基于层次化时间STM软件设计的形式化验证

周宽久[1,2] 任龙涛[1] 王小龙[1] 勇嘉伟[1] 侯 刚[1]
(大连理工大学软件学院 大连 116620)[1] (大连理工大学软件评测中心 大连 116620)[2]

摘 要 状态迁移矩阵(State Transition Matrix,STM)是一种基于表结构的程序建模语言。事件变量类型单一,事件和状态数量的增加很容易造成状态空间爆炸问题,无法表达具有时间语义的软件系统等原因,极大限制了该建模方法的推广应用。文中针对这些问题,首先提出层次化时间状态迁移矩阵(Hierarchical Time State Transition Matrix,HTSTM)模型,用于设计、建模和验证具有时间条件约束的软件系统,并给出形式化表示方法。基于该表示方法提出一种符号化编码方法,采用有界模型检测思想将需要验证的LTL性质输入SMT(Satisfiability Modulo Theories)求解器进行验证,从而在一定程度上证明了软件设计的正确性。

关键词 层次化时间状态迁移矩阵,形式化验证,有界模型检测
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.008

Modeling and Formal Verification for Software Designs Based on Hierarchical Timed State Transition Matrix

ZHOU Kuan-jiu[1,2] REN Long-tao[1] WANG Xiao-long[1] YONG Jia-wei[1] HOU Gang[1]
(School of Software,Dalian University of Technology,Dalian 116620,China)[1]
(Software Evaluation & Test Center,Dalian University of Technology,Dalian 116620,China)[2]

Abstract State Transition Matrix (STM) is a table-based modeling language. The popularization and application of this modeling method are greatly limited by the singleness of its event variable type, the state space explosion problem caused by the increasing events and states, and the weak expressive power in time semantics of software systems. We firstly presented a concept of Hierarchical Timed State Transition Matrix (HTSTM) which is mainly used for design, modeling and verification of software systems with time constraints. Then a formalization of HTSTM designs was provided as a state transition system. Based on the formalization, we proposed a symbolic encoding approach which adopts the idea of Bounded Model Checking (BMC). Finally the LTL properties to be verified were determined by Satisfiability Modulo Theories (SMT) so that the correctness of the software design was proved to a certain extent.

Keywords Hierarchical time state transition matrix, Formal verification, Bounded model checking

1 引言

状态迁移矩阵(State Transition Matrix,STM)[1,2]是一种基于表结构的软件系统设计建模语言,相比其他建模语言,具有结构规范化程度更高、模型修改更为简单、形式化过程更为简明的特点,但是其事件变量只能是布尔类型,内部表达形式单一,并且随着事件和状态的增多很容易造成状态空间爆炸的问题,单一表结构无法表达较为复杂的软件模型设计,这些问题极大地限制了STM的表达能力和应用范围。

当今软件设计通常会涉及到时间因素,尤其是具有高实时性的嵌入式软件设计,所以针对当前软件发展趋势和STM的不足,本文在状态变迁矩阵(STM)基础上引入时间事件,提出了层次化时间STM建模语言,并给出形式化表示方法;基于该表示方法提出一种符号化编码方法,采用有界模型检测思想[3,4]将需要验证的LTL性质输入SMT求解器进行验证[5],从而证明软件设计的正确性。

另外,目前基于STM模型的验证主要面临两个问题:1)状态空间爆炸问题,2)多STM并发控制问题。为了解决状态空间爆炸问题,孔维强[6,7]在状态变迁矩阵(STM)中引入状态性质即可满足性来减少测试用例,在一定程度上解决了状态爆炸问题,但可满足性验证方面(即判断采用谓词逻辑表达式为真)也容易产生爆炸问题。文献[8]采用时间抽象互模拟方法;文献[9,10]分别采用动态层次化UML状态机模型和符号模型进行验证;文献[11]将TA模型转换为有限状态迁移图,并将有限状态迁移图转换为非确定FSM,从而采用基于FSM的方法进行测试。为了降低状态空间爆炸给测试用例生成时带来的难度,文献[12]采用各种状态等价关系来对系统进行简化。本文根据层次化时间STM自身特点,在待验证性质中加入条件约束来纯化模型状态空间,从而在一定程度上解决了状态空间爆炸问题。

