软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(2):223−238 [doi: 10.13328/j.cnki.jos.004777]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于时间 STM 的软件形式化建模与验证方法*

侯 刚, 周宽久, 常军旺, 王 洁, 李明楚

(大连理工大学 软件学院,辽宁 大连 116623)

通信作者: 王洁, E-mail: wangjie1003@163.com

摘 要: 状态迁移矩阵(state transition matrix,简称 STM)是一种基于表结构的状态机建模方法,前端为表格形式,后端则具有严格的形式化定义,用于建模软件系统行为.但目前 STM 不具有时间语义,这极大地限制了该方法在实时嵌入式软件建模方面的应用.针对这一问题,提出了一种基于时间 STM(time STM,简称 TSTM)的形式化建模方法,通过为 STM 各单元格增加时间语义和约束,使其适用于实时软件行为刻画.此外,针对 TSTM 给出了一种基于界限模型检测(bounded model checking,简称 BMC)技术的时间计算树逻辑(time computation tree logic,简称 TCTL)模型检测方法,以验证 TSTM 时间及逻辑属性.最后,通过对某型号列控制软件进行 TSTM 建模与验证,证明了上述方法的有效性.

关键词: 时间 STM;界限模型检测;时间计算树逻辑;实时嵌入式软件

中图法分类号: TP311

中文引用格式: 侯刚,周宽久,常军旺,王洁,李明楚.基于时间 STM 的软件形式化建模与验证方法.软件学报,2015,26(2):223−238. http://www.jos.org.cn/1000-9825/4777.htm

英文引用格式: Hou G, Zhou KJ, Chang JW, Wang J, Li MC. Software formal modeling and verification method based on time STM. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):223−238 (in Chinese). http://www.jos.org.cn/1000-9825/4777.htm

Software Formal Modeling and Verification Method Based on Time STM

HOU Gang, ZHOU Kuan-Jiu, CHANG Jun-Wang, WANG Jie, LI Ming-Chu

(School of Software Technology, Dalian University of Technology, Dalian 116623, China)

Abstract: State transition matrix (STM), designed for modeling software system, is a table-based modeling language in which the front-end is expressed in the table form and the back-end has strict formalized definition. At present, however, STM has no time semantics, which greatly limits the application of this method in real-time embedded software modeling. In order to solve this problem, this paper proposes a time STM (TSTM) modeling method attained by adding time semantics and constraint for each cell in STM, making it suitable for describing real-time system behavior. In addition, a time computation tree logic (TCTL) model checking method is presented based on bounded model checking (BMC) technology for verification of time and logic properties of TSTM model. At last, the effectiveness of the proposed method is validated by modeling and verifying certain type train control software.

Key words: time STM; bounded model checking; time computation tree logic; real-time embedded software

随着软件本身及其运行环境的日益复杂,软件可信性问题引起人们越来越多的关注[1].如何保证可信性涉及到软件开发过程众多方面,包括图灵奖得主 A.Pnueli 在内的许多计算机科学家都认为,采用形式化方法(formal method)对软件进行模型设计与验证是其中的重要手段[2].模型检测(mode checking)[3,4]是一种验证有限状态系统满足规范的形式化方法,由于检测过程在算法支持下可以自动执行,因此被越来越多的研究者或工程人员用于对软件安全性、活性、公平性等功能属性进行可信验证.

软件模型检测主要包括 3 个方面的内容:一是软件建模,通过构造软件计算模型来刻画软件不同的行为特

* 基金项目: 国家自然科学基金(61402073, 61272174)
收稿时间: 2014-07-01; 修改时间: 2014-10-31; 定稿时间: 2014-11-26

征;二是属性描述,通过时序逻辑语言(例如:计算树逻辑 CTL[5],线性时态逻辑 LTL[6],时间计算树逻辑 TCTL[7],概率计算树逻辑 PCTL[8],连续语义线性时态逻辑 LTLC[9],连续随机逻辑 CSL[10]等)定义软件必须满足的一些性质;三是检测算法,根据所使用模型的特点和时序逻辑规约设计对应的模型检测算法,这方面面临的最大问题就是状态空间爆炸,相关学者也提出了大量的解决方法,如 OBDD 符号化技术[11]、组合验证[12]、界限模型检测[13−15]等.

在软件建模方面,可采用形式化模型和半形式化模型.形式化模型主要是自动机、Petri 网以及它们的相关扩展模型[16−18].形式化模型的优势在于本身具有严格的数学规范,可直接用于模型检测;缺点在于模型难以理解,不利于软件开发人员进行后续软件开发.半形式化模型主要为 UML 及其扩展模型[19,20],其优势在于可直接用于软件开发;其缺点是不能直接进行模型检测,需要进一步转化为形式化模型.在这个过程中,也缺少自动化工具支撑,所以实用性较差.

针对上述问题,日本 CASE 株式会社提出了一种基于 STM 的软件建模方法,并已开发出商用工具 ZIPC[21],目前已在嵌入式软件开发领域得到了广泛应用.在 STM 中(如图 1 所示),列表示软件中存在的状态;行表示软件中将要发生的事件;行列交叉单元表示在某一状态下,当某一事件发生时,软件需要进行的处理.处理包括两部分内容:一是正常的事务处理,二是软件状态迁移.通过 STM 结构,既保留了软件模型的形式化语义,又便于后续软件开发(软件需求、设计均可通过 STM 完成,并且可以自动生成软件代码框架).STM 支持层次化设计,可将大型软件分解为若干子系统,各子系统由若干 STM 表格组成,各表格之间可通过共享变量、消息传递或者表格调用等机制进行通信.此外,STM 还可以显式地表示出软件设计遗漏(使用图 1 中的“错误(×)”单元和“忽略(/)”单元,例如:在电源 ON 状态下,发生了 ON 事件该如何处理;在电源 OFF 状态下,发生了 OFF 事件该如何处理),这些遗漏对于软件安全性影响较大,而通过传统的软件建模方法难以被有效地发现.

