Constructing the Reaching Region Graph for Timed Automata with PVS

Huaikou Miao, Qingguo Xu

School of Computer Engineering and Science, Shanghai University
No. 149, Yanchang Rd., Shanghai 200072, P. R. China

Summary
Based on our existing works, this paper firstly gives clock region equivalence PVS specification, and then constructs the reachable region graph for a given Timed Automaton (TA) via characterizing some kinds of clock regions, finally analyses the reachable states using the region graph. These works can conveniently analysis some real-time system in the form of TA model. A case study is investigated and the results are satisfying. As a by-product, an error is detected in the region-equivalence definition which is extensively referred in many papers.

Key words:
Timed Automata, Clock region, Region equivalence, Reachability analysis, PVS

1. Introduction

Timed automata (TAs) are a specification and verification model for timed systems [1], systems that involve real-valued variables. The problem that underlies the safety verification for a TA is reachability: can an unsafe state be reached from an initial state by executing the system? The traditional approach to reachability in model checking techniques attempts to construct region timed automaton or zone automata for a given TA. Almost of this kind of works are carried out in programming language.

In this paper, we want to introduce the region construction techniques in PVS (Prototype Verification System), a well-known formal verification tool developed by SRI International. Other research works [2, 3, 9, 12] about modeling and verifying real-time system modeled by TA using PVS only consider the deduction method. These works share a common shortcoming: the reachable states space for a TA cannot be determined using PVS.

The works in this paper are the extensions to our previous works [13], a mechanized system called FVofTA (Formal Verification of Timed Automata) for specifying and reasoning about real-time systems using TA theory in PVS. The overall structure of FVofTA is shown in Figure 1. Some preliminaries may refer to [13].

This work is supported by the Natural Science Foundation of China (60373072), National 973 Program (2002CB312001) and the Science Development Foundation of Education Committee of Shanghai.

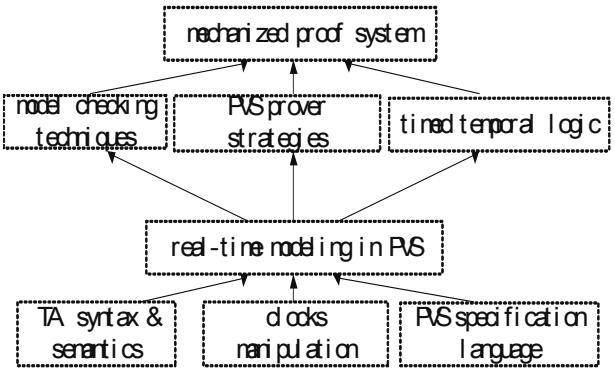


Figure 1. The Structure of FVofTA

The rest of this paper is organized as follows. Section 2 introduces the formal syntax and semantics for timed automata. Section 3 investigates the clock regions and its classifications with the corresponding characterizations. Section 4 firstly formalizes the clock region equivalence theory, and then details the PVS implementations about the region automaton construction method and the reachability analyzing techniques using model checking algorithm, as well as a sample case study. The related works are discussed in section 5. Finally, the conclusions and the future works under consideration are sketched out.

2. Timed Automata Model

The Timed Automata [1, 8] for modeling real-time system was invented by Rajeev Alur and David L. Dill. In order to model timed behaviors, the clock constraints and the clock interpretations should be firstly defined.

Definition 1 clock interpretations
A clock interpretation v for a set C of clocks assigns a real value to each clock. It is a mapping from C to the set $\mathbb{R}$ of nonnegative real numbers. ■

Definition 2 clock constraints
For a set of C of clock variables, the set $\Phi(C)$ of clock constraints φ is defined as

$$\varphi := x \le c \mid c \le x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2$$

where x is a clock in C and c is a constant in $\mathbb{R}$, the set of nonnegative rational numbers. ■
We say that a clock interpretation v for C satisfies a clock constraint φ over C if and only if φ evaluates to true according to the values given by v. For $\delta \in \mathbb{R}$, $v+\delta$ denotes the clock interpretation which maps every clock x to the value $v(x)+\delta$. For $Y \subseteq C$, $v[Y:=0]$ denotes the clock interpretation for C that assigns 0 to each $x \in Y$, and agrees with v over the rest of the clocks.
Timed Automaton (TA) is finite state systems with some clocks, and the formal definition for TA is as follows:

Definition 3 Timed Automata
A timed automaton is a tuple$<L, L^0, \Sigma, E, C, Inv, Gad, Rst>$ with

- L, a finite set of locations with initial locations set $L^0 \subseteq L$
- Σ, a finite set of labels
- $E \subseteq L \times \Sigma \times L$, a set of edges (also called switches)
- C, a finite set of clocks
- $Inv: L \to \Phi(C)$, a function that assigns to each locations $l \in L$ an invariant $Inv(l)$
- $Gad: E \to \Phi(C)$, a function that labels each edge $e \in E$ with a clock constraint Gad(e) (called *guard*) in $\Phi(C)$ over C, and
- $Rst: E \to 2^C$, a function that assigns to each edge $e \in E$ a set of clocks Rst (e).

$L \times \Sigma \times 2^C \times \Phi(C) \times L$ is a set of switches. A switch $<s, a, \varphi, \lambda, s'>$ represents a transition from location s to location s' on symbol a, if $e=(s, a, s')$ is in E, $\varphi = Inv(s) \wedge Gad(e)$ is enabled, and $\lambda = Rst(e)$. ■
Definition 3 is a little different from that given in [8] to some extent. Using our definition here, it is easy to generate φ automatically for every switch $<s, a, \varphi, \lambda, s'>$ using the conjunction of Gad and Inv.
The semantics of a TA A is defined by associating a transition system S_A with it.
Definition 4 Transition System
Transition System is a tuple $<Q, Q^0, \Sigma, \to>$, where

- Q is a set of states,
- $Q^0 \subseteq Q$ is a set of initial states,
- Σ is a state of labels (or events), and
- $\to \subseteq Q \times \Sigma \times Q$ is a set of transitions.

For a transition $<q, a, q'>$ in $\to$, we write $q \xrightarrow{a} q'$. The system starts in an initial state, and if $q \xrightarrow{a} q'$ then the system can change its state form q to q'. We write $q \to q'$ if $q \xrightarrow{a} q'$ for some label a. $q \to^* q'$ denotes that state q' is reachable from the state q. ■
Thus, a state of S_A is a pair (s, v) such that s is a location of TA A and v is a clock interpretation for clock set C such that v satisfies the invariant $Inv(s)$. The set of all states of A is denoted Q_A. A state (s, v) is an initial state if s is an initial location of A and $v(x) = 0$ for all clocks x. There are two types of transitions in S_A :

- State can change due to elapse of time: for a state (s, v) and a real-valued time increment $\delta \ge 0$, $(s, v) \xrightarrow{\delta} (s, v+\delta)$ if $\forall \delta', 0 \le \delta' \le \delta, v+\delta'$ satisfies the invariant $Inv(s)$,
- State can change due to a locations-switch: for a state (s, v) and a switch $<s, a, \varphi, \lambda, s'>$ such that v satisfies φ, $(s, v) \xrightarrow{a} (s, v[\lambda := 0])$.

Based on this point, 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).

3. Region Graph Construction

3.1 Clock Region and Its Characterization

For any $\delta \in \mathbb{R}$, $fr(\delta)$ denotes the fractional part of δ, and $\lfloor \delta \rfloor$ denotes the integral part of δ; that is $\delta = \lfloor \delta \rfloor + fr(\delta)$. For each clock $x \in C$, let c_x be the largest integer c such that x is compared with c in the some clock constraint appearing in an invariant or a guard. Base on these notations, Rajeev Alur gave the formal definition about region equivalence $\cong$, which is an equivalent relation and defined over the set of all clock interpretations for C:

Definition Region Equivalence [8] (inaccurate)
For two clock interpretations v and v', $v \cong v'$ iff all the following conditions hold:

- For all $x \in C$, either $\lfloor v(x) \rfloor$ and $\lfloor v'(x) \rfloor$ are the same, or both $v(x)$ and $v'(x)$ exceeds c_x.
- For all $x, y \in C$ with $v(x) \le c_x$ and $v(y) \le c_y$, $fr(v(x)) \le fr(v(y))$ iff $fr(v'(x)) \le fr(v'(y))$.
- For all $x \in C$ with $v(x) \le c_x$, $fr(v(x)) = 0$ iff $fr(v'(x)) = 0$. ■