到稿日期:2013-07-01 返修日期:2013-07-25 本文受国家自然科学基金(61272174)资助。
周宽久(1966-),男,博士,教授,主要研究方向为软件可靠性、嵌入式系统建模,E-mail:zhoukj@dlut.edu.cn(通信作者);**任龙涛**(1988-),男,硕士生,主要研究方向为软件可靠性、形式化验证;**王小龙**(1989-),男,硕士生,主要研究方向为软件可靠性、嵌入式系统建模;**勇嘉伟**(1989-),男,硕士生,主要研究方向为软件可靠性、模型检测;**侯 刚**(1982-),男,博士生,讲师,主要研究方向为软件可靠性、复杂性理论。

2 HTSTM 形式化

2.1 HTSTM 行为语言

本文首先介绍定义 HTSTM 的行为语言 L，为了增强 HTSTM 建模的通用性，选择 C 语言语法和语义作为 L 语言的语法语义。L 的类型域包括布尔型、整数型、实数型。L 支持的表达式如下：(1)布尔型文字 true 和 false，整数和实数文字；(2)变量名；(3)时钟表达式；(4)中缀表达式 *leftexpr op rightexpr*，其中 *op* 包括 +，−，*，/，&&，‖，!=。

L 支持的语句包括：(1)赋值表达式 *lhs*=*rhs*；(2)子 STM 调用语句 □*child_STM_id*；(3)父 STM 返回语句 *returnXXX*。本文使用 L_{bool}，L_{time} 和 L_{stmt} 分别表示布尔表达式、时间表达式和其他赋值表达式的集合。

2.2 时间系统构建

大多数系统都需要测量时耗(Passage of Time)和执行特定动作，时间系统可以利用一阶微分方程建模[13]：

$$\forall t \in T_m, \dot{s}(t) = a \tag{1}$$

其中，$s: R \to R$ 是一个连续时间信号；$s(t)$ 是 t 时刻时钟；$T_m \subset R$ 是系统处于模式 m 时的时间子集；当系统处于该模式时，时钟速率 a 是一个常数。以 T 个时间单位为一个周期的时间系统如图 1 所示。

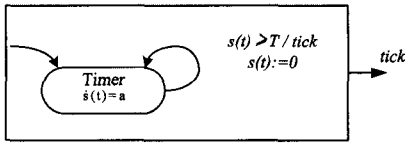

图 1 每隔 T 时间单位生成纯输出事件时间自动机

2.3 STM、HTSTM 形式化表示

定义 1 TSTM 是一个三元组 (S, E, C)，其中 S 是状态的有限集合，E 是事件的有限集合，C 是单元的有限集合；$VS(TSTM) = Var(TSTM) \cup \{ActiveStatus\}$，其中 $Var(STM)$ 表示 TSTM 中涉及的所有变量，*ActiveStatus* 表示当前活跃状态，TSTM 结构 C_{tstm} 可以表示 $Var(TSTM)$ 的一次赋值。

TSTM 中每个状态和事件都有唯一的自然数映射 $index(s) \in N$；每个 TSTM 至多有一个活跃状态，表示为 $active(TSTM)$，且初始的活跃状态 $active(TSTM) = index(s_0) = 0$。事件 E 包括 3 种情形事件：外部事件 E_{ext}(外部环境分配的事件，事件名以小写字母“x”开头)、内部事件 E_{int}(HTSTM 执行过程中内部分配的事件)、时间事件 E_{time}(时间约束事件)；其中 $E_{ext} \cap E_{int} = \emptyset$，$E_{ext} \cap E_{time} = \emptyset$，$E_{time} \cap E_{int} = \emptyset$；$E_{ext}$、$E_{int}$ 属于布尔类型，E_{time} 属于实数类型(通常为实数等式或者不等式表达式)，系统时间满足 E_{time} 时，该时间事件被触发。单元 C 包括 3 种类型：普通单元 C_{nor}，可忽略单元 C_{ign} 和错误单元 C_{err}。其中 $c_N \in C_{nor}$ 可以用五元组 $\langle s, e, u, a, s' \rangle \in S \times E \times L_{bool} \times L_{stmt} \times S$ 表示，定义 $source(c_N) = s$，$event(c_N) = e$，$guards(c_N) = u$，$actions(c_N) = a$，$target(c_N) = s'$；可忽略单元为 $\langle s, e, / \rangle$，表示该单元中不做任何改变；错误单元为 $\langle s, e, \times \rangle$，表示系统正常运行不可能到达该单元，如果到达则发生错误。