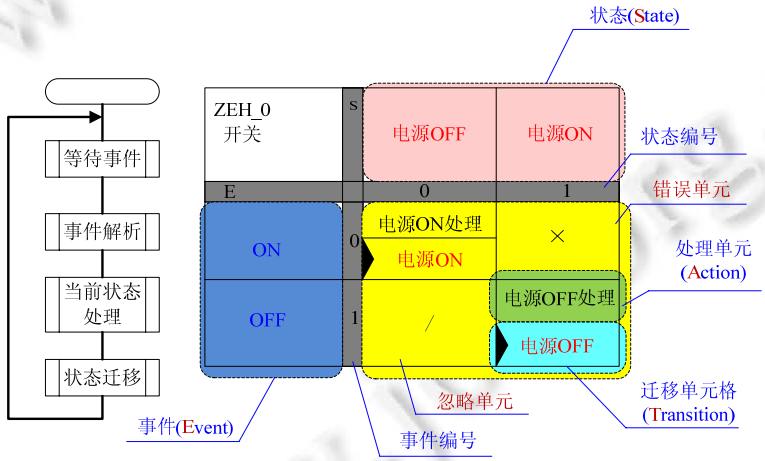


Fig.1 Structure of STM

图 1 STM 结构

目前,针对软件 STM 模型的正确性验证包括两种方法:一种是通过动态仿真运行,实时观测 STM 的事务处理与状态迁移是否满足软件设计要求,另一种是使用模型检测手段,将 STM 模型与待验证的软件属性编码为一阶谓词逻辑公式,通过模型验证工具验证软件设计是否满足属性要求[22,23].

作为一种实用性较强的软件建模方法,STM 也存在一定的问题.主要体现在建模手段上,无法有效地表示事务处理时间,进一步导致了系统状态迁移时间不能被有效分析.而嵌入式软件对实时性要求严格,因此,通过 STM 无法分析嵌入式软件时间特性.针对这一问题,本文提出了一种 TSTM 的形式化建模方法,通过为各单元操作引入时钟映射函数,使 TSTM 具有显示表示时间的能力.此外,针对 TSTM,给出了一种基于 BMC 策略的 TCTL 模型检测方法,用以验证 TSTM 时间及逻辑属性.最后,通过 TSTM,对某型号列控制软件进行建模与验证,证明上

述方法的有效性.

1 TSTM 与 TCTL 相关定义

1.1 TSTM形式化定义

在给出 TSTM 形式化定义之前,首先定义 TSTM 建模语言 *L*.*L* 语言是 ANSI-C 语言的一个子集,它遵从 C 语言的语法和语义.*L* 语言数据类型域由布尔型(Boolean)、整数型(integer)和实数型(real)组成,支持的表达式包括:(1) 布尔文字{true,false}、整数文字和实数文字;(2) 变量标识符;(3) 中缀表达式:{*expression* 1}操作符{*expression* 2},其中,操作符包括+,−,*,/,&&,||,!=,=.*L* 支持的语句包括:(1) 赋值语句 *stl*=*str*;(2) 条件判断语句 if {*condition*}{*statement* 1} else {*statement* 2};(3) 分支判断语句 *switch* {*condition*}, *case* 1 {*statement* 1}; … *case n* {*statement n*}.

定义 1(TSTM). 一个 TSTM 表格 H 可以用五元组$\langle S,E,C,T,\Phi\rangle$表示,其中,

- S 是有限状态集合.每个状态 $s\in S$ 都有唯一的自然数索引值 $index(s)\in\mathbf{N}$ 与之相对应.在 H 运行过程中,只有唯一的激活状态,用 $active(H)$表示.初始时,默认激活状态 s 的索引值 $index(s)=0$(即最左侧状态).
- E 是有限事件集合.包括外部事件 $E_{external}$ 和内部事件 $E_{internal}$,且 $E_{external}\cap E_{internal}=\varnothing$.每个外部事件 $e_X\in E_{external}$ 用 L 的一个布尔变量表示;每个内部事件 $e_I\in E_{internal}$ 用 L 的一个布尔表达式或布尔变量表示.每个状态 $e\in E$ 都有唯一的自然数索引值 $index(e)\in\mathbf{N}$ 与之相对应.
- C 是有限单元集合,包括正常单元 C_{normal}、忽略单元 C_{ignore} 和错误单元 C_{error}(也称为不可用单元):
  (1) 正常单元 $c_N\in C_{normal}$ 是一个六元组$\langle s,e,u,a,s',\varphi\rangle\in S\times E\times(L\cup I(X)\cup\{null\})\times(L\cup\{/\})\times S\times\Phi$,其中,$source(c_N)=s$ 为 c_N 所对应的状态,$event(c_N)=e$ 为触发 c_N 的事件,$guards(c_N)=u$ 为守卫条件(包括逻辑条件与时间约束),$actions(c_N)=a$ 为执行动作,$target(c_N)=s'$ 为迁移目标状态,$\Phi(c_N)=\varphi$为执行时间赋值,$I(X)$为时钟约束集,文字 *null* 表示无守卫条件.单元 c_N 规定了当 H 处在状态 $source(c_N)$ 且有事件 $event(c_N)$到来,当满足 $guards(c_N)$中的逻辑条件和时间约束时,H的动作行为 $actions(c_N)$ 以及对应的执行时间消耗 $\Phi(c_N)$.
  (2) 忽略单元 $c_I\in C_{ignore}$ 是一个三元组$\langle s,e,/\rangle$,其中,s,e 定义与 c_N 相同,符号"/"表示什么都不做.
  (3) 错误单元 $c_E\in C_{error}$ 是一个三元组$\langle s,e,\times\rangle$,其中,$s$,$e$ 定义与 c_N 相同,符号"×"表示发生错误.
- T 是有穷时钟集合,X 为时钟变量集合,时钟约束δ的集合 $I(X)=\{\delta|\delta::=x\leqslant\alpha|\alpha\leqslant x|\neg\delta|\delta_1\wedge\delta_2\}$,其中,$x$ 是 X 中的一个时钟,α是非负有理数集 $\mathbf{Q}$ 中一个常量.
- Φ是时钟映射函数,它为 TSTM 中每个正常单元 $c_N\in C_{normal}$ 指定 $\Phi(c_N)\in T$ 中某个时钟赋值.时钟集合 T 上的时钟赋值是指为每个时钟分配一个实数值,也可以说它是从集合 T 到非负实数集 $\mathbf{R}$ 的一个映射.T 上一个时钟赋值满足 X 上一个时钟约束δ,当且仅当依照时钟赋值给出的时钟值,时钟约束δ的布尔值为真.

1.2 TCTL语法与语义

TCTL 是在 CTL 基础上,通过在时态逻辑算子上增加时间约束实现对实时系统时间属性的刻画.下面给出 TCTL 的语法和语义.

定义 2(s-路径). 对于状态集合 S 和 $s\in S$,通过 S 的 s-路径是从 R 到 S 满足$\rho(0)=s$ 的映射ρ.

定义 3(TCTL 结构). TCTL 结构是三元组 $M=\langle s,\mu,f\rangle$,其中,

- S 是状态集合;
- $\mu:s\to 2^{AP}$ 是一个标记函数,AP 为原子命题集合,μ赋给每一个状态原子命题集合为真;
- f 是给定每一个 $s\in S$ 一通过 S 的 s-路径集合的映射.

