)* and a real-valued time increment *g(s v)(s,v-Fg)*if V *',08'8,v+6''* satisfies the invariant *Inv(s),*
- State can change due to a locations-switch: for a state *(s,* v) and a switch *<s,co,A,,s'>* such that v satisfies p, (s' v) (s v[11 := 0])

A run for *A* is defined by a state sequences starting from an initial state and triggered by an action (time-delay or locations-switch):(s_o, v_o)4 ($s_/$, v_i) 4 *(s_2, v_2) 4 (s_2, v_2)*.............

where 4E { , }, *(s_o, v_o)E* $Q^°$, and *(s_1, v_i)E* Q.

3 Logical Representation of a TA

- **clock representation and its manipulation**

Let *A = (L, $I^°$, E, C, Inv, Gad, Rst)* be a timed automaton. A new clock *gt* C —called absolute time (or global time) reference — is introduced. This clock *gt* is used to measure the time that has passed through during the run of *A.* At the ith state in a run,

ofgtd(i)=m iff at the ith state, time amount m was elapsed since the beginning of the run. (f1)

For every clock xE *C,* a rational timed variable x_t is introduced, with the intended

cr(Y_t)(0=m mat the ith state in a run, m is the absolute time when x was last rest. *(f2)*

Based on the regulations in *(f1)* and *(f2),* we have

cr(gt)(i)- of X_t)(i) = c ?ff the value of clock x at the ith state in a run is c.

Remark 1: According to the (f1), (12) and (13), all the clock constraints can be expressed in the style ofx — y *c.*

- **Representation of state transition**

Let *A=(L, 1°, E, C, Inv, Gad, Rst)* be a timed automaton, Let C be the set containing all these variables x *$_f$*. a state of S_A is a pair *(s, v)* can be expressed by *(s, vt, gt),* where v_t is an clock interpretation substitute x by .Y „

According to semantics in section 3.1, there are two kinds of transitions: action transition and time delay transition.

1) Action Transition

For an action transition, state change (which is instantaneous), our representation must ensure the clocks which are not reset keep their values, while those which are reset are set to the absolute point in time. The state variables change according to the states involved in the transition, and the guard has to hold.

An action transition *(s,v)—>(s',14/1:=* 0]) can be rewritten *(s,v_t, gt)* *(s ',v $_1$[2 := gt], gt)* . Using the expression of clock, we can encode this kind of transition as

TranAct ((s v) *(s',* *:= 0])) := S_i* **A** $\mathbf{T_{i+i}}$ **A** *(gt = g^t i+1)* **A**
xe *+^{1}* *^{1}+^{1}* *x0*

(.Y_i = *)* **A** **A(Y$_{i\pm i}$ =** *(f4)*

where s_1 and T_{i+i} are encoding of location component for a state in a run of *A* at *t^h* and (i+1)th step. This can be implemented by defining the data type of locations

2) Time delay transition

The time delay transition $^{(s,v)\text{—>}(s,v\overset{g}{+}(5)}$ can be expressed by *(s,* v„ *gt)* *,v gt + 8)*

Similarly, this transition can be encoded using the following expression:

Tran$_s$((s, v) `(s,v+ 8)) :=7 **A7$_{1+1}$** *AInv(7,) AInv(7,$_{\pm 1}$)A(gt, < gt$_{+i}$)* Aq *(15)*
xEc

3) Complete Transition Relation

The transition step in a run of a TA is either a time delay transition or an action one:

PTran (A) := V $_{(s\ s,)}$Tran $_{act}$ ((s, V) —>^{a} (s , v[2 := 0])) V Tran$_s$ ((s , v) *(s,v + S)) (f6)*

- **Representation Of Initial State**

Let A be a timed automaton, and *lo* be its initial location and *1* 0 the encoding of the initial locations. The representation of initial conditions is given by

9,„„ *(A):=/_* **0 A** *(gt $_0$* **= 0) A A$_{xec}$ (X$_0$** *(F)*

- **Representation of Timed Automaton Initial State**

The whole TA *A* can be expressed by these conjunctions:

9(A) *$^($Pinit (A) A trap (A)* *f)*

In the following Section 4, we will translate this representation into PVS (Prototype

Verification System) theories written in PVS specification. The three components *(f4), (f6)* and *(f7)* are ensured by three constraints in runs of *A*.

4. Formalization of TA via PVS

- About PVS

PVS[7] provides an integrated environment for the development and analysis of formal specifications, and supports a wide range of activities involved in creating, analyzing, modifying, managing, and documenting theories and proofs. The system consists of a specification language, a parser, a type checker, and an interactive proof checker.

Specifications in PVS consist of one or more theories. Each theory may be parameterized and may import other theories. In proving theorems in PVS, users can apply a sequence of primitive or used-defined proof steps. This is very convenient for user to formalize a system and prove the corresponding theorems[8].