本文以交通灯_行人控制系统为例，建立层次化时间 STM 模型，如图 2 所示，该模型包括两部分：TRAFFIC_LIGHT_TSTM 和 PEDESTRIAN_TSTM，其中 TRAFFIC_LIGHT_TSTM 有子状态迁移矩阵 GREEN_LIGHT_TSTM。TRAFFIC_LIGHT_TSTM 和 PEDESTRIAN_TSTM 之间通过共享变量 *pedestrian*(有行人等待过马路)、*sigR*(红灯亮)、*sigG*(绿灯亮)进行通信。TRAFFIC_LIGHT_TSTM 中初始状态为 RED($source(0) == 0$)，当时间事件($event(c) = (x(t) \geqslant 60)$)成立时，执行普通单元 $c(0,0)$ 中 $action(c)$ 使得红灯熄灭，绿灯点亮，时钟归零，状态跳转到状态 GREEN，此时进入绿灯时段，车辆可通行；在绿灯状态下，调用子 *STM(GREEN_LIGHT_TSTM)*，如果有行人等待(*pedestrian*==true)，护卫条件 $x(t) < 60$，则返回状态标识 PENDING，使得父 TSTM 状态跳转到 PENDING；如果护卫条件 $x(t) \geqslant 60$，则绿灯熄灭，黄灯点亮，时钟归零，返回状态标 YELLOW，使得父 TSTM 状态跳转到 YELLOW；在黄灯时段($source(c) == 3$)，等待 5 个时间单位，黄灯熄灭，红灯点亮，时钟归零，状态跳转回到红灯状态。状态迁移矩阵 PEDESTRIAN_TSTM 表示行人状态转换系统，初始状态(NONE)表示路口没有行人等待过马路，当第一个行人到达路口时会给一个外部事件 *xWait*，此时普通单元 $c(0,0)$，$source(c) = 0$，$event(c) = xWait$，$action(c)$ 是 *pedestrian*=true，$target(c) = 1$；行人进入等待状态，当红色信号灯亮时(*sigR*==true)，进入 CROSSING 状态，行人可以过马路，当绿色信号灯亮时，行人停止过马路，系统返回到 NONE 状态。

TSTM0:TRAFFIC_LIGHT

□0		RED	GREEN	PENDING	YELLOW
		0	1	2	3

x(t) > 60	0	GREEN
		sigR=false;
		sigG=true;
			□1
			/
				YELLOW
				sigG=false;
				sigY=true;
				x(t)=0;
					/

x(t)==5	1	/	/	/	RED
					sigY=false;
					sigR=true;
					x(t)=0;

x(t)<60	2	/	□1 /	/	/

TSTM1:GREEN_LIGHT

□1		GREEN	
		0	

pedestrian	0	x(t)<60
		GREEN
		pedestrian=false;
		return PENDING;
			x(t) > 60
			GREEN
			sigG=false;
			sigY=true;
			x(t)=0;
			pedestrian=false;
			return YELLOW;

TSTM0:PEDESTRIAN

□0		NONE	WAITING	CROSSING
		0	1	2
xWait	0	WAITING pedestrian=true;	/	×
sigR	1	/	CROSSING /	/
sigG	2	/	/	NONE

TrafficLight PedestrianControlSystem: Structure

- TSTM0:TRAFFIC_LIGHT
  - TSTM1:GREEN_LIGHT
- TSTM0:PEDESTRIAN

图 2 HTSTM 运行实例：交通灯_行人控制系统

定义 2 HTSTM 使用元组形式可表示为 $(TSTM_0, TSTM_1, \cdots, TSTM_n, R)$，其中 $TSTM_0$ 是根 $TSTM$，R 元组中各 TSTM 之间的嵌套关系集合，如 $R(i,j) = \text{true}$，那么 $TSTM_i$ 是 $TSTM_j$ 的直接父 TSTM；HTSTM 状态用 $C_{htstm} = \{C_{tstm_0}, C_{tstm_1}, \cdots, C_{tstm_n}\}$ 表示，其中 C_{tstm_i} 可以表示为 $VS(TSTM_i) = Var(TSTM_i) \cup \{c\text{-}flag_i\}$ 的一次赋值，变量 $c\text{-}flag_i$ 用来控制每个 HTSTM 中至多有一个活跃 TSTM。