定义 4(TCTL 公式). $\alpha,\beta::=\text{true}|p|\neg\alpha|\alpha\wedge\beta|\alpha\vee\beta|EF_{\varphi}\alpha|E\alpha U_{\varphi}\beta|A\alpha U_{\varphi}\beta$,其中,$p$ 是原子公式,$\varphi\in\mathbf{R}^+$是时间约束,包含以下形式:$[n,m]$,$[n,m)$,$(n,m]$,(n,m),$[n,\infty]$和(n,∞),$n,m\in\mathbf{N}$.

定义 5(TCTL 语义). 对于TCTL结构$M=\langle s,\mu,f\rangle$,令$s_i\in S,p\in AP,\pi\in f(s_i)$是以$s_i$为起始状态的$s$-路径,$\psi$为TCTL公式,则满足关系$(M,s_i)\models\psi$可定义如下:

- $s_i\models p$ 当且仅当 $p\in\mu(s_i)$;
- $s_i\models\neg p$ 当且仅当 $p\notin\mu(s_i)$;
- $s_i\models\alpha\wedge\beta$当且仅当$(s_i\models\alpha)\wedge(s_i\models\beta)$;
- $s_i\models\alpha\vee\beta$当且仅当$(s_i\models\alpha)\vee(s_i\models\beta)$;
- $s_i\models EF_{\varphi}\alpha$当且仅当$\exists\pi.(s_0=s_i\wedge\exists k\geqslant 0(s_k\models I_{\varphi}^{k}\wedge a))$;
- $s_i\models E\alpha U_{\varphi}\beta$当且仅当$\exists\pi.(w_0=s\wedge 0\leqslant\exists i\leqslant k((M_k,w_i\models_k I_{\varphi}^{i}\wedge\beta)\wedge 0\leqslant\forall i\leqslant j(M_k,w_j\models_k\alpha))$;
- $s_i\models A\alpha U_{\varphi}\beta$当且仅当$\forall\pi.(w_0=s\wedge 0\leqslant\exists i\leqslant k((M_k,w_i\models_k I_{\varphi}^{i}\wedge\beta)\wedge 0\leqslant\forall i\leqslant j(M_k,w_j\models_k\alpha))$.

这里,用I_{φ}表示时间约束φ被满足,用I_{φ}^{i}表示状态i满足时间约束φ.

1.3 TSTM动态行为及全局状态空间构造

一个软件设计D通常由多个TSTM $H_1,H_2,\ldots,H_n$组合而成,即$D=(H_1,H_2,\ldots,H_n)$.软件运行时,这些TSTM内部状态将进行迁移,TSTM间全局共享变量将发生变化,这些状态和全局变量变化即为软件D的状态迁移.

定义 6(TSTM 动态行为). 用TSTM建模的软件设计D,其动态行为可以定义为$B(D)=\langle G,g_{init},\Delta\rangle$,其中,

- G为全局状态集合,包含了各TSTM的激活状态$active(H_1),active(H_2),\ldots,active(H_n)$以及$D$中所有变量$X(D)$的当前取值$value(X(D))$.即,$G=\langle active(H_1),active(H_2),\ldots,active(H_n),value(X(D))\rangle$.
- g_{init}为初始状态.在初始状态,TSTM H_i的激活状态s_i满足$index(s_i)=0$,变量$x_i\in X(D)$为初始默认值.
- $\Delta\subseteq G\times(\bigcup_{i=1}^{n}H_i.C_{normal})\times G$为状态迁移,即,当TSTM H_i的$c_N\in C_{normal}$类型单元格运行时(使能),全局状态G将发生迁移.用$enabled(c_N,g)$表示c_N在全局状态g下使能,当且仅当以下条件都为真:
  (1) 单元c_N的对应状态是H_i的激活状态$active(H_i)=source(c_N)$;
  (2) 在全局状态g下,触发c_N的事件发生$event(c_N,g)$=true;
  (3) 在全局状态g下,c_N的守卫条件满足$guards(c_N,g)$=true.

当单元c_N使能时,$action(c_N)$表示的操作语句将自动执行,使软件设计D从全局状态g迁移到另一个全局状态g',用$\langle g,c_N,g'\rangle\in\Delta$表示,并且在下一个状态$g'$中,$H_i$的激活状态$active(H_i)=target(c_N)$.

给定一个软件设计D及其动态行为$B(D)$,则一条执行序列sq可表示为$g_0,g_1,\ldots,g_n$,其中,$g_0=g_{init}$,对任意$i\in\mathbf{N}$,$\langle g_i,c_N,g_{i+1}\rangle\in\Delta$.全局可达状态为包含在与动态行为$B(D)$相关任何执行序列中的状态,所有全局可达状态集合用$R_{B(D)}$表示.因此,要描述一个软件属性$f:G\to Boolean$在$B(D)$的所有全局可达状态得到满足,可表示为

$$\forall g:R_{B(D)}:f(g).$$

2 基于 BMC 技术的 TSTM 验证方法

BMC技术是针对OBDD符号模型检测不足而产生的一种新的模型检测技术[24−26],其基本思想是在整数界限为k长度之内的执行路径中,寻找不满足性质的反例(counter-example).BMC的优势就是把BMC问题编码成SAT[27]/SMT[28]实例,并充分利用已有验证工具进行求解,使可验证的系统规模和变量数得到较大的提升.目前的SAT/SMT求解器[27,29]可以处理具有几千个变量的公式,保证了BMC方法的有效性.此外,由于BMC采用广度优先搜索策略,因此所获得的反例长度最短、最简明,有利于理解和找出违反性质的原因.

2.1 针对TSTM的BMC方法

基于BMC技术的TSTM模型验证方法如下:

(1) 将待验证软件建模为TSTM模型,并构造其全局可达状态集合构成的有限自动机模型M.
(2) 设验证边界上界k,用TCTL公式表示待验证软件属性的否定形式(negative normal form,简称NNF)f.
(3) 将模型M和待验证属性的NNF形式进行合取,构成k步之内的BMC公式:

$$[[M,f]]_k=[[M^{f,s_0}]]_k\wedge[[f]]_{M_k},$$

其中,$[[M^{f,s_0}]]_k$ 表示第 k 步时 M 的执行序列(s_0 为起点),$[[f]]_{M_k}$ 表示第 k 步时的属性表达式.

(4) 将 BMC 公式求解问题规约为逻辑公式的可满足性判定,即,转化为 SAT/SMT 实例,然后通过求解工具自动判定.若判定结果可满足,则找到反例;若不可满足,则表明待验证属性在软件运行到 k 阶段以内是成立的.