Unfortunately, some faults are found if we investigate the above definition thoroughly. For example, considering two clock interpretations over $C = \{x\}$ with $v(x) = c_x + 0.5$ and $v'(x) = c_x$, according to the definition, $v \cong v'$ holds because the antecedent $v(x) \le c_x$ in the third condition is false, but $v' \cong v$ does not. This fault was captured by modeling and verifying this inaccurate definition with PVS. Now we give the following correct definition:

Definition 5 Region Equivalence

For two clock interpretations v and v', $v \cong v'$ iff all the following conditions hold:

- For all $x \in C, v(x) \le c_x$ iff $v'(x) \le c_x$.
- For all $x \in C$ with $v(x) \le c_x$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$.
- For all $x, y \in C$ with $v(x) \le c_x$ and $v(y) \le c_y$, $fr(v(x)) \le fr(v(y))$ iff $fr(v'(x)) \le fr(v'(y))$.

For all $x \in C$ with $v(x) \le c_x$, $fr(v(x)) = 0$ iff $fr(v'(x)) = 0$. ■

A clock region for a TA A is an equivalence of clock interpretations induced by $\cong$. [v]denotes the clock region to which v belongs. Each clock region can be uniquely characterized by a (finite) set of clock constraints it satisfies. The number of clock regions is finite. For a clock constraint φ of TA A, if $v \cong v'$ then v satisfies φ iff v' satisfies φ.

Each region can be represented by specifying

(1) for every clock x, one clock constraint from the set

$\{x=c \mid c=0,1,..,c_x\} \cup \{c-1 < x < c \mid c=1,..,c_x\}$
$\cup \{x > c_x\}$

(2) for every pair of clocks x and y such that $c-1<x<c$ and $d-1 < y < d$ appear in (1) for some c, d, whether $fr(x)$ is less than, equal to, greater than $fr(y)$.

Region equivalence relation $\cong$ over the clock interpretations can be extended to an equivalence relation over the state-space by requiring equivalent states to have identical locations and region-equivalent clock interpretations: $(l,v) \cong (l',v')$ iff $l = l'$ and $v \cong v'$. Because the stability property [8] of region-equivalence relation.

Now we consider the evolution of a clock interpretation v due to elapse of time. As time elapses in a TA, v moves along the diagonally upwards direction since all the clocks have the same rate. Because all the cock interpretations in one clock region has the same timed behavior, we choose one as a representative and postulate it has only one time-successor.

Definition 6 **Region's Time Successor [1]**

A clock region α' is a time successor of a clock region α iff for each $v \in \alpha$, there exists a $t \in \mathbb{R}$ such that $v' = v + t \in \alpha'$. ■

For an arbitrary clock region [v], $\forall x$, $v(x) > c_x$, we call it a bound-exceeded region. A bound-exceeded region is a time successor of itself. If [v] isn't a bound-exceeded and $\forall x$, $v(x) \le c_x \Rightarrow fr(v(x)) = 0$, we call it a point region and postulate t in Definition 6 is 0.5. If $\forall x,y$, $v(x) \le c_x$ & $v(y) \le c_y \Rightarrow fr(v(x)) = fr(v(x)) \ne 0$, we call it a diagonal region and postulate t in Definition 6 is 1- $fr(v(x))$. Other clock regions forms a polyhedron in the n-dimensional euclidean space in general, we definition m = 1 - $max\{fr(v(x)) \mid v(x) \le c_x\}$, and if $\exists x$: $v(x) \le c_x$ & $fr(v(x)) = 0$, then we call this kind of region polyhedron region I and postulate $t = m/2$; and the last kind of clock region is called polyhedron region II and $t = m$. These can be summarized in Table 1. We call this time successor computed via the above algorithm *next* clock region, and denote [v'] = [*next*(v)] if v' is the time successor of v.

3.2 Constructing Reachable Region Graph for TA

Because the region equivalence $\cong$ over clock interpretation is stable, i.e., a bisimulation of the time-abstract transition system [8], we can construct a reachable region graph for a given TA with the aid of Table 1.

