实时系统形式规格说明在 PVS 中的建立*)

许庆国　缪淮扣
(上海大学计算机学院　上海 200072)

摘　要　本文首先简介了时间自动机和时间 Büchi 自动机形式模型，结合时间化时序逻辑(Timed Temporal Logic)的语法和语义，利用定理证明器 PVS(Prototype Verification System)实现了定义在时间自动机状态(或运行)上的时间化分支(或线性)时序逻辑规格说明的形式体系。在此基础上，结合一个经典的实时系统实例，用该体系对其实时特性进行了形式描述和形式验证，并得到了良好的结果。

关键词　实时系统，时间 Büchi 自动机，时间化时序逻辑，PVS

Formalizing Real-time Specification Using the Timed Temporal Logic in PVS

XU Qing-Guo　MIAO Huai-Kou
(School of Computer Engineering & Computer Science, Shanghai University, Shanghai 200072)

Abstract　On the Basis of the formal model of the timed automata, timed Büchi automata and the formal syntax and semantics of the timed temporal logic, some formal theories about timed branch (linear) temporal logic are implemented over the states (runs) of the timed automata using PVS (Prototype Verification System) in this paper. A well-known real-time system model by timed automata and its real-time specifications are investigated using these theories. The results show the formal verification of real-time specification using the formalization in this paper is effective.

Keywords　Real-time, Timed temporal logic, Timed büchi automata, PVS

1　前言

时间自动机[1]TA (Timed Automata)是对实时系统(Real-Time System)建模很自然的工具和方法，研究人员提出了用时间自动机对实时系统进行形式建模和形式验证的不少方法。这些方法给开发人员确保满足所需要的特性(尤其是安全性)提供了很大的信心。形式验证是保证实时系统正确的严格的数学方法，可以分为两大类，一是模型检查的方法[2]，二是定理证明的方法。模型检查的方法是用时间自动机对实时系统建模，将系统应该满足的特性描述为时间化的时序逻辑公式，然后利用特定的模型检查算法进行系统的全状态空间搜索来判定，这不可避免地受到系统的状态空间大小的限制。若采用定理证明的方法，可从根本上克服状态爆炸的问题，但这一方法也需要有相应的时序逻辑特性的形式体系予以支持，才能使实时特性的描述能够比较完备并避免其中的不一致性[2]。

在以前的工作[3]中，我们提出基于自动机建模的实时系统的形式证明框架 FVofTA(Formal Verification of Timed Automata)，并做了一些基础性的工作。该框架如图 1 所示。

在工作[3]中，我们已经实现了完成基于时间自动机的实时系统的建模和形式分析，也就是图 1 中的下面 4 个方框。本文将主要完成"实时系统的特性描述方法"的工作。

实时特性的描述方法文献上的描述方法有许多种。为了框架中验证的方便，我们采用时间化的线性时序 TLTL (Timed Linear Temporal Logic)和时间化的分支时序 TBTL (Timed Branch Temporal Logic)描述的方法，线性的用时间 Büchi 自动机 TBA(Timed Büchi Automata)来表示，这是因为时间自动机的运行的定义正好可以看作是带有时钟赋值的命题序列，刚好与 TLTL 对应，而分支的则用 TCTL(Timed Computational Temporal Logic)来表示。

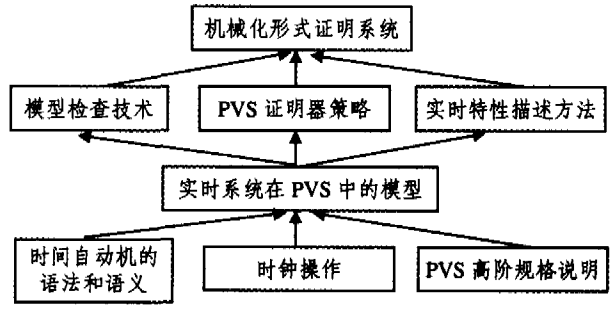

图 1　时间自动机的形式验证框架

本文的其余部分如下安排：第 2 节在时间自动机基础上，给出 Büchi 自动机 BA(Büchi Automata)的实时性扩充时间，Büchi 自动机 TBA(Timed Büchi Automata)的语法和语义，它将用于描述线性时序特性。第 3 节给出分支时序逻辑 CTL (Computational Temporal Logic)的实时性扩充 TCTL(Timed CTL)，用它来描述分支时间时序特性。第 4 节用 PVS(Prototype Verification System)[4] 建立实时特性语法的形式表述，而后基于时间自动机建立其语义模型。这将为用时间自动机建模的实时系统进行形式验证建立良好的基础。第 5 节结合一个实时系统的实例，在我们的形式体系中来考查其中的实时特性和形式验证。最后是相关工作和结论。