2.2 符号编码方法

2.2.1 TSTM 符号编码方法

文献[22,23]针对 STM 模型给出了符号化编码方案,在此基础上,由于 TSTM 为 $c_N\in C_{normal}$ 增加了执行时间赋值 $\Phi(c_N)$,因此可以计算状态迁移过程中累计时间消耗.分析此类问题时,软件 TSTM 模型中需增加时间变量.此外,对于时间敏感的状态迁移,也需在 $guards(c_N)$ 判定中增加对应的时间约束判定.在采用 BMC 技术对 TSTM 进行符号化编码时,设模型检测边界为 bd,令 $0\leqslant k\leqslant bd$,在第 k 步的变量和时间累积分别使用 $x[k]$ 和 $time[k]$ 表示.

$B(D)$的初始状态用 $step[0]$表示,公式如下:

$$step[0]=(value(X(D)[0])\wedge\bigwedge_{i=1}^{n}(active(H_i)[0]=0)\wedge(time[0]).$$

其中,$value(X(D)[0])$表示所有全局变量的初始值,$active(H_i)[0]$表示 TSTM H_i 的初始状态,$time[0]$表示系统的初始时间.

(1) 单元格 c_N 的符号编码方法

在 TSTM 中,模型状态迁移是由于 $c_N\in H_i.C_{normal}$ 的执行,即,正常单元格被触发.为了得到 $B(D)$在第 k 步的状态公式 $step[k]$,需要给出第 k 步时 c_N 的使能条件:

$$enabled(c_N)[k]=guards(c_N)[k-1]\wedge event(c_N)[k-1]\wedge active(H_i)[k-1]=source(c_N).$$

其中,$guards(c_N)[k-1]$表示在第 $k-1$ 步时 c_N 守卫条件,$event(c_N)[k-1]$表示在第 $k-1$ 步时能够触发单元格 c_N 执行的事件,$active(H_i)[k-1]$表示在第 $k-1$ 步时 TSTM H_i 的激活状态,$source(c_N)$表示单元格 c_N 所对应的状态.

当 c_N 满足运行条件(被使能后)时,就需运行 c_N 中相关操作,则在第 k 步 c_N 执行的操作可编码为

$$effects(c_N)[k]=actions(c_N)[k]\wedge active(H_i)[k]=target(c_N)\wedge time[k]=time[k-1]+\Phi(c_N).$$

$action(c_N)[k]$表示 c_N 中相关操作(具体编码方案可见文献[22]),即,事务处理或变量赋值;$active(H_i)[k]$表示在第 k 步时 TSTM H_i 的激活状态,这个状态为 c_N 对应的迁移目标状态 $target(c_N)$;$time[k]=time[k-1]+\Phi(c_N)$表示更新模型运行时间,因为 c_N 运行需要消耗时间,通过这一操作实现动态行为 $B(D)$的时间累计.

最后,单元格 c_N 可符号编码为

$$c_N[k]=(enabled(c_N)[k]\wedge effects(c_N)[k])\vee(\neg enabled(c_N)[k]\wedge\bigwedge_{x\in X(D)}x[k]=x[k-1]).$$

$x[k]=x[k-1]$表示由于 c_N 的使能条件没有满足,因此第 k 步的变量取值与第 $k-1$ 步相同.

(2) 事件 e 的符号编码方法

在 TSTM 中,模型状态迁移需要通过事件进行触发,事件又分为内部事件和外部事件:内部事件由 c_N 的运行而触发,外部事件需要由与模型交互的环境触发.下面给出当外部事件 e 触发时模型需要进行的处理:

$$ext(e)[k]=(\neg e[k-1]\wedge e[k]=\text{true}\wedge\bigwedge_{x\in(X(D)\setminus\{e\})}x[k]=x[k-1])\vee(e[k-1]\wedge\bigwedge_{x\in X(D)}x[k]=x[k-1]).$$

其中,$e[k]$表示第 k 步时的事件状态,为布尔型变量.

(3) 第 k 步 $step[k]$符号编码方法

通过 $c_N[k]$和 $ext(e)[k]$的编码方法,可以得到软件 TSTM 模型动态行为 $B(D)$在第 k 步时的符号编码方法:

$$step[k]=\bigvee_{i=1}^{n}\left(\left(\bigvee_{c\in H_i.C_{normal}}c_N[k]\right)\vee\left(\bigvee_{e\in H_i.external}ext(e)[k]\right)\right).$$

2.2.2 TCTL 属性编码方法

针对 TCTL 的界限模型检测,相关学者已给出了 BMC 编码及优化方案.本文采用文献[13]的符号编码方法,

下面给出 TCTL 的边界语义.

定义 7(TCTL 边界语义). 令 M_k 为模型 M 的第 k 步时的转换公式,$s_i \in S, p \in AP, \alpha, \beta$ 为 TCTL 公式的否定形式(NNF),$M_k, s \models_k \alpha$ 表示在状态 $s \in M_k$ 中 α 为真.则 $\models_k$ 可定义如下:

- $M_k, s \models_k p$ 当且仅当 $p \in \mu(s)$;
- $M_k, s \models_k \neg p$ 当且仅当 $p \notin \mu(s)$;
- $M_k, s \models_k \alpha \wedge \beta$ 当且仅当 $(M_k, s \models_k \alpha) \wedge (M_k, s \models_k \beta)$;
- $M_k, s \models_k \alpha \vee \beta$ 当且仅当 $(M_k, s \models_k \alpha) \vee (M_k, s \models_k \beta)$;
- $M_k, s \models_k EF_\varphi \alpha$ 当且仅当 $\exists \pi.(w_0 = s \wedge 0 \leqslant \exists i \leqslant k(M_k, w_i \models_k I_\varphi^i \wedge a))$;
- $M_k, s \models_k E(\alpha U_\varphi \beta)$ 当且仅当 $\exists \pi.(w_0 = s \wedge 0 \leqslant \exists i \leqslant k((M_k, w_i \models_k I_\varphi^i \wedge \beta) \wedge 0 \leqslant \forall i \leqslant j(M_k, w_j \models_k \alpha))$;
- $M_k, s \models_k A(\alpha U_\varphi \beta)$ 当且仅当 $\forall \pi.(w_0 = s \wedge 0 \leqslant \exists i \leqslant k((M_k, w_i \models_k I_\varphi^i \wedge \beta) \wedge 0 \leqslant \forall i \leqslant j(M_k, w_j \models_k \alpha))$.

其中,$\pi = w_0, \ldots, w_k$ 为模型 M 在第 k 步时路径公式,$(w_i, w_{i+1}) \in T, T$ 为模型中的迁移.