Definition 7 reachable region automaton

For a TA A= <L, L^0, Σ, E, C, Inv, Gad, Rst>, Let R_A be its transition system and of the form of Definition 4, called region automaton.

- The states of R_A are of the form <s,α> where s$\in L$, and α is a clock region.
- The initial states are of the form <s, [v_0]> where $s_0 \in L_0$, and $v_0(x) = 0$ for all $x \in C$.
- State can change due to (1) a *next* step, i.e., $(s,[v]) \xrightarrow{t} (s,[next(v)])$ if both v and v' satisfies the invariant $I(s)$ and t is the same as that in Table 1, or (2) a locations-switch: for a state (s, [v]) and a switch $<s,a,\varphi,\lambda,s'>$ such that v satisfies φ, $(s,[v]) \xrightarrow{\alpha} (s,[v[\lambda := 0]])$.■

For a given TA A, the reachable region automaton R_A given in Definition 7 and S_A given in Sections 2 are bisimilar as transition systems, the proof can be found in [7]. Therefore, some properties of TA A can also be investigated over R_A. A location in target locations $L^F \subseteq L$ of TA A is reachable in S_A iff exists an equivalence class π of $\cong$ such that π contains a state whose location is in L^F. Therefore, the reachability problem can be solved in R_A rather than in S_A.

Table 1. Classification of Clock Region

Category No.	Category Name	Characterization	t in Definition 6	Next region Category
E	bound-exceeded region	$\forall x, v(x) > c_x$	0	E
P	point region	$\forall x, v(x) \le c_x \Rightarrow fr(v(x)) = 0$	0.5	D or U
D	diagonal line region	$\forall x,y, v(x) \le c_x \,\&\, v(y) \le c_y \Rightarrow fr(v(x)) = fr(v(x)) \neq 0$	$1\text{-}fr(v(x))$, where $v(x) \le c_x$	P
PI	polyhedron region I	$(\exists x: v(x) \le c_x \,\&\, fr(v(x)) = 0)$ & $(\exists x, v(x) \le c_x \,\&\, fr(v(x)) \neq 0)$	$m/2$	E or PII
PII	polyhedron region II	$\forall x, v(x) \le c_x \Rightarrow fr(v(x)) \neq 0$	m	PI

Note: (1) the characterization for Categories P, D and PII omit the conjunction $\exists x: v(x) \le c_x$.

(2) $m = 1 - max\{fr(v(x)) \mid v(x) \le c_x\}$

(3) Categories PI and PII may be degraded to P and D respectively if the dimension decreases. So this classification is not disjoint pairwise.

4. Implementation in PVS

4.1 PVS

PVS [4] is a specification and verification environment developed by SRI International's Computer Science Laboratory. It 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. In distinction to other widely used proof systems, such as HOL and the ACL2 theorem prover, PVS supports both a highly expressive specification language and an interactive theorem prover in which most low-level proof steps are automated. The system consists of a specification language [5], a parser, a type checker, and an interactive proof checker [6]. The PVS specification language is based on a richly typed higher-order logic that permits a type checker to catch a number of semantic errors in specifications. The PVS prover consists of a set of inference steps that can be used to reduce a proof goal to simpler sub goals that can be discharged automatically by the primitive proof steps of the prover. The primitive proof steps incorporate arithmetic and equality decision procedures, automatic rewriting, and BDD-based boolean simplification.

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 proof steps. In addition to primitive proof steps, PVS supports more complex proof steps called strategies, which can be invoked just like any other proof step. Strategies may be defined using primitive proof steps, applicative Lisp code, and other strategies, and may be built-in or user-defined. It is very convenient for us to formalize a system and prove the corresponding theorems.

4.2 Preliminaries

Some preliminaries, i.e., some underlying theories for characterizing clock region, must be firstly developed.

Figure 2 gives the PVS specification about clock region equivalence, where *fr* is the abbreviation of prelude function [11] *fractional*. The theory *Region_equiv* is based on our existing works [13] which include the theories *Time*, *clock interpretation* and *clock constraints* etc. The functions *exb*, *point, diag, ply1* and *ply2* characterize the classifications for clock region shown in Table 1 respectively. The recursive function is used to calculate the maximum fractional part over the clocks whose values don't exceed their bound in a clock interpretation. The function *next* is defined according to Table 1. The formulae *lem44_1~44*, which are copied from the lemma 44 in [6], formally depict some properties of clock region.