2　时间 Büchi 自动机及其对线性实时特性的描述

用于描述实时系统规格说明的时间自动机[1]是 Rajeev

*)本文受国家自然科学基金项目(批准号 60173031)和国家 973 项目(编号：2002CB312001)资助。

Alur 和 David L. Dill 首先提出来的，为了在自动机模型中很好地建模时间行为，首先引入在时间自动机语法定义中用到的时钟约束和时钟解释的形式定义[8]：

定义 1(时钟约束) 对于有理数集合 $\cdot$ 中任一数值可认为是一个时钟常量，对于一个时钟变量的有穷集合 X，时钟约束的集合 $\Phi(X)$ 的形式语法为：

$\varphi:=x\leqslant c|c\leqslant x|x<c|c<x|\varphi\wedge\varphi$

其中 x 是一个时钟，而 c 是 $\cdot$ 中的一个常量。

定义 2(时钟解释) 关于时钟集合 X 的时钟解释 v 定义为集合 X 到非负实数集 $\cdot$ 的一个映射，即，为该集合中的每一时钟赋予一个实数值。

对于时钟集合 X，我们说一个时钟解释 v 满足 X 上的一个时钟约束 φ，当且仅当根据 v 给出的值求 φ 值为真。对于 $\delta\in\cdot$，$v+\delta$ 表示将每一时钟 x 映射为值 $v(x)+\delta$ 的时钟解释。对于 $Y\subseteq X$，$v[Y:=0]$ 表示将每一 $x\in Y$ 赋值为 0、而每一 $x\in X\backslash Y$ 同 v 保持一致的时钟解释。

时间自动机是包含时钟的有穷状态系统，为了形式化建模和验证的方便，我们根据文[8]进行了一些变换。

定义 3(时间自动机的语法) 时间自动机是一个元组 $\langle L,L^0,\Sigma,E,C,Inv,Gad,Rst\rangle$，其中

- L，位置的有穷集合，其中初始位置集合 $L^0\in L$；
- Σ，标号(也称事件)的有穷集合；
- $E\subseteq L\times\Sigma\times L$，边的集合(也称转换)；
- C，时钟的有穷集合；
- $Inv:L\to\Phi(C)$，为每一位置 $l\in L$ 赋予一个不变式 $Inv(l)$ 的函数；
- $Gad:E\to\Phi(C)$，为每一条边 $e\in E$ 标以一个时钟约束 $Gad(e)$(称为卫式)，$Gad(e)$ 是时钟集合 C 上时钟约束 $\Phi(C)$ 中的元素；
- $Rst:E\to 2^C$，为每一条边 $e\in E$ 赋予一个要重置的时钟集合 $Rst(e)$。

$L\times\Sigma\times 2^C\times\Phi(C)\times L$ 是转换的一个集合，如果 $e=(l,a,l')$ 是 E 中的元素，$\varphi=Inv(l)\wedge Gad(e)$ 为真，并且 $\lambda=Rst(e)$，则集合中的元素 $\langle l,a,\varphi,\lambda,l'\rangle$ 表示从位置 l 到位置 l' 通过符号 a 的一个转换。

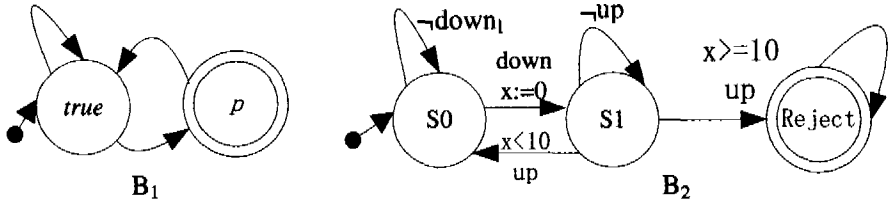

图 2 (时间)Büchi 自动机表示(实时)特性的例子

TA A 的状态包括控制位置和相应的时钟值，这形成了一个元素对 $s=(l,v)$，其中 l(可用 $l(s)$ 来存取之)是 A 的位置，v(可用 $v(s)$ 来存取之)是使得不变式 $Inv(l)$ 满足的时钟集合 X 上的一个时钟赋值。如果 $l\in L^0$，且对所有的时钟 x 有 $v(x)=0$，则状态 (l,v) 是系统的初态。为方便起见，我们用 $t(s)$ 表示状态 s 的全局时间，显然，初态 s_0 的全局时间 $t(s_0)=0$。