在得到 TSTM 和 TCTL 的 BMC 公式之后,即可进行模型检测.设需要验证的软件属性为 f,则在 bd 步之内验证系统设计 D 是否满足性质 f 的 BMC 公式为

$$BMC(D, f, bd) \Leftrightarrow \left(\bigwedge_{k=0}^{bd} step[k] \right) \wedge \left(\bigvee_{k=0}^{bd} \neg f[k] \right).$$

3 实验分析

下面以某型号列车控制系统周期通信环节和站台门联动控制通信过程为例来说明本文方法的实用性,其中,ATP 代表列车,CCS 代表控制中心系统.

3.1 列控系统TSTM模型设计

ATP 与 CCS 周期通信过程如图 2 所示.

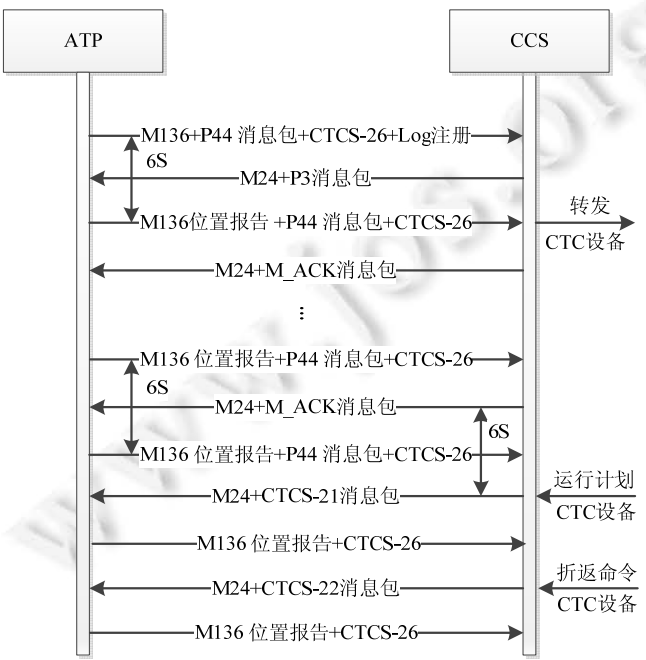


Fig.2 Periodic communication process between ATP and CCS

图 2 ATP 与 CCS 周期通信过程

具体通信流程如下:

(1) CCS 第 1 次接收到 ATP 应答器在管辖范围内的 M136 消息包后,若此前没有向该车发送过配置参数,则立即向该 ATP 发送 M24+P3 消息包.

(2) 当 CCS 检测到 ATP 注册完毕后,以 *T_ATP*=6s 为周期,固定向 ATP 发送 M24 消息包;同时,ATP 应按 *T_ATP* 周期向 CCS 发送 M136 消息包;通信双方若超过 *T_ATP* 时间未收消息包,则内部报警.

(3) 当 CCS 接收到 CTC 设备(调度控制系统)发送的列车运行计划或折返命令时,将在 M24 消息包中加入 CTCS-21 包(运行计划)或 CTCS-22 包(折返命令).

(4) CCS 在接收到 ATP 发送的 CTCS-26(车载状态包)后,应转发给 CTC 设备.

在 ATP 与 CCS 周期通信过程中,会发生站台门联动控制通信过程,如图 3 所示.具体控制流程如下:

(1) CCS 根据 ATP 发送的 CTCS-24 中开门侧信息和股道(站内岔道)信息,计算出所要控制的屏蔽门侧信息:当股道为偶数时,屏蔽门在列车左侧;当股道为奇数时,屏蔽门在列车右侧.

(2) 当 CCS 检测到 ATP 发送的 CTCS-24 中站台编号不在 CCS 管辖范围内时,CCS 应注销列车.

(3) 当 CCS 检测到 ATP 发送的 CTCS-24 中开门侧命令与 CCS 配置门侧信息不一致时,CCS 应注销列车.

(4) CCS 应仅在接受 CTCS-24 中的股道编号和门侧信息正确后,才能向 TCC(技术控制中心)发送更新后的屏蔽门开关命令.

(5) CCS 在收到 ATP 发送的开门命令(CTCS-24 包)后,判断屏蔽门状态,若变为开门状态,则立即向 ATP 回复 CTCS-23 屏蔽门状态数据包.

(6) CCS 在收到 ATP 发送的关门命令(CTCS-24 包)后,判断屏蔽门状态,若变为关门状态,则立即向 ATP 回复 CTCS-23 屏蔽门状态数据包.

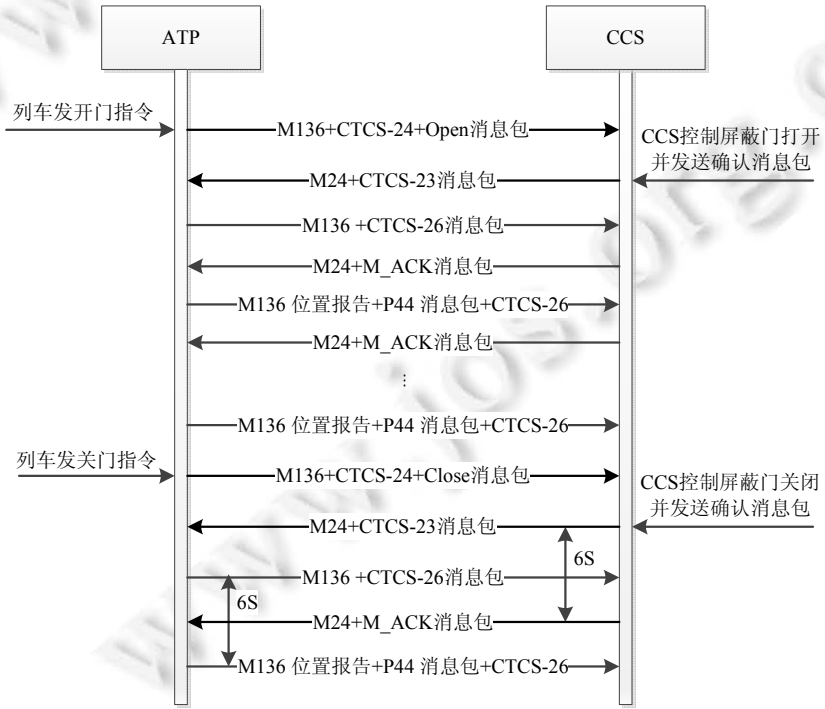


Fig.3 Communication process of linkage control between vehicle door and platform screen door

图 3 车门和站台门联控通信过程

在明确上述建模需求后,即可对其进行 TSTM 建模.由于 CCS 负责与 ATP 进行周期通信以及车门与站台门

联动控制通信,因此逻辑较为复杂.若采用单一 TSTM 表格进行设计,则表格过大.因此,针对 CCS 采用层次化 TSTM 建模方法,将 CCS 功能建模为 3 个表:CCS 表(父表 ZEH_0)、CYCLE 表(子表 ZEH_0.1)和 DOORSCTRL 表(子表 ZEH_0.2).其中,

- CCS 表负责接受 ATP 的 M136 消息包(如图 4 所示)以及 CTC 设备的运行控制指令,除 ATP 注册功能由自身实现外,其余各事件的处理均由 CYCLE 表和 DOORSCTRL 表完成;
- CYCLE 表负责完成 CCS 与 ATP 的周期通信(如图 5 所示);
- DOORSCTRL 表负责完成车门与站台门联动控制通信(如图 6 所示).