4.3 Region Automaton theory in PVS

We have some theories and templates about modeling TA system in [13]. Here we suppose the TA model PVS specification has been established. That is to say, we have owned the definitions for TYPE *States*, *Runs*, time-delay step *delta* function and locations-switch transition *locswitch* between two States, etc.

Based on the theory *Region_equiv*, we can construct the reachable region graph for a given TA structure with PVS. We restrict the time-elapse step instead giving the region graph explicitly in the PVS specification shown in Figure 3. In other words, the transition system starts from the initial states, and each transition is triggered by the *RStep* function defined over two states.

In this theory, we can carry out the reachability states analysis, i.e. , encoding a search algorithm in PVS.

4.4 Reachability Analysis

A location s of timed automaton *A* is said to reachable if some state if some state *q* with location component *s* is reachable in the corresponding region automaton R_A. Because the finity of the region space, it is possible to transverse using some search algorithm.

In order to implement this function, a new type *SL* defined as *list[States]* is firstly introduced. This type of variable or constant can be used to restore a list of states which are in a certain order. An overloaded *member* function for TYPE *SL* is recursively defined with respect to region equivalence, that is to say, a state *s* belongs to a list of states *sl* iff *s* is equal to the head element of *sl* or belongs to its tail list in the essence of region equivalence. The *succ* function is used to compute the entire successor states for a single state, where *AIdx* is a function that maps a natural number which is below *NA,* the number of actions, onto an action. This definition is based on the fact that the next state is achieved by a delay transition or an action in R_A. The reachable state set of TGC system can be acquired by BFS (breadth first search) or DFS (depth first search) with the aid of *succ*. Here a sample BFS algorithm *BFS* shown in Figure 4 is given using recursive function, which

```
RegionAutomaton:    THEORY BEGIN
…. % Importing a TA Model PVS specification, so we have the definition
      % about TYPE and the transition function, as well as the TA structure.
s, s1, q, q1, u, u1 : VAR States;
A: VAR Actions
IMPORTING region_equiv[N, c];
 ~(s, s1): bool = loc(s) = loc(s1) AND v(s) ~ v(s1)
bisem(q, u): bool = ∀q1: q ~ u & Step(q, q1) ⇒
            (∃u1: Step(u, u1) & q1 ~ u1);
Stable: THEOREM bisem(q, u)
nd(s): States = s with [v := next(s`v)]
locswitch(s, A): States =(# loc := Edge(s0`loc, A), v := reset(s0`v)(ResetC(A)))
RStep(s,s1) : bool = s`v |= Inv(s`loc) & s1`v |= Inv(s1`loc) &
    (s1 = nd(s) OR (∃A: s1 = locswitch(s, A) AND s`v |= Guard(A))
…
END RegionAutomaton
```