定义 4(时间自动机的语义) TA A 中有两种类型的转换：

- 因时间的流逝引起的状态变化(相应的时间自动机的位置不变)：对于状态 (l,v) 和非实数值的时间增量 $\delta\geqslant 0$，如果 $\forall\delta,0\leqslant\delta'<\delta,v+\delta'$ 满足 $Inv(l)$，则有 $(l,v)\xrightarrow{\delta}(l,v+\delta)$；
- 由于位置迁移引起的状态改变(相应的时间自动机中的所有时钟值不变)：对于状态 (l,v) 和一个迁移 $\langle l,a,\varphi,\lambda,l'\rangle$，使得 $(l,v)\xrightarrow{a}(l',v[\lambda:=0])$。

时间自动机 A 的运行，即，A 所接受的语言 $Lang(A)$，定义为开始于初态的状态序列的集合，因建模实时系统可执行的需要，这些状态序列还需满足时间发散性(Nonzenoness)的需求：

(1)当违反了一个位置的不变式时钟约束时，必须能够执行转出该位置的转换以转移到其它位置。

(2)对每一可达状态，自动机应当允许时间发散，即允许实时系统永远运行。

为了增加对于系统特性的描述能力和描述时的方便，实时系统的线性特性可用 TBA 来表示。

定义 5(时间 Büchi 自动机[18]) 一个 TBA 是一个元组 $B=\langle A,F\rangle$，其中 $A=\langle L,L^0,\Sigma,E,C,Inv,Gad,Rst\rangle$ 是一 TA，$F\subseteq L$ 是一个重复的位置集合。TA 的状态、转换和运行的概念可以很容易地扩充到 TBA 中，如果 B 的一个运行 ρ 中有无穷多个重复的位置，即对于任意的 i，总是存在 $j>i$ 使得运行 ρ 中的第 j 个元素 $\rho(j)\in F$，则称运行 ρ 是可接受的(accepting)。B 的语言，表示为 $Lang(B)$，是所有可接受的并且具有进展性的运行的集合。

例如图 2 所示的 BA 和 TBA 分别可以表示活性(liveness)"系统最终要进入 p 状态"和有界响应性(bounded response)"系统在发生 down 动作后的 10 个时间单位内一定要发生 up 动作"。

定义 6(TA 与 TBA 的积) 令 TA $A=\langle L,L^0,\Sigma,E,C,Inv,Gad,Rst\rangle$，TBA $B=\langle L',L^{0\prime},\Sigma',E',C',Inv',Gad',Rst',F\rangle$，则二者的积 $A\times B$ 定义为 TBA$\langle L\times L',L^0\times L^{0\prime},\Sigma\cup\Sigma',E'',C\cup C',Inv'',Gad'',Rst'',F''\rangle$，其中 $Inv''(l,l')=Inv(l)\wedge Inv'(l')$，$Gad''(l,l')=Gad(l)\wedge Gad'(l')$，$Rst''(l,l')=Rst(l)\cup Rst'(l')$，$F''=L\times F$，$E''$ 定义为：

- 对每一 $e=(l_1,a,l_2)\in E$ 和 $e'=(l'_1,a,l'_2)\in E'$，若 $a\in\Sigma\cap\Sigma'$，则 E'' 包括 $e''=((l_1,l'_1),a,(l_2,l'_2))$。
- 对每一 $e=(l_1,a,l_2)\in E$，若 $a\in\Sigma\backslash\Sigma'$，则对于 L' 中的每一位置 t，E'' 包括 $e''=((l_1,t),a,(l_2,t))$。
- 对每一 $e'=(l'_1,a,l'_2)\in E'$，若 $a\in\Sigma'\backslash\Sigma$，则对于 L 中的每一位置 t，E'' 包括 $e''=((t,l'_1),a,(t,l'_2))$。

考虑一个实时系统可以建模为 TA A，该系统应该满足的特性 φ 的否定 $\neg\varphi$ 可以描述为 TBA $B_{\neg\varphi}$。系统 A 满足特性 $B_{\neg\varphi}$，当且仅当 $Lang(A\times B_{\neg\varphi})$ 非空。反之，若 $Lang(A\times B_{\neg\varphi})$ 为空，则系统 A 满足特性 φ。这就为利用自动机理论进行特性形式验证提供了一条途径[2]。