交通灯_行人控制系统包括两个 HTSTM：$HTSTM_0 = (TRAFFIC\_LIGHT, GREEN\_LIGHT)$，$HTSTM_1 = (PEDESTRIAN)$；$R(TRAFFIC\_LIGHT, GREEN\_LIGHT) =$

= true，表示 TRAFFIC_LIGHT 是 GREEN_LIGHT 的直接父 TSTM。

一个系统的设计 D 通常是由多个 $HTSTM_i=(TSTM_0, TSTM_1, \cdots, TSTM_n, R)$ 构成，可以表示为图 3。

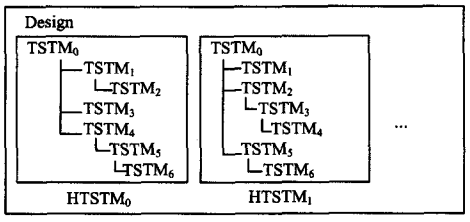

图 3　基于 HTSTM 系统构成示意图

系统设计可以形式化表示为 $D=(HTSTM_0, HTSTM_1, \cdots, HTSTM_n)$，其动态行为用 $M(D)$ 表示，$M(D)=(G, g_0, \Delta)$，其中 G 是所有全局声明的集合，$g\in G$，g_0 是初始全局声明，$\Delta\in G\times(U_{i=0}^{n}(U_{j=0}^{m}(HTSTM_i.TSTM_j)))\times G$ 表示所有 HTSTM 中任何一个 TSTM 状态迁移都会导致整个系统状态的迁移。

(1) 对于初始全局声明 g_0，每一个 $HTSTM_i$ 的活动 TSTM 均为根 TSTM，而每个 TSTM 的初始活跃状态 $active(TSTM)=index(s_0)=0$，每一个其他变量 $x\in Var(D)$ 都有默认的初始值。

(2) 动态迁移系统，迁移规则主要分为 3 类：普通单元导致的迁移、外部事件导致的迁移和时间事件导致的迁移。

普通单元导致的迁移规则表示普通单元的执行使得系统状态改变，其形式化表示为 $\langle g, c, g'\rangle\in\Delta$，设 c 是 $TSTM_i\in HTSTM_j$ 中的一个普通单元，$0\leqslant i\leqslant m$，$0\leqslant j\leqslant n$，rule1 表示该迁移规则，$g\xrightarrow{rule1}g'$ 表示在 rule1 作用下系统从状态 g 迁移到状态 g'，rule1 定义如下：

$$rule1.condition\triangleq v(c\text{-}flag_i^j)\wedge v(event(c))\wedge v(guards(c))\wedge v(ActiveStatus_i^j)=index(source(c)) \tag{2}$$

$$rule1.effects\triangleq g'=g[C'_{htstm_0}/C_{htstm_0},\cdots,C'_{htstm_n}/C_{htstm_n}] \tag{3}$$

其中，$condition$ 表示执行条件，$effects$ 表示执行效果，v 表示变量或者表达式的取值，C'_{htstm_i}/C_{htstm_i} 表示 $HTSTM_i$ 新状态 C'_{htstm_i} 取代旧状态 C_{htstm_i}。

外部事件导致的迁移规则 rule2 表示外部事件 E_{ext} 的发生导致系统状态改变 $g\xrightarrow{rule2}g'$，其定义规则如下：

$$rule2.condition\triangleq v(xE_{ext})=\text{false} \tag{4}$$

$$rule2.effects\triangleq g'=g[C'_{htstm_i}/C_{htstm_i}],\text{ where }C'_{htstm_i}=\{C_{tstm_0},\cdots,C_{tstm_j}[\text{true}/v(xE_{ext})],\cdots,C_{tstm_m}\} \tag{5}$$

时间事件导致的迁移规则 rule3 表示时间事件 E_{time} 的发生导致系统状态改变 $g\xrightarrow{rule3}g'$，其定义规则如下：

$$rule3.condition\triangleq v(E_{time}[x_{time}\rightarrow x])=\text{true} \tag{6}$$

$$rule3.effects\triangleq g'=g[C'_{htstm_i}/C_{htstm_i}],\text{ where }C'_{htstm_i}=\{C_{tstm_0},\cdots,C_{tstm_j}[v(E_{time})],\cdots,C_{tstm_m}\} \tag{7}$$