- **Clock Constraints in PVS**

```
clkp[N: posnat]: theory begin
  importing clkinterp[N]
  x, y: var clock
  neq(x): type = {y 1 y # x}
  dif: datatype begin
    -(x: clock, y: neq(x)): sub?
  end dif
  %=---Syntax construct--
  clkp: datatype begin
        <=(s: dif, c: int): le?

    true: true?
    AND(fl, f2: clkp): and?
          clkp): not?
  end clkp
```

```
 Guard: type = clkp
 islnv(f:clkp): inductive bool =
        if and?(f) then islnv(fl(f)) and islnv(f2(f))
       else not (not?(f) or or?(f)) endif
 Inv: type = (islnv)
 v: var clockVal
%---Semantics interpretation=====
 1=(v, (Eclkp)): RECURSIVE bool =
 cases f OF       (s,  c):  v(x(s))  -  v(y(s))  c,
         AND(fl, f2): (v 1= fl) and (v 1= f2),

       endcases measure (Xv, f) by <<;
 convex_inv: lemma V(f: Inv):
 v 1= f && v +t                          v+t11= 0
end clkp
```

Figure 1. The Theory about clock constraints

According to **Remark 1,** Figure 1 gives the clock constraints in the form of x — y *c* , for XE *C* U {gt}, yE C and —E {<,,=,,>}. The theory also reveals the differences between location invariant and guard of an edge in a TA. The formulae *convex_inv,* which can be proved using structural induction method, tells us the clock invariant is convex.

- Representation of TA in PVS

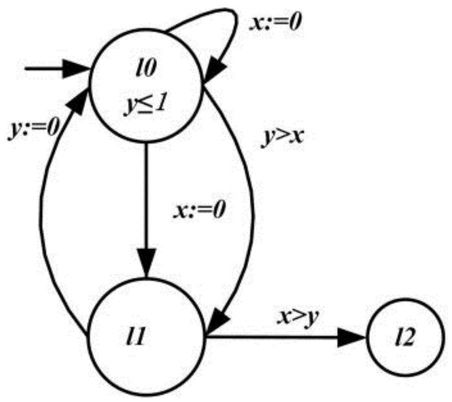
```
TimedAutomaton[Locations: type, 10:
Locations,N: posnat, (importing clkp[N]) Inv: [Locations—Inv], Edge:
pred[[Locations, Locations]],Guard: [(Edge)—Guard],ResetC: [(Edge)⁻
'pred[clock]]]: theory begin
States:type=[# loc: Locations, v: clockVal#]
PreRuns:type= sequence[States] .........
delta(s0, d, sl): bool = sl = sO + d && (sO'v 1= Inv(sO'loc)) && (sl'v 1= Inv(sl'loc))
locswitch(s, sl): bool = let sw = (s'Ioc, sl'loc) in Edge(sw)&&(s'v 1= (Inv(s'loc) and
Guard(sw))) && (sl'v = reset(s'v)(ResetC(sw))(s'v(0))) && (sl'v 1= Inv(sl'loc)) Step(s,
sl:States): bool=(3d:delta(s, d, sl)) or locswitch(s, sl)
Runs: type = {pr 1 NonZeno(pr) && Init(pr(0))&&
Vi:Step(pr(i),pr(i+1))1 end TimedAutomaton
```

Figure 2. The Theory TimedAutomaton

In this subsection we will give the generic TA theory. According to the formal definition *A=<L, E, C, Inv, Gad, Rst>* of TA in Definition 2 and the corresponding transition system in Definition 4, we model *TimedAutomaton* theory in PVS. This theory shown in Figure 2 has 7 formal parameters, where *Locations, 10, Inv, Guard,* and *ResetC* correspond with the components *L*, $\overset{\circ}{L}$, *Inv*, *C*, *Gad* and *Rst* of *A* respectively, *Edge* correspond with *E* of A. The other parameter *N* represents the clocks number. It is worthy noted that some functions about these arguments are implemented by importing the theories defined in section 3. The operator "'", which occurs followed by a variable or a constant in PVS specification, is the field accessor of a tuple or a record type.

Figure 2 defines the semantics of TA using data type *Runs* over the *States.* Its attributes must be ensured by *(f8)* in section 3. It is noted that the clock manipulation has been modified compared with that defined in most of the references[1, 3, 9]. The time delay step (see the function *delta()* in Figure 2) only increase the global time rather than all the clock value because of the representation $Tran_s$ introduced in section 3, and the location-switch transition (or action transition, see function *locswitch()* in Figure 2) resets some clocks to the current global time *v(gt)(i)* rather than 0 because an action transition *(s ,V) —> (s* := 0]) can be rewritten *(s,* v_t, *gt)* *(s',*v_t*[A, := gt], gt)* that had been mentioned in section 3.