3 时间自动机中的 TCTL

时间化的计算树逻辑 TCTL 是在文[6]中引入的，它是分支逻辑 CTL 的实时性扩充，定义在时间化的原子命题序列上。首先给出语言的语法和语义定义。

定义 7(TCTL 的语法) 令 $\cdot$ 表示非负实数集 $\cdot$ 上的形如$[c,c']$、$[c,c')$、$(c,c']$、(c,c')、(c,∞)和$[c,\infty)$的区间集合，其中 $c,c'\in\cdot$(非负整数集合)。则 TCTL 公式的语法递归定义如下：

$\varphi::=\text{true}\mid p\mid\neg\phi\mid\phi\vee\phi\mid\exists\phi\cdot_I\phi\mid\forall\phi\cdot_I\phi$

其中 $p\in Props$ 是一个原子命题，$I\in\cdot$ 是一个区间。

定义 8(TCTL 的语义) 令 A 是一 TA，其位置集合为 L，令 $P:Props\cdot 2^L$ 为将每一原子命题同 A 的位置相关联的函数。TCTL 公式解释在 A 的状态上。给定一个公式 ϕ 和一个状态 s，公式的满足关系 $s\models_P\phi$ 可以归纳定义如下(为简单起见，此处忽略了下标 P)：

$s\models\text{true}$

$s\models p$ 当且仅当 $l(s)\in P(p)$

$s\models\neg\phi_1$ 当且仅当 $s\cdot\phi_1$

$s\models\phi_1\vee\phi_2$ 当且仅当 $s\models\phi_1$ 或 $s\models\phi_2$

$s\models\exists(\forall)\phi_1\cdot_I\phi_2$ 当且仅当 $\exists(\forall)\rho=s_1s_2s_3\cdots,s=s_1$，$\exists i.t(s_i)\in I,s_i\models\phi_2$ 且 $\forall j.0\leqslant j<i,s_j\models\phi_1$

非形式地说，$s\models\exists\phi_1\cdot_I\phi_2$ 表示存在一条开始于状态 s 的运行，在运行的全局时间到达区间 I 内时的状态 s_i 时，ϕ_2 是成立的并且该运行在 s_i 之前的所有状态上 ϕ_1 成立。

为表示的方便，可以同时定义下面这些 TCTL 公式的缩写：

$\exists\Diamond_I\phi:=\exists\,true\cdot_I\phi$

$\exists\Diamond_I\phi:=\forall\,true\cdot_I\phi$

$\forall_I\phi:=\neg\exists\Diamond_I\neg\phi$

$\exists_I\phi:=\neg\forall\Diamond_I\neg\phi$

其中的区间 I 也可以缩写，如 $\exists\Diamond_{[0,5]}\phi$ 可以简写为 $\exists\Diamond_{\leqslant 5}\phi$，$\forall_{[0,\infty)}\phi$ 可以简写为 $\forall\phi$，等等。

如果一个 TA A 的初态满足公式 ϕ，则 A 满足 ϕ。

4 实时特性的描述在 PVS 中的实现

4.1 PVS 简介

PVS[4](Prototype Verification System)是开发和分析形式化规格说明的原型证明系统，支持涉及创建、分析、修改管理和文档化理论和证明的一大类活动。PVS 系统包含规格说明语言[5](类型化的高阶逻辑语言)及相应的语法分析器、类型检查器、证明器[7]、内嵌的规格说明库[13]、关于证明脚本及规格说明语言的浏览和编辑工具。PVS 规格说明语言是基于高阶逻辑(higher-order)的，这种语言非常适合用于描述计算机软件系统的行为[5]。PVS 有丰富的类型系统和相应的严格的类型检查器，通过提供有效的定理证明器来支持错误检测。PVS 证明检查器的主要特点是支持构造易读的证明，为了证明过程容易进行，PVS 证明器提供了强大的证明命令集，利用定义、引理、等价性和算术演算，可将这些证明命令结合成策略[7]。由于 PVS 的这些优点，有不少研究者在 PVS 中对时间系统的形式验证进行了部分工作[9~11]。