在系统执行的 t 阶段，系统迁移规则集合 $RS=\{r1_0^t, r2_0^t, \cdots, r1_k^t, r2_k^t\}$ 使能。为避免系统交叉执行，任意选择使能迁移规则 rule 执行迁移 $g\xrightarrow{rule}g'$。

3　HTSTM 符号编码方法

基于 HTSTM 模型的系统设计可表示为 $D=(HTSTM_0, HTSTM_1, \cdots, HTSTM_n)$，其中任意子项 $HTSTM_i=(TSTM_0, TSTM_1, \cdots, TSTM_n, R)$，$n, m$ 分别表示 HTSTM 和每个 HTSTM 中 TSTM 的数量，$0\leqslant i\leqslant m$，$0\leqslant j\leqslant n$，$VS(D)=\{x\mid x\in VS(HTSTM_j.TSTM_i^j)\}$ 表示系统中所有变量。

为了解决状态空间爆炸问题，本文符号化编码方法采用有界模型检测(BMC)的思想[3,4]，定义变量 bd 表示模型检测边界，即系统 D 执行序列最大长度为 bd，如果在 bd 范围内系统满足某项性质，则可以认为系统满足该性质。令 $0\leqslant k\leqslant bd$，在第 k 步的变量、表达式、语句分别使用新名称 $x[k]$、$expr[k]$、$stmt[k]$ 表示，用以区别不同步骤系统的执行情况。

3.1　HTSTM 初始符号编码表示

系统 D 在 step 0 初始状态表示为：

$$step[0]\triangleq\bigwedge_{x\in VS(D)}x[0]=v(x) \tag{8}$$

$$\{\text{true}/c\text{-}flag_0[0],\text{false}/c\text{-}flag_i[0]\mid 0<i\leqslant m\} \tag{9}$$

$$\{0/ActiveStatus_i^j\mid 0\leqslant i\leqslant m, 0\leqslant j\leqslant n\} \tag{10}$$

通过这个公式，将系统 D 中所有变量 x 重命名为 $x[0]$；$HTSTM_j$ 中根 TSTM 使能，每个 TSTM 中初始活跃状态为 $index(state)=0$。

3.2　迁移规则符号编码表示

针对 2.3 节中 3 条迁移规则：普通单元导致的迁移 rule1、外部事件导致的迁移 rule2 和时间事件导致的迁移 rule3，分别进行基于有界模型检测思想[11,12]的符号编码。

3.2.1　基于 rule1 符号编码方法

对于 $TSTM_i\in HTSTM_j$ 中普通单元 c，rule1 的条件编码定义如下：

$$rule1.condition[k]\triangleq c\text{-}flag_i^j[k-1]\wedge event(c)[k-1]\wedge guards(c)[k-1]\wedge ActiveStatus_i^j[k-1]=index(source(c)) \tag{11}$$

上面表达式 $c\text{-}flag_i^j[k-1]$，$event(c)[k-1]$，$guards(c)[k-1]$，$ActiveStatus_i^j[k-1]$ 的值是从 $step[k-1]$ 得到的。

rule1 效果编码定义如下：

$$rule1.effects[k]\triangleq actions(c)[k]\wedge(\bigwedge_{x\in VS(D)/V}x[k]=x[k-1]) \tag{12}$$

其中，V 表示 $actions(c)$ 中涉及到的已经被重新赋值的变量的集合。

3.2.2　基于 rule2 符号编码方法

$$rule2[k]\triangleq\neg xE_{ext}[k-1]\wedge xE_{ext}[k]\wedge(\bigwedge_{x\in VS(D)/\{xE_{ext}\}}x[k]=x[k-1]) \tag{13}$$

该公式将 $step[k-1]$ 步的外部事件 $xE_{ext}[k-1]$ 置为 false，将 $step[k]$ 步的外部事件 $xE_{ext}[k]$ 置为 true，其他变量保持不变。

3.2.3　基于 rule3 符号编码方法

$$rule3[k]\triangleq E_{time}[k][x_{time}\rightarrow x]\wedge(\bigwedge_{x\in VS(D)/\{E_{time}\}}x[k]=x[k-1]) \tag{14}$$