ATP 所有功能均由 ATP 表实现(如图 7 所示),包括 ATP 启动、处理 CCS 发送的 M136 消息包事件以及处理车门开关事件.

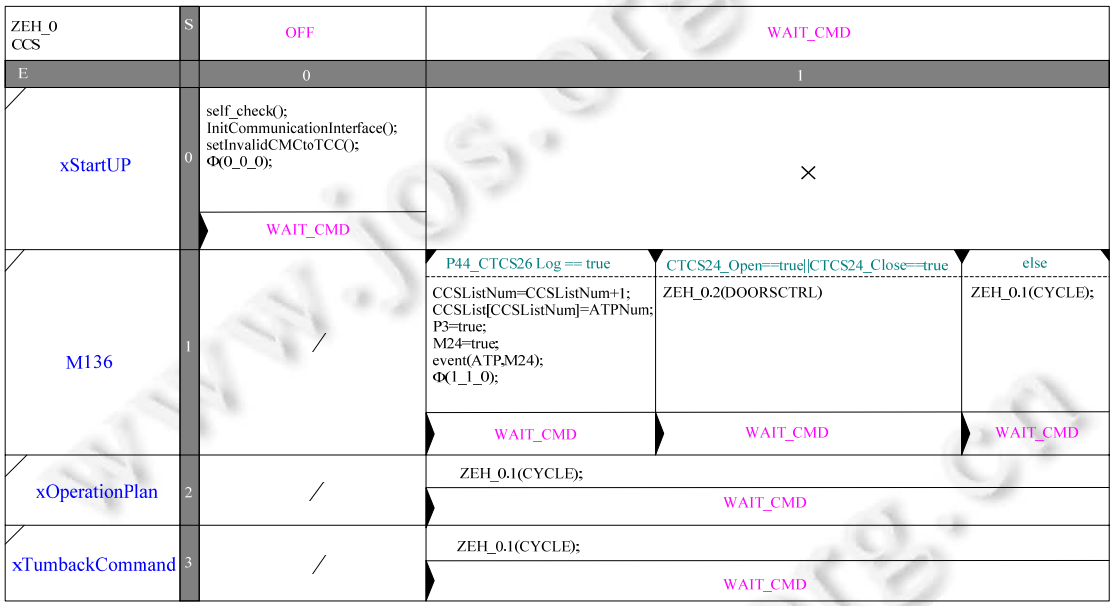


Fig.4 TSTM of CCS system

图 4 CCS 系统的 TSTM

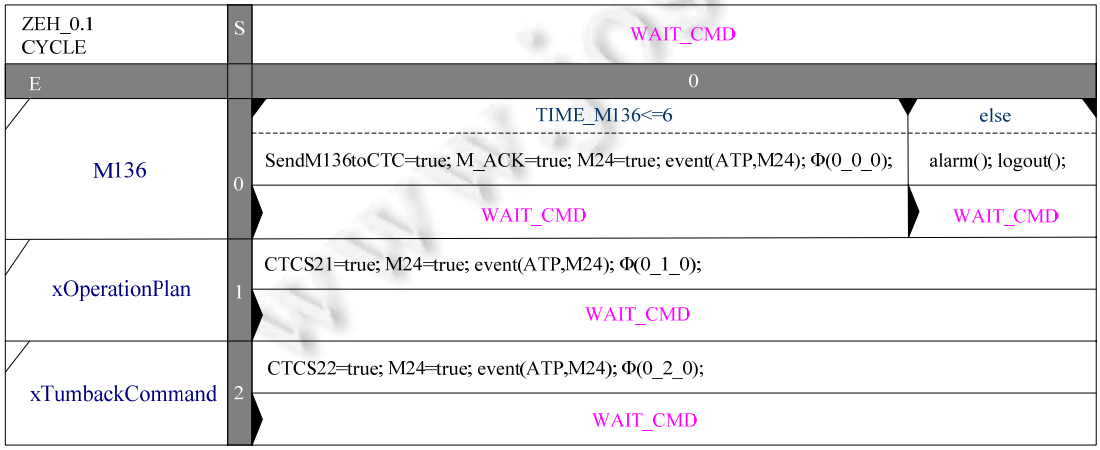


Fig.5 TSTM of periodic communication between ATP and CCS

图 5 ATP 与 CCS 周期通信过程的 TSTM

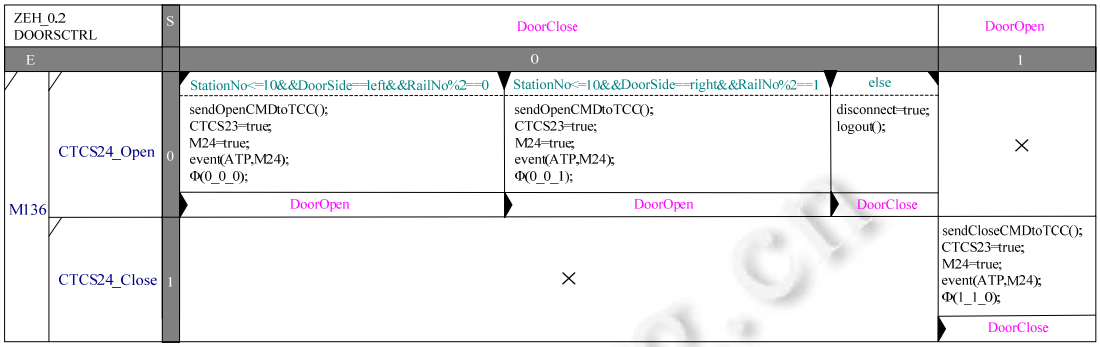


Fig.6 TSTM of communication process of linkage control between vehicle door and platform screen door