4.2 PVS 中的 TBA 理论

PVS 中的 TA 理论已经在文[3]中建立，TBA 与 TA 的主要不同在于其增加了接收位置及其相应对自动机所接受的语言(运行)的定义。我们可以将二者纳入统一的框架之中，一般的 TA 作为 TBA 的特例，它的接受位置是全集：

```
TBA[Locations, Actions: TYPE, N: nat,
  (IMPORTING clkinterp[N]) c: clock_bound,
  (IMPORTING clkp[N,c]) Init: pred[Locations],
  Inv:[Locations→clkp],
  Edge:pred[[Locations, Actions, Locations]],
  Guard:[(Edge)→clkp],
  ResetC:[(Edge) pred[clock]],
  Accept:pred[Locations]]: THEORY BEGIN
  States:TYPE =[# loc: Locations,
          v:clockInterpretation, now: Time #];
pr:VAR sequence[States]
t:VAR Time %非负实数
d: VAR PosTime %正实数
s,s0,s1:VAR States
i,j:VAR nat
x,y:VAR clock
A:VAR Actions
+(s,(t:Time)):States=s WITH
                [v:= s'v + t,now :=s'now+t]
delta(s0,d,s1):bool=s1=s0+d &
  (s0'v ⊨Inv(s0'loc))&(s1'v ⊨Inv(s1'loc))
locswitch(s0,A,s1):bool=
  LET sw=(s0'loc,A,s1'loc) IN Edge(sw)&
      (s0'v⊨(Inv(s0'loc)AND Guard(sw)))&
       s1'v=reset(s0'v)(ResetC(sw))&
       (s1'v ⊨ Inv(s1'loc))
Step(s0,s1):bool=(∃d:delta(s0,d,s1))OR
                 (∃A:locswitch(s0,A,s1))
Init(s):bool = Init(s'loc) & s'v=(λx:0)& s'now=0
NoTimeDecrease(pr):bool=
          ∀i:now(pr(i))≤now(pr(i+1))
NonZeno(pr):bool = ∀t:∃i:t< now(pr(i))
accept? (pr):bool=∀i:(∃j:j>i & Accept(pr(j)'loc))
run? (pr):bool=NoTimeDecrease(pr)& NonZeno(pr) &((i: Step
       (pr(i), pr(i+1)))
init_run? (pr:(run?)):bool=Init(pr(0))
accept_run? (pr:(run?)):bool= Accept? (pr)
Runs:TYPE ={pr:(run?)|Init(pr(0))}
r,w:VAR Runs
Lang:TYPE=Runs
lang_empty:bool = not ∃r:true
accept_empty:LEMMA (∀(r:(init_run?)):
     NOT Accept? (r))⇒lang_empty
     ……
```

上面规格说明中的 N 表示 TBA 中的时钟数量，clkinterp 和 clkp 是关于时钟解释和时钟约束的 PVS 理论，其它参数的意义可参见定义 3。

上面语言中的[#…#]表示定义了记录类型，其中的以“,”分隔的部分是其组成字段的各个分量，“'”是字段分量的存取算子[5]。

若将以上的 TBA 实例化为定义 6 中的 $A\times B$，则 A 满足 B 可以通过验证 lang_not_empty 来完成。验证的方法可以使用模型检查技术中的深度或宽度优先搜索算法。

4.3 PVS 中 TCTL 的形式体系

基于以上 TBA 的形式定义，根据定义 7，我们可以利用 PVS 的抽象数据类型[12]定义 TCTL 的语法，其中的函数 P 可以表示为 TA 中的位置 Locations 域上的谓词：

```
TCTL:DATATYPE BEGIN
    TRUE:true?
    Hold(p:pred[Locations]): Hold?
    NOT(f:TCTL): NOT?
    OR(f1:TCTL,f2:TCTL):AND?
    EU(f1:TCTL,I:interval,f2:TCTL):EU?
    AU(f1:TCTL,I:interval,f2:TCTL):AU?
  END TCTL
```

其中的数据类型 Interval 本身也定义一个抽象数据类型，一个非负实数是否属于一个区间，用 member 函数来判断。

```
interval:THEORY BEGIN
  interval:DATATYPE BEGIN
    cc(c0:nat,c1:upfrom(c0)):cc?
    co(c0:nat,c1:upfrom(c0)):co?
    oo(c0:nat,c1:upfrom(c0)):oo?
    oc(c0:nat,c1:upfrom(c0)):oc?
    cinf(c:nat):cinf?
```

```
    oinf(c:nat):oinf?
  END interval
member(r:Time,I:interval):bool=CASES I OF
      cc(c0,c1): r≥c0 & r≤c1,
      co(c0,c1): r≥c0 & r<c1,
      oo(c0,c1): r>c0 & r<c1,
      oc(c0,c1): r>c0 & r≤c1,
      cinf(c):r ≥ c,
      oinf(c):r >c
    ENDCASES
END interval
```

TCTL 公式的语义可以根据定义 8,定义在 TA 的状态上:

```
⊨(s: States, f: TCTL): RECURSIVE bool=
    CASES f OF
     TRUE: TRUE,
     Hold(p):(p(s'loc)),
     NOT(f1): NOT (s⊨f1),
     OR(f1,f2): (s⊨f1)OR (s ⊨ f2),
     EU(f1,I,f2):((r: (run?)):
       r(0)=s AND (i:(r(i)⊨f2)AND
         (∀(j:below(i)):(r(j)⊨ f1)) AND
                member(r(i)'now-s'now, I)),
     AU(f1,I,f2):(r:(run?)): … %省略的同 EU 部分 END-
CASES MEASURE f BY<<;
```