该公式表示 $step[k]$步时间事件 $E_{time}[k]$满足条件，其他变量保持不变。

3.3 BMC 公式符号编码方法

针对 rule1，定义 $RULE1=\{c|c\in TSTM_i.C_{ror}\}$表示系统 D 中所有普通单元；针对 rule2，定义 $RULE2=\{e|e\in TSTM_i.E_{ext}\}$表示系统 D 中所有外部事件；针对 rule3，定义 $RULE3=\{e|e\in TSTM_i.E_{time}\}$表示系统 D 中所有时间事件；令 $|RULE1|+|RULE2|+|RULE3|=w$，$r1\in RULE1$，$r2\in RULE2$，$r3\in RULE3$，定义布尔变量集合 $Flag=\{fg_1,\cdots,fg_w\}$，系统 D 在 $step[k]$的状态可以表示为：

$$step[k]\triangleq(\bigvee_{r1\in RULE1}(r1[k]\wedge fg(r1)[k]))\vee(\bigvee_{r2\in RULE2}(r2[k]\wedge fg(r2)[k]))\vee(\bigvee_{r3\in RULE3}(r3[k]\wedge fg(r3)[k])) \tag{15}$$

由于系统可能有多个规则条件同时成立，为了防止交叉执行和并行执行的情况，必须使系统执行要满足式(16)：

$$control\triangleq\bigwedge_{k=1}^{bd}(\bigwedge_{p=1}^{u}(fg_p[k]\Rightarrow\neg(\bigvee_{q=1,q\neq p}^{u}(fg_q[k])))) \tag{16}$$

系统需要验证的 LTL 性质为 ρ，采用可满足性模理论思想[14-17]，使用 SMT 求解器进行验证时需要将 LTL 性质取反，表示为 ρ_neg，综上可得式(17)：

$$BMC(D,\rho,bd)\triangleq(\bigwedge_{k=0}^{bd}step[k])\wedge control\wedge\rho\_neg \tag{17}$$

4 实验与评价

4.1 实验设计

以交通灯_行人控制系统为例，使用 LTL 性质形式化规范[8]对系统性质进行形式化表示[18,19]，本文主要对系统状态可达性、静态状态约束和动态状态约束进行形式化。

(1)验证系统状态可达性主要是验证系统中错误单元的不可达性，在系统 HTSTM 设计中符号╳表示该单元为错误单元，系统不可能到达该单元，使用 $active(H,g)$表示一个可达全局状态 g 中时间状态迁移矩阵 H 的活跃状态，则该性质可以表示为：

$$\forall c\in H_i.C_{invalid}.\ \forall g\in G.\ \neg((active(H_i,g)=source(c)\wedge event(c))) \tag{18}$$

本文交通灯_行人控制系统实例中，PEDESTRIAN 中单元(2,0)是一个错误单元，表示如果行人正处于可通行状态，就不能再请求通过，该错误单元可以表示为：

$$invalid=not((xWait=\text{true})\ \text{and}\ (statusPed=2)) \tag{19}$$

(2)系统静态约束包含两个 HTSTM 之间的一种静态关联情况，当 $HTSTM_i.TSTM_j$ 处于状态 $statusA$ 时，$HTSTM_e.TSTM_f$ 处于状态 $statusB$。这种静态约束关系可以形式化表示为：

$$\forall g\in G.active(H_A,g)=statusA\Rightarrow active(H_B,g)=statusB \tag{20}$$

针对本文中交通灯_行人控制系统实例，静态约束性质可以表示为：

$$static1=((statusTra=0)implies(statusPed=2)) \tag{21}$$

$$static2=((statusTra=2)implies(statusPed=1)) \tag{22}$$

$$static3=((statusTra=1)implies((statusPed=0)\ \text{or}\ (statusPed=1))) \tag{23}$$

静态性质 static1 表示当交通灯处于红灯亮的状态，行人系统一定处于通行状态；性质 static2 表示当交通灯处于挂起的状态，行人系统处于无人等待状态；static3 表示当交通灯处于绿灯亮的状态，行人系统处于无人等待状态或者行人等待状态。

(3)系统动态约束表示某个 TSTM 处于某项特定状态迁移时，另一个 TSTM 一定处于某个特定状态，这种性质可以形式化表示为：

$$\forall g\in G.active(H_A,g)=statusA\wedge active(H_A,g')=statusA'\Rightarrow active(H_B,g')=statusB \tag{24}$$