5 A Toy Case Study



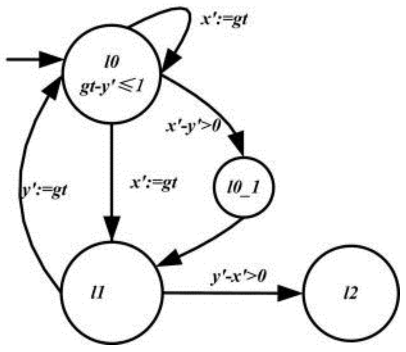


Figure 3. A toy TA Named *toyTA* **Figure 4.** modified *toyTA* named *ourTA*

Consider the TA named *toyTA* in Figure 3, with three locations /0, /1, l2 and two clocks *x, y.* This is an example appeared in Maria Sorea's work [9].

Let's introduce the absolute time *gt* for *toyTA,* and clocks **x'** and *y'* for clocks **x** and y based on *(fl), (f2)* and *(f3)* in section 3. Of course, the clock constraints occur in a location or an edge should be changed correspondingly according to **Remark 1. The** clock reset can be rewritten with respect to *(f2).* The associated TA named *ourTA* shows in Figure 4. It is noted a location is added for distinguish edges that has the same source and destination locations.

```
ourTA: theory begin
  Locs: type = {10, 11, 12, 10_1}
  N: posnat = 3 importing clkp[3]
  L, Ll : var Locs x: clock = 1 y: clock = 2
  Inv(L): Inv = if L = 10 then gt - y <= 1 else true endif
  Edge(L, L1): bool = cases L of 10: Ll # 12, 11: Ll = 10 or Ll = 12, 10_1: Ll = 11 else false endcases
  Gad(e):Guard= if e = (11, 12) then y-x > 0 elsif e = (10,10_1) then x-y>0 else true endif
  RstC(e)(xx): boot= if e=(11,10) then xx=y elsif e=(10,10) or e=(10,11) then xx=x else false endif
  importing TimedAutomaton[Locs, 10, N, Inv, Edge, Gad, RstC]
  12_not: lemma Vi: r(i)'Ioc # 12
end ourTA
```

Figure 5. The Theory ourTA

We can get the specification for *ourTA* given in Figure 5 by specifying the actal parameters to instance the formal ones. The type of locations are defined the abstract data type to ensure all the locations are dijoint pairwise. The clocks x and *y* are corresponding to x' and *y'* respectively for simplity.

The property "the location *12* will not be reachable in all the runs" ourTA, can be ensured and proved easily using the induction method on run step. That is to say, This specification for *ourTA* is trusted to some extent.

6 Conclusion and Future Work

Based on the logical representation of every component of TA, this paper implements a real-time system modeling method using PVS. This method adapts the clock constraints expression by replace the new clock for every clock variable. The results about modeling and verify a case study demonstrate the method is correct and in effect.

The future works about FVofTA can be done further in the following parts: first, the system (or network) of several TAs, in which cases, the communication mechanism of different sub TA must be determined and specified in a simple style. Second, in order to improve the verification efficiency, some preliminary rules or proving sechemas for verifying some kind of properties should be specified using PVS in advance.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science 126, 183-235 (1994)
2. Archer, M., Heitmeyer, C.: TAME: A Specialized Specification and Verification System for Timed Automata. Work In Progress (WIP) Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96), pp. 3-6, Washington, DC (1996)
3. Xu, Q., Miao, H.: Formal verification framework for safety of real-time system based on timed automata model in PVS. IASTED International Conference on Software Engineering, as part of the 24th IASTED International Multi-Conference on APPLIED INFORMATICS, Febrary 14, 2006 - Febrary 16, 2006, vol. 2006, pp. 107-112. Int. Assoc. of Science and Technology for Development, Innsbruck, Austria (2006)
4. Olderog, E.R., Dierks, H., Corporation, E.: Real-time systems: formal specification and automatic verification. Cambridge University Press (2008)
5. Kemper, S.: SAT-based Verification for Abstraction Refinement. University of Oldenburg (2006)
6. Alur, R.: Timed Automata. Theoretical Computer Science 126, 183-235 (1999)
7. Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. 11th International Conference on Automated Deduction (CADE), vol. 607, pp. 748-752. Springer-Verlag, Saratoga, NY (1992)
8. Owre, S., Shankar, N., Rushby, J.M., Stringer-Calvert, D.W.J.: PVS System Guide. Menlo Park, CA (1999)
9. Moller, M.O., Rue , H., Sorea, M.: Predicate Abstraction for Dense Real-Time Systems. Electronic Notes in Theoretical Computer Science 65, 218-237 (2002)