这里的<<是由 PVS 系统自动生成的 TCTL 语法的子项关系,即,若 f1<<f2,则说明 f1 是 f2 的子公式,这就保证了复杂的 TCTL 公式的验证可以归纳地用其子项进行验证。

在 PVS 中同样可以引入 TCTL 公式的简写:

```
AND(f1,f2):TCTL=NOT((NOT f1)OR(NOT f2))
EF(I,f): TCTL=EU(TRUE, I, f)
AU(f1,f2):TCTL=AU(f1, cinf(0), f2)
AG(f):TCTL=NOT EF( NOT f1)
    ……
```

为了 TCTL 公式的书写简单,可以省略 Hold,这需要使用 PVS 中的类型自动转换功能:

```
tctl(p: pred[Locations]): TCTL=Hold(p)
CONVERSION tctl
```

如果一个 TA 的初态满足一个 TCTL 公式 f,则该 TA 满足 f:

```
⊨(f):bool= ∀ s:(Init(s)⇒(s⊨f))
```

5 实例研究

5.1 TGC 系统

我们考虑一个典型的可用 TA 建模的实时系统及其特性。

图 3 建模了日常生活中的火车过岔道口的控制器[8],当一列火车到达铁路交叉口时,Train 发一个 approach 信号给 Controller,并在 2 个时间单位后进入交叉口,当火车离开交叉口时,Train 发一个 exit 信号给 Controller,exit 信号应在 5 个时间单位内发送;当 Controller 接收到 approach 信号后 1 个时间单位时,Controller 发送 lower 信号给 Gate,当接收到 exit 信号后,1 个时间单位内发送 raise 信号给 Gate;Gate 响应 lower,1 个时间单位内完成 down,响应 raise,2 个时间单位内完成 down。对该系统可用时间自动机建模为图 1 中的三个 TA:Train、Gate 和 Controller。本文称之为 TGC 系统。

根据文[3],我们可以对 TGC 系统在 PVS 中形式化。我们将在 3 个自动机一起建模。实例化 TBA 理论的参数为:

```
TLoc:TYPE={S0, S1, S2, S3}
GLoc:TYPE={T0, T1, T2, T3}
CLoc:TYPE={U0, U1, U2}
Locations:TYPE=[# t:TLoc,g:GLoc,c:CLoc #]
Actions:TYPE = {approach, up, down, enter, out, exit,
                lower, raise}
```

至于其它的参数,时钟数 $N=3$,Init、Inv、Guard、ResetC 和 Accept 等可根据图 3 相应实例化,最后用 PVS 语言中的 importing 子句得到 TGC 的理论模型:

```
IMPORTING TBA[Locations, Actions, N, c, Init, Inv,
Edge,Guard,ResetC,Accept]
```

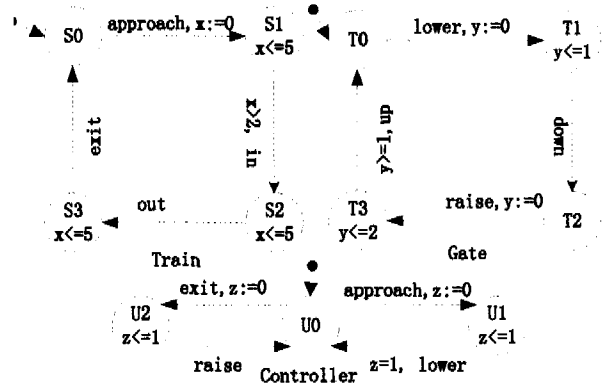

图 3 火车过岔道口的时间自动机模型

有了系统模型,可以来考虑系统的实时特性。系统的正确性要求,在火车通过交叉口时,安全门应该自动处于关闭状态。关闭一段时间后,应该自动打开以利于火车轨道垂直方向的车辆或行人通过,即,TGC 应该满足以下两则特性:

(safety)如果 Train 处于位置 s2(火车处于交叉口),则 Gate 一定处于位置 t2(安全门处于关闭状态)。(liveness)控制门处于关闭的时间不会超过 10 个时间单位。

我们先考虑特性 safety。

由上面的安全性验证的规格说明可以在 TGC 的 TA 理论中表示为:

```
safety:THEOREM⊨(AG(λl:l't=S2⇒l'g=T2))
```