针对本文中交通灯_行人控制系统实例，动态约束性质可以表示为：

$$dynamic1=((statusTra=0)\ \text{and}\ (statusTra=1)\ implies(statusPad=0)) \tag{25}$$

$$dynamic2=((statusTra=1)\ \text{and}\ (statusTra=3)\ implies\ (statusPad=1)) \tag{26}$$

dynamic1 表示如果交通灯系统由红灯变为绿灯，那么行人系统一定会处于无人通行状态；dynamic2 表示如果交通灯系统由绿色变为黄色，那么行人系统一定处于有人等待状态。

4.2 实验环境

实验平台配置是 CPU：AMD Athlon(tm) Ⅱ X4 610e Processor 2.4GHz，内存：4.00GB，硬盘：1T，Windows 7 操作系统(32 位)，以图 2 交通灯_行人控制系统为原型系统，使用上面介绍的方法进行形式化和符号化编码，然后使用 Z3 SMT 求解器对系统相关性质进行求解[20]，从而证明本文所提方法的有效性。

4.3 性质验证

使用 SMT 求解器 Z3 对上述 6 条性质进行验证，可得表 1 所列结果(BMC 边界值 $bd=50$)。

表 1 Z3 求解器验证原始系统性质结果

Property	Step	Verdict	Time(s)
invalid	50	unsat	2.79
static1	50	unsat	2.94
static2	24	sat	1.47
static3	50	unsat	3.14
dynamic1	18	sat	0.95
dynamic2	32	sat	1.81

针对图 2 所示交通灯_行人控制系统 HTSTM 模型，对上述 6 条性质进行验证，结果如表 1 所列，其中 Step 是验证性质找到反例或者找不到反例所执行的长度，如果找到反例则停止验证；Verdict 是模型验证结果，如果是 *unsat* 则表示该性质可满足，如果是 *sat* 则表示该性质不可满足；Time 是验证某条性质所用的时间(单位是秒)。以 static2 为例分析性质不可满足的原因，当普通单元(0,0)使能，TRAFFIC_LIGHT 活跃状态会跳转到 GREEN，但是在跳转之前系统时间没有初始化为 0，所以当调用子 TSTM GREEN_LIGHT 时会导致 $x(t)<60$ 的护卫条件永远不可能成立，该条件下的 $action(c)$也不会被执行，所以性质 static2 会发生错误。需要注意的是，通过 Z3 输出的所有变量值来理解反例是非常困难的，需要仔细分析实验数据。

通过在 TRAFFIC_LIGHT 的(0,0)中添加 $x(t)=0$ 来修正原始模型中的错误，再次进行模型验证，可以得到如表 2 所列实验结果，除了 dynamic1 本身是一条错误性质之外，其他性质都是可满足的。

表 2 Z3 求解器验证修正后系统性质结果

Property	Step	Verdict	Time(s)
invalid	50	unsat	2.86
static1	50	unsat	3.35
static2	50	unsat	3.62
static3	50	unsat	4.18
dynamic1	16	sat	0.87
dynamic2	50	unsat	4.79

结束语 本文针对 STM 的不足提出 TSTM 的概念，用以表示包含时间因素的软件系统的设计、建模和验证，将时间因素形式化为模型的时间变量来驱动模型的运转。针对当前软件规模和复杂性的不断提高使得建模过程极易出现状态空间爆炸的问题，本文提出 HTSTM 的概念，将 TSTM 进行层次化建模，从而在一定程度上解决该问题。基于 HTSTM 模型本文给出形式化表示方法，并在此基础上提出一种符号化编码方法，采用有界模型检测的方法，使用 SMT 求解器对模型性质进行验证。从实验结果可以看出，本文提出的方法可以较好地整合 SMT 求解器[14]等现有资源，在可以接受的时间范围内验证模型性质，找出模型中的设计错误或者逻辑错误。

但本文方法仍然有很多改进的空间。我们下一步的工作重点是使用启发式思想或者其他方法对需要验证的性质进行优化，或者对形式化过程进行优化，从而降低模型验证需要的时间和空间方面的开销。

参 考 文 献

[1] Matsumoto M, Anada K, Ueshima D, et al. Model Checking of State Transition Matrix[R]. ITSSV 2005, AIST Technical Report. 2005:2-11