图 6 车门和站台门联动通信的 TSTM

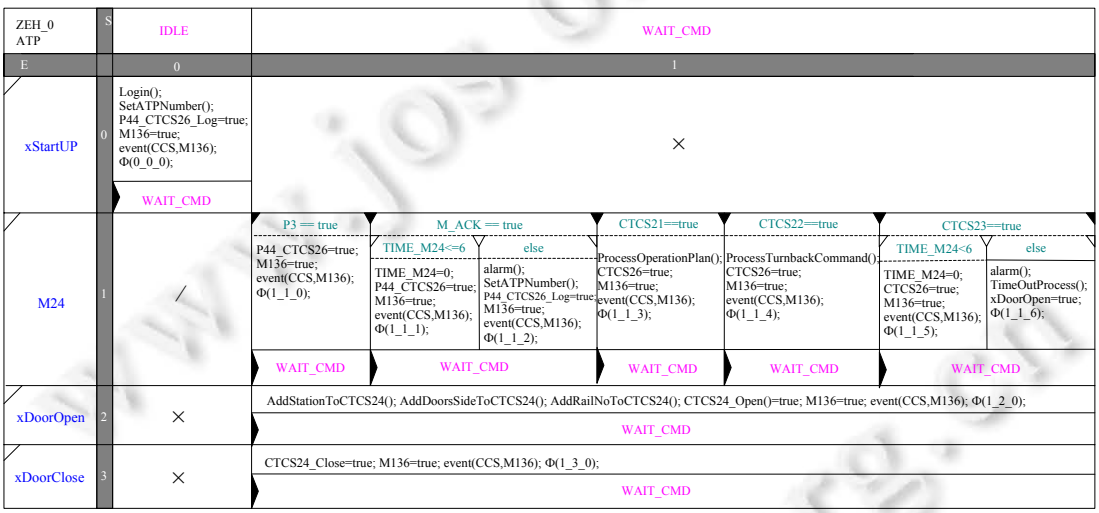


Fig.7 TSTM of ATP

图 7 ATP 的 TSTM

TSTM 为每个可以正常运行的单元格增加时间赋值$\Phi(c_N)$,用于表示 c_N 内部所有操作的时间之和.在本实验各 TSTM 模型中,事件和状态的含义见表 1 和表 2.

Table 1 Meaning of events in TSTM

表 1 TSTM 中事件的含义

事件名(所属 TSTM)	事件含义
xStartUP (CCS)	CCS 系统启动事件
M136 (CCS)	CCS 接收到 ATP 发送的 M136 消息包事件
xOperationPlan (CCS)	CTC 设备向 CCS 发送列车运行计划事件
xTurnbackCommand (CCS)	CTC 设备向 CCS 发送列车折返命令事件
M136 (CYCLE)	CCS 向 CYCLE 传递 ATP 发送的 M136 消息包事件
xOperationPlan (CYCLE)	CCS 向 CYCLE 传递 CTC 发送列车运行计划事件
xTurnbackCommand (CYCLE)	CCS 向 CYCLE 传递 CTC 发送列车折返命令事件
M136 (DOORSCTRL)	CCS 向 DOORSCTRL 传递 M136 消息包事件
CTCS24_Open (DOORSCTRL)	由 M136 消息包传递过来的开门事件
CTCS24_Close (DOORSCTRL)	由 M136 消息包传递过来的关门事件
xStartUP (ATP)	ATP 系统启动事件
M24 (ATP)	ATP 接收到 CCS 发送的 M24 消息包事件
xDoorOpen (ATP)	列车驾驶员向 ATP 系统发开门指令事件
xDoorClose (ATP)	列车驾驶员向 ATP 系统发关门指令事件

Table 2 Meaning of states in TSTM

表 2 TSTM 中状态的含义

状态名(所属 TSTM)	状态含义
OFF (CCS)	CCS 系统处于关闭状态
WAIT_CMD (CCS)	CCS 等待 CTC 设备指令或者 ATM 的 M136 消息包
WAIT_CMD (CYCLE)	CCS 处于与 ATP 正常周期通信的状态
DoorClose (DOORSCTRL)	联动门处于关门状态
DoorOpen (DOORSCTRL)	联动门处于开门状态
IDLE (ATP)	ATP 处于未运行状态
WAIT_CMD (ATP)	ATP 处于运行状态并等待 CCS 的 M24 消息包或者列车控制人员的车门开关指令

各单元格进行的事务处理请见 TSTM 中函数名,这里不再额外进行解释.此外,为压缩 TSTM 表格规模,默认下述 3 种情况不在表中画出:(1) 在 TSTM 模型中的事件(布尔型,事件发生为 true)均由其对应的 c_N 进行置 false 操作;(2) 在 M24 和 M136 消息包周期发送过程中,凡由发送方置 true 的逻辑事件或者处理,均由接收方进行置 false 处理;(3) 对于 CCS 表和 ATP 表,均可通过外部事件触发(关闭动作)返回 OFF(CCS)或 IDLE(ATP)状态.

3.2 列控系统TSTM模型验证

对于 TSTM 模型,有 4 类属性需要进行验证,这些属性对于 TSTM 模型可靠性来说都是至关重要的.

(1) 错误单元格的不可达性

在 TSTM 中,错误单元格是指在某一状态下不应该发生的特定事件所对应的单元格,即模型中标记为"×"的单元格.有两个原因会导致系统进入错误单元格:一个是该单元格本应该为普通单元格,即 $c_N \in C_{normal}$,但由于模型设计人员失误将其标记为错误单元格;另一个是该单元格本身应为错误单元格,但由于系统设计错误,在该单元格对应状态下发生了不该触发的事件,使 TSTM 模型运行进入到错误单元格.上述两种情况都应该避免出现,因此,在 TSTM 模型检测中需要对其进行验证.我们用 $active(H,g)$ 表示在全局可达状态 g 中 TSTM H 的当前激活状态,对于 TSTM H_i 的错误单元格不可达属性可以表示为

$$\forall c_E \in H_i.C_{error}, \forall g \in R_{B(D)}, \neg((active(H_i,g)=source(c_E)) \wedge (event(c_E,g)=\text{true})).$$

在本例中,有 6 个单元格为错误单元格,这里用 $Cell(index(s),index(e))$ 去标识单元格位置,它们分别是 CCS 表的 $Cell(1,0)$、DOORSCTRL 表的 $Cell(1,0)$和 $Cell(0,1)$,ATP 表的 $Cell(1,0)$,$Cell(0,2)$和 $Cell(0,3)$.通过观察上述单元格可以发现:DOORSCTRL 表的 $Cell(1,0)$和 $Cell(0,1)$是由内部事件触发,需要进行不可达性分析;而其他错误单元均为外部事件触发,这里可以不做分析(外部事件不受系统内部行为约束,因此可以随时触发,由其引发的错误单元格是可达的.若将与本模型交互的外部系统也进行整体建模,则这些事件将变为内部事件,由它们引发的错误单元不可达性就需要进行分析).

DOORSCTRL 表的 $Cell(1,0)$表示站台门已处于开门状态,而 CCS 又接收到 ATP 开门请求;DOORSCTRL 表的 $Cell(1,0)$表示站台门已处于关门状态,而 CCS 接收到 ATP 关门请求.这两条属性可以表示为:

- $Unreachability=\neg((CTCS24\_Open=\text{true}) \wedge DoorOpen(DOORSCTRL))$;
- $Unreachability=\neg((CTCS24\_Closs=\text{true}) \wedge DoorCloss(DOORSCTRL))$.