用 PVS 证明器激发该公式的证明得:

```
safety:
|-------
{1}⊨(AG(tctl(λl:t(l)=S2⇒g(l)=T2)))
```

其中的 tctl 函数是 PVS 自动应用类型转换的效果。通过引入 skolem 常量,结合 TCTL 语法和语义的形式定义,运行 TCTL 公式的重写规则和满足关系"⊨"的定义,上面的公式可以等价地转换为:

```
[-1]Init(s! 1)
[-2]r! 1(0)=s! 1
|-------
{1}t(r! 1(i! 1)'loc)= S2⇒g(r! 1(i! 1)'loc)=T2
```

为证明该式,需要考察运行"r! 1"的性质,通过 PVS 证明器命令(typepred "r! 1")显式引入 r! 1 的类型,并展开相应的函数定义,并进行一定的技术处理,得:

```
[-1]∀ i: Step(r! 1(i),r! 1(1+i))
[-2]Init(s! 1)
[-3]r! 1(0)=s! 1
[-4]t(r! 1(i! 1)'loc)=S2
|-------
[1]g(r! 1(i! 1)'loc)=T2
```

在"i! 1"上进行归纳,可以证明上式[19]。

相对上面介绍的安全性表示和形式验证而言,该系统的实时活性的处理要复杂一些。特性 liveness"控制门处于关闭的时间不会超过 10 个时间单位"表示的是进展性,是定义在

一个状态序列上的，因此这是一个线性特性，不能用 TCTL 公式来表示，而要用时间化的线性时序特性规格说明语言(参见定义 5)来描述。为了验证的方便，由于活性的否定形式是安全性[14]，因此，我们验证系统中的该特性的否定形式是不可满足的。liveness 的否定可用图 2 中的 TBA B_2 来表示，这样一来，该特性就可以通过对 B2 建模为 PVS 的 TBA 理论，然后与建模 TGC 系统的 TBA 求积得到一个新的 TBA，利用判断该 TAB 所包含的语言是否为空来验证 liveness 是否成立。

根据定义 5，若所有的运行都不能进入到 TBA 的接受位置(Reject)，则不存在可接受的运行，则该 TBA 所接受的语言为空。这验证简单起见，可以先证明 Reject 不可达：

strong_liveness: THEOREM

∀(r:(lv. init_run?),i:nat):r(i)'loc'2≠Reject

其中，*lv* 是 TA *TGC* 与 TBA B_2 的积自动机的理论的名称。该引理的证明与上面的安全性证明方法类似[19]，这里不再赘述。应用引理 TBA 理论中的引理 accept_empty 可验证活性：

liveness:THEOREM lv. lang_empty。

6 相关工作

关于在 PVS 验证系统中建立时序逻辑的语法和语义形式规格说明的工作，S. Rajan 和 N. Shankar 等给出了模型检查和定量证明相结合的技术，给出了 CTL 在 PVS 中的 μ-演算理论和相关的引理，并在 PVS 证明器引入了(bddsimp)、(model-check)等命令支持有关公式的化简[15]。Paul. Gloess 中给出了 LTL 语言及某些性质的 PVS 规格说明[17]。T. Arons 和 A. Pnueli 给出了 LTL 在参数化公平系统 PFS(Parameterized Fair System)框架中的 LTL 验证系统 TLPVS[16]，这是一个涉及众多演绎规则的、功能强大的验证系统。本文的工作受到了如上工作的启发，但上面这些系统都是建立在离散序列之上的时序逻辑，而本文给出的时序逻辑系统是带有稠密时间(时间域是非负实数集合)扩充的，解释在 TA 形式模型上的时间化的时序逻辑，并且时间化的线性时序逻辑 TLTL 用 TBA 来表示，使其验证变得简单易行。

结论 本文在时间自动机和时间 Büchi 自动机的基础上，建立了解释在 TA 上的、带有稠密时间扩充的线性和分支时序逻辑的语法和语义模型，并在定理证明器 PVS 中实现解释在 TA 形式模型上的规格说明。本文还对一个建模为 TA 的经典的实时系统的特性进行了形式验证。下一步的工作应该考虑对时间化的时序逻辑进行形式验证的通用证明规则，以使实时特性的形式验证可以较为容易地进行。