[2] Watanabe M. Extended Hierarchy State Transition Matrix Design Method -Version 2.0[R]. CATS Technical Report. 1998

[3] Clarke E, Kroening D, Ouaknine J, et al. Completeness and complexity of bounded model checking[C]// Proceedings of the Verification, Model Checking, and Abstract Interpretation. Springer Berlin Heidelberg, 2004:85-96

[4] Clarke E M, Grumberg O, Peled D. Model checking[M]. The MIT press, 1999

[5] Armando A, Mantovani J, Platania L. Bounded model checking of software using SMT solvers instead of SAT solvers[M]// Model Checking Software. Springer Berlin Heidelberg, 2006: 146-162

[6] Kong Wei-qiang, Fukuda A, Watanabe M. An SMT approach to bounded model checking of design in state transition matrix[C]// Proceedings of the Computational Science and Its Applications (ICCSA). 2010:231-238

[7] Kong Wei-qiang, Katahira N, Watanabe, et al. Formal verification of software designs in hierarchical state transition matrix with SMT-based bounded model checking[C]// Proceedings of the Software Engineering Conference (APSEC). 2011:81-88

[8] Latvala T, Biere A, Heljanko K, et al. Simple bounded LTL model checking[C]// Proceedings of the Formal Methods in Computer-Aided Design-2004. 2004:186-200

[9] Jussila T, Dubrovin J, Junttila T, et al. Model checking dynamic and hierarchical UML state machines[C]// Proc. MoDeV2a: Model Development, Validation and Verification. 2006:94-110

[10] Dubrovin J, Junttila T. Symbolic model checking of hierarchical UML state machines[C]// Proceedings of the Application of Concurrency to System Design. 2008:108-117

[11] Biere A, Cimatti A, Clarke EM, et al. Symbolic Model Checking without BDDs[C]// TACAS 1999. 1999:193-207

[12] Latvala T, Biere A, Heljanko K, et al. Simple is better: Efficient bounded model checking for past LTL[C]// Proceedings of the Verification, Model Checking, and Abstract Interpretation. 2005:380-395

[13] Lee E A, Seshia S A. Introduction to embedded systems: A cyber-physical systems approach[M]. Lee & Seshia, 2011

[14] Barrett C, Sebastiani R, Seshia S A, et al. Satisfiability modulo theories[J]. Handbook of Satisfiability, 2009, 185:825-885

[15] Veanes M, Bjørner N, Alexander R. An SMT approach to bounded reachability analysis of model programs[C]// Proceedings of the Formal Techniques for Networked and Distributed Systems-FORTE 2008. Springer Berlin Heidelberg, 2008:53-68

[16] 曾锃，赵建华. 基于程序分析的代码查询技术[J]. 计算机科学，2012，39(2):143-147

[17] 何炎祥，吴伟，陈勇，等. 基于 SMT 求解器的路径敏感程序验证[J]. 软件学报，2012，23(10)

[18] Dubrovin J, Junttila T, Heljanko K. Symbolic step encodings for object based communicating state machines[M]. Springer Berlin Heidelberg, 2008:96-112

[19] Cousot P. Abstract interpretation based formal methods and future challenges[C]// Proceedings of the Informatics. 2001:138-156

[20] De Moura L, Bjørner N. Z3: An efficient SMT solver[M]// Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008:337-340

(上接第 41 页)

[2] Scott M. Dynamic frequency and voltage scaling for a multiple-clockdomain microprocessor[J]. IEEE Micro, 2003, 23(6):62-68

[3] Nathuji R, Schwan K. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems[C]// SOSP'07. Stevenson, Washington, USA, October 2007

[4] Verma A, Ahuja P, Neogi A. Power-aware Dynamic Placement of HPC Applications[C]// Proceedings of the 2008 ACM International Conference on Supercomputing(ICS'08). 2008:175-184

[5] Fallenbeck N, Picht H, Smith M, et al. Xen and the art of cluster scheduling[C]// First International Workshop on Virtualization Technology in Distributed Computing. 2006:4-4

[6] Kim K H, Buyya R, Kim J. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters[C]// CCGRID. 2007:541-548

[7] Ge R, Feng X, Cameron K. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters[C]// Proceedings of the 2005 ACM/IEEE conference on Supercomputing. IEEE Computer Society Washington, DC, USA, 2005