Figure 3. The Theory Region Automaton

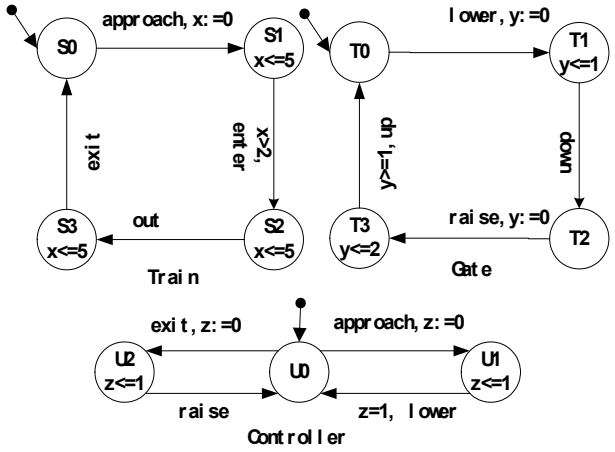
```
RegionAutomaton:    THEORY BEGIN
      ….
s, s1, q, q1, u, u1 : VAR States;
SL: TYPE = list[States]
member(s, (l: SL)): RECURSIVE bool =
   CASES l OF null: FALSE, cons(hd, tl): s ~ hd OR member(s, tl) ENDCASES
      MEASURE length(l)
succ(s)(i: upto[NA]): RECURSIVE SL = IF i = NA THEN null[States]
       ELSE LET    A = AIdx(i), s1 =locswitch(s, A) IN
        IF (s`v |= (Inv(s`loc) AND Guard(A)) & s1`v |= Inv(s1`loc) THEN cons(s1,
          succ(s)(i + 1))    ELSE succ(s)(i + 1)    ENDIF
      ENDIF MEASURE NA - i
succ(s): SL = LET s1 = nd(s), sa = succ(s)(0) IN
       IF ((s`v |= Inv(s`loc)) & (s1`v |= Inv(s`loc))) THEN cons(s1, sa)
          ELSE sa ENDIF
BFS(sl, vl: list[States]): RECURSIVE list[States] = CASES sl
         OF null: reverse(vl),
            cons(s, l): IF member(s, vl) THEN BFS(l, vl)
               ELSE BFS(append(l, succ(s)), cons(s, vl)) ENDIF
         ENDCASES MEASURE …
initS: States = (# loc :=..., v := zero #)
reachable(s) : bool = member(s, BFS((: initS :), (: :)))
NonDeadlock : THEOREM reachable(s) & (NOT null?(succ(s)))
END RegionAutomaton
```

Figure 4. Reachability Analysis via BFS

Figure 5. Train-gate controller

is implemented by maintaining a visiting and a visited states list. The recursive process must be terminated because of the finity of the region space. Based on these works, the system behaviors can be investigated. For example, as a correct and robust real-time system, the theorem *NonDeadlock* about deadlock-freedom, which claims for all the reachable states, there must exist at lest one successor must be ensured, is formalized in this theory. Once the reachable states are calculated, the safety property, which asserts some *bad* states will never be entered, can be verified.

We have tested this PVS specification via investigating a case study TGC (Train-Gate-Controller) system [8] shown in Figure 5, about which modeling details can refer to [13]. Using some interactions in PVS prover, we can prove that the reachable locations for TGC system are the set {*S0T0U0, S1T0U1, S1T1U0, S1T2U0, S2T2U0, S3T2U0, S0T2U2, S0T3U0, S1T3U1*}, where *Si, Tj* and *Uk* represent the corresponding location for the three components Train, Gate and Controller respectively. Therefore, we can assert that the TGC system is safe because the unsafe states (with the location component *S2* for Train and non-*T2* for Gate respectively) are not in the reachable states.



5. Related Works and Discussions

One of the research work about formal verification for real-time system based-on timed automata model using theorem proving method had been conducted in TAME (Timed Automata Modeling Environment) project [2] in US Naval Research Laboratory. TAME provides mechanical assistance that allows humans to specify and reason about real-time systems in a direct manner. Nevertheless, TAME doesn't supply some operation on clocks. Therefore, it is inconvenient and unnatural to use TAME especially in proving some properties about clocks of TA. Another research work, which is conducted by Jozef Hooman [9], gives the TA model theory and some lemmas about the corresponding runs of TA in PVS. Our previous work is a further development and application of [9], we give the explicit definition for Runs of TA and show how to construct product TA in PVS [12]. The works in [13] can reduce the modeling loads for TA and its corresponding operations in PVS to some extent.

The common shortcoming for both TAME and the works of Jozef Hooman was that they didn't provide the manipulation for the clocks, a very important notion in TA theory. Therefore, some operations on TA, such as clock region-equivalence or time-abstract application in terms of clocks interpretation, and the formal verifications of some properties defined over the clock variables became unnatural and difficult.

We don't intend to compare the efficiency of *BFS* function shown in Figure 5 with that of the other known model checkers such as UPPAAL [14], Kronos [15] etc., because our algorithm is implemented in PVS and others are implemented using programming language, and therefore they are not comparable in efficiency. But the formalization to BFS can be used to verify correctness of some search algorithm. Furthermore, this method can be looked upon as a basis for detecting reachable states or locations for TA after some improvements such that it can be automatically carried out in the PVS prover. For examples, developing some special strategies for FVofTA should be helpful.