参 考 文 献

1 Alur R, Dill D L. A theory of timed automata. Theoretical Computer Science, 1994, 126: 183～235
2 Clarke E, Grumberg O, Peled D. Model Checking. Cambridge: MIT PRESS, 1999
3 Qingguo XU, MIAO Huaikou. Modeling Timed Automata Theory in PVS:[上海大学计算机学院内部技术报告]
4 Owre S, Shankar N, Rushby J, et al. PVS system guide version 2.4. Computer Science Laboratory. SRI International, 2001. 1～97
5 Owre S, Shankar N, Rushby J, et al. PVS language reference version 2.4. Computer Science Laboratory. SRI International, 2001. 1～102
6 Alur R, Henzinger T A. Really temporal logic. Journal of the ACM, 1994, 41: 181～204
7 Shankar N, Owre S, Rushby J, et al. PVS prover guide version 2.4. Computer Science Laboratory. SRI International, 2001. 1～127
8 Alur R. Timed Automata. In: 11th International Conference on Computer-Aided Verification. LNCS 1633, Springer-Verlag, 1999. 8～22
9 Archer M. TAME: Using PVS Strategies for Special-Purpose Theorem Proving. Annals of Mathematics and Artificial Intelligence, 2000, 29(1-4): 139～181
10 Kolano P Z. Proof assistance for real-time systems using an interactive theorem prover. Theoretical Computer Science, 2002, 282(1): 53～99
11 Hooman J. Timed Automata in PVS. Summer School Zhengzhou, 2004
12 Owre S, Shankar N. Abstract Datatypes in PVS. Computer Science Laboratory, SRI International, 1997. 1～52
13 Owre S, Shankar N. The PVS Prelude Library. Computer Science Laboratory, SRI International, 2003. 1～31
14 Berard B, Bidoit M, Finkel A. System and Software verification Model-Checking Techniques and Tools. Springer-Verlag, 2001
15 Rajan S, Shankar N, Srivas M K. An integration of model-checking with automated proof checking. In: Wolper P, ed. Computer Aided Verification, CAV '95 volume 939 of Lecture Notes in Computer Science, Liege, Belgium, Springer-Verlag, June 1995. 84～97
16 Arons T, Pnueli A. TLPVS: A PVS-based LTL verification system, Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of his 64th Birthday, volume 2772 of Lecture Notes in Computer Science, Springer-Verlag, 2004. 598～625
17 http://www.enseirb.fr/~gloess/pvsdir/pvs/ temporal_logics/
18 Tripakis S. The analysis of timed systems in practice. Universite Joseph Fourier, Grenoble, France, December 1998
19 XU Qingguo, MIAO Huaikou. Formal Verification Framework for safety of Real-time System Based-on Timed Automata Model in PVS. accepted by the IASTED International Conference on Software Engineering (SE 2006), Innsbruck, Austria, 2006

(上接第 216 页)

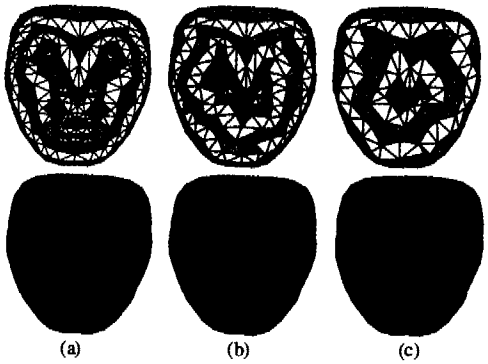

图 4 Nefertiti 模型的多分辨率表示
下为着色后的效果图

参 考 文 献

1 Hoppe H. Progressive meshes. In: Rushmeier H, ed. Proceedings of SIGGRAPH'96. New Orleans: ACM Press, 1996. 99～108
2 Pajarola R, Rossignac J. Compressed Progressive Meshes. IEEE Transactions on Visualization and Computer Graphics, 2000, 6(1): 79～93
3 Cohen-Or D, Levin D, Remez O. Progressive Compression of Arbitrary Triangular Meshes. In: Ebert D, Gross M, Hamman B, eds. Proceedings of IEEE Visualization'99, New Orleans: ACM Press, 1999. 67～72
4 Alliez P, Desbrun M. Progressive Encoding for Lossless Transmission of 3D Meshes. In: Fiume E, ed. Proceedings of SIGGRAPH'01. New Orleans: ACM Press, 2001. 198～205
5 Karni Z, Bogomjakov A, Gotsman C. Efficient Compression and Rendering of Multi-Resolution Meshes. In: Moorhead R, Gross M, Joy K I, eds. Proceedings of IEEE Visualization '02. New Zealand: IEEE Computer Society Press, 2002. 347～354
6 Taubin G, Gueziec A, Horn W, et al. Progressive Forest Split Compression. In: Cohen M, ed. Proc. Proceedings of SIGGRAPH'98. New Orleans: ACM Press, 1998. 123～132