6. Conclusions and Future Works

Based on the our existing works [14], this paper firstly gives clock region equivalence PVS specification, and then constructs the reachable region graph for a given TA via characterizing some kinds of clock regions, finally analyses the reachable states using this graph. These works can conveniently analysis some real-time system in the form of TA model. As a by-product, an error is detected in the region-equivalence definition which is extensively referred in many papers.

The future works about the further extensions for FVofTA mainly include the following parts:

(1) We want to establish the TTL (Timed Temporal Logic) framework based on the TA modeling theory, and to ensure that TTL formula can be verified using our clock manipulation theories.

(2) The efficiency of model checking algorithms such as BFS must be improved in PVS because PVS is used as theorem prover assistant rather than algorithm programming language despite that the correctness for them will be ensured by the PVS proof system.

(3) Other model checking algorithms, such as checking the TA's emptiness etc., are considered to be developed in our system in the future.

References

[1] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science* 126:183-235, 1994.

[2] M. Archer and C. Heitmeyer. TAME: A specialized specification and verification system for timed automata. *1996 IEEE Real-Time Systems Symp*., Washington, D.C., 1996, 3-6.

[3] Thomas A. Henzinger and Vlad Rusu. Reachability verification for hybrid automata. *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control* (HSCC), Lecture Notes in Computer Science 1386, Springer, 1998, pp. 190-204.

[4] Owre S, Shankar N, Rushby J, Stringer-Calvert D. PVS system guide version 3.2. Computer Science Laboratory, SRI International, September, 2004, 1–95.

[5] Owre S, Shankar N, Rushby J, Stringer-Calvert D. PVS language reference version 3.0. Computer Science Laboratory, SRI International, February, 2003, 1–123.

[6] Shankar N, Owre S, Rushby J, Stringer-Calvert D. PVS prover guide version 3.2. Computer Science Laboratory, SRI International, September, 2004, 1–128.

[7] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. Cambridge: MIT PRESS, 1999, 265-292.

[8] R. Alur. Timed Automata. *NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems.*

[9] Jozef Hooman. Timed Automata in PVS. *Summerschool Zhengzhou*, 2004.

[10] Owre S, Shankar N. Abstract Datatypes in PVS. Computer Science Laboratory, SRI International, 1997,1-52.

[11] Owre S, Shankar N. The PVS Prelude Library. Computer Science Laboratory, SRI International, 2003,1-31.

[12] Qingguo XU, Huaikou MIAO. Formal Verification Framework for safety of Real-time System Based-on Timed Automata Model in PVS, the IASTED International Conference on Software Engineering, Innsbruck, Austria, February 12-14, 2006, 107-112, ACTA Press 2006.

[13] Qingguo XU, Huaikou MIAO: Modeling Timed Automata Theory in PVS. Proceedings of the International Conference on Software Engineering Research and Practice, SERP, Las Vegas, Nevada, USA, June 27-29, 2006. CSREA Press 2006.

[14] Johan Bengtsson, Wang Yi. Timed Automata: Semantics, Algorithms and Tools, Johan Bengtsson and Wang Yi. In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag 2004.

[15] S. Yovine. Kronos: A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Issue 1/2, pages 123-133, October 1997.

Huaikou Miao received the M.Sc degree from Shanghai University of Science and Technology in 1986. After working as a lecturer (from 1986) and an associate professor (from 1990) in the Department of Computer Science, Shanghai University of Science and Technology, and an associate professor (from 1994) in the School of Computer Engineering and Science, Shanghai University. He has been a professor at Shanghai University since 1997. His research interest includes formal methods and software engineering. He is a member of IEEE Computer Society.

Qingguo Xu received the B.S. and M.S. degrees in Chemical Engineering from East China University of Metallurgy, (now named Anhui University of Technology) in 1995 and 1998, respectively. During 1998-2002, he worked as a research assistant, a lecturer (from 2001) in system engineering research laboratory in Qingdao Institute of Chemical Technology (now named Qingdao University of Science & Technology). He now is a PHD. candidate in the School of Computer Engineering and Science, Shanghai University. His main research interest is about the formal verification and validation of real-time system.