(2) 静态属性

静态属性是指不同 TSTM 模型之间存在某种相关性,这种相关性可以表述如下:如果 TSTM A 处于状态 s_a,则可以推断出 TSTM B 处于(或者不处于)状态 s_b,可以表示为如下公式:

$$\forall g \in R_{B(D)}, active(H_A,g)=s_a \Rightarrow active(H_B,g)=s_b.$$

当 TSTM 模型规模较小时,这类问题看似容易解决;但当模型规模较大时,设计人员就很难从全局掌握这类静态属性.在本例中,我们分析如下几条静态属性(下列蕴含公式的前置条件已在本模型中验证为真):

- $Static\ 1=IDLE(ATP) \Rightarrow WAIT\_CMD(CCS)$;
- $Static\ 2=WAIT\_CMD(ATP) \Rightarrow WAIT\_CMD(CYCLE)$;
- $Static\ 3=WAIT\_CMD(ATP) \Rightarrow DoorOpen(DOORSCTRL)$;

- *Static* 4=*IDLE*(*ATP*)⇒*DoorOpen*(*DOORSCTRL*).

Static 1 表示当 ATP 处于未运行状态时,CCS 处于等待命令状态;*Static* 2 表示当 ATP 处于等待命令状态时,CYCLE 也处于等待命令状态;*Static* 3 表示当 ATP 处于等待命令状态时,DOORSCTRL 处于联动门开门状态;*Static* 4 表示当 ATP 处于未运行状态时,DOORSCTRL 处于联动门开门状态.

(3) 动态属性

动态属性是指当一个 TSTM 模型状态发生变化时,另一个 TSTM 模型处于某个特定状态.即,当 TSTM 模型 *A* 从状态 s_a 迁移到 s'_a 后,TSTM 模型 *B* 应该处于状态 s_b,可以表示为如下公式:

$$\forall g \in R_{B(D)}, active(H_A, g) = s_a \wedge active(H_A, g') = s'_a \Rightarrow active(H_B, g') = s_b.$$

这类属性当 TSTM 模型规模较大时也很难直接观察到,因此需要进行模型检测验证这类问题.在本例中,我们分析如下几条动态属性:

- *Dynamic* 1=*IDLE*(*ATP*)→*WAIT_CMD*(*ATP*)⇒*DoorOpen*(*DOORSCTRL*);
- *Dynamic* 2=*IDLE*(*ATP*)→*WAIT_CMD*(*ATP*)⇒*WAIT_CMD*(*CCS*);
- *Dynamic* 3=*DoorCloss*(*DOORSCTRL*)→*DoorOpen*(*DOORSCTRL*)⇒*WAIT_CMD*(*ATP*).

Dynamic 1 表示当 ATP 由未运行状态迁移到等待命令状态后,DOORSCTRL 处于联动门开门状态;*Dynamic* 2 表示当 ATP 由未运行状态迁移到等待命令状态后,CCS 处于等待命令状态;*Dynamic* 3 表示当 DOORSCTRL 由联动门关状态迁移到联动门开状态后,ATP 处于等待命令状态.

(4) 逻辑及时间属性

TSTM 模型中存在大量逻辑事件,各单元格内部又包含着各类事务处理,这些事件之间、事务处理之间以及事件与事务处理之间存在着逻辑相关性.例如在本例中,当 ATP 的 xDoorOpen 事件触发时,DOORSCTRL 表中的 *sendOpenCMDtoTCC*(·)函数将执行.但在逻辑验证中,我们只关心 *sendOpenCMDtoTCC*(·)函数是否发生,因此可将其定义为布尔型变量,再进行相关验证.在本例中,可以验证如下逻辑属性:

- *Logic* 1=*EF*(*xDoorOpen*==true→*sendOpenCMDtoTCC*==true);
- *Logic* 2=*EF*((*M*136==true∧*CTCS*23=true)→(*M*24==true)).

Logic 1 属性含义为:当事件 xDoorOpen 发生时,是否存在执行路径使 *sendOpenCMDtoTCC*(·)函数执行;*Logic* 2 属性含义为:当 CTCS23 加入到 M136 消息包时,是否存在执行路径使 CCS 发送 M24 消息包.

此外,时间属性也需要进行验证,这对于嵌入式软件和通信类软件尤为重要.TSTM 为每个 $c_N \in C_{normal}$ 增加一个时间赋值 $\Phi(c_N)$ 用于记录单元格内所有操作的消耗时间.这个时间可由模型设计人员根据建模需求直接给出,也可根据单元格内操作复杂程度进行估算(此时,具体操作内容需完整给出.例如,函数 *sendOpenCMDtoTCC*(·)需要给出其函数体代码).在正确标记 $\Phi(c_N)$ 后,验证时间属性时需要在各 c_N 中增加语句 $time=time+\Phi(c_N)$ 实现状态迁移时间累计.

目前,可以验证两个方面的时间属性:一是周期性事件到达时间是否满足时间约束,二是系统状态从 *g* 迁移到 *g*′是否满足时间约束($g,g' \in R_{B(D)}$ 为任意两个全局可达状态).在本例中,可以验证如下时间属性:

- *Time* 1=$EF_{(0,6]}$(*M*24==true→*M*136==true);
- *Time* 2=$EF_{(0,50]}$((*M*136==true∧*P*44_*CTCS*26_*Log*==true)→(*sendM*136*toCTC*=true)).

Time 1 表示在时间约束小于等于 6 秒的情况下,是否存在一条执行路径使得当 M24 消息包触发时,M136 消息包也将被触发;*Time* 2 表示在时间约束小于等于 50s 的情况下,是否存在一条执行路径使得当 *P*44_*CTCS*26_*Log* 加入 M136 消息包并且被发送时,事物处理 *SendM*136*toCTC* 将被执行.

下面可通过第 2 节提出的符号编码方法对上述实例进行编码,把生成的 BMC 逻辑公式转化为 SMT-LIB 2.0 公式,并使用 Z3 v4.3.2(http://z3.codeplex.com/releases)对上述实例属性进行验证,运行环境为(Windows 7 64bit, 2.8GHz,8GB RAM).附录中给出了 TSTM 模型的部分 SMT-LIB 2.0 公式.为了验证本方法的实用性,我们使用 UPPAAL[30]对列控系统需求进行重新建模(如图 8、图 9 所示,这里,CCS 不需要层次化设计),并验证上述属性(属性需改写为 UPPAAL 表示形式),对比结果见表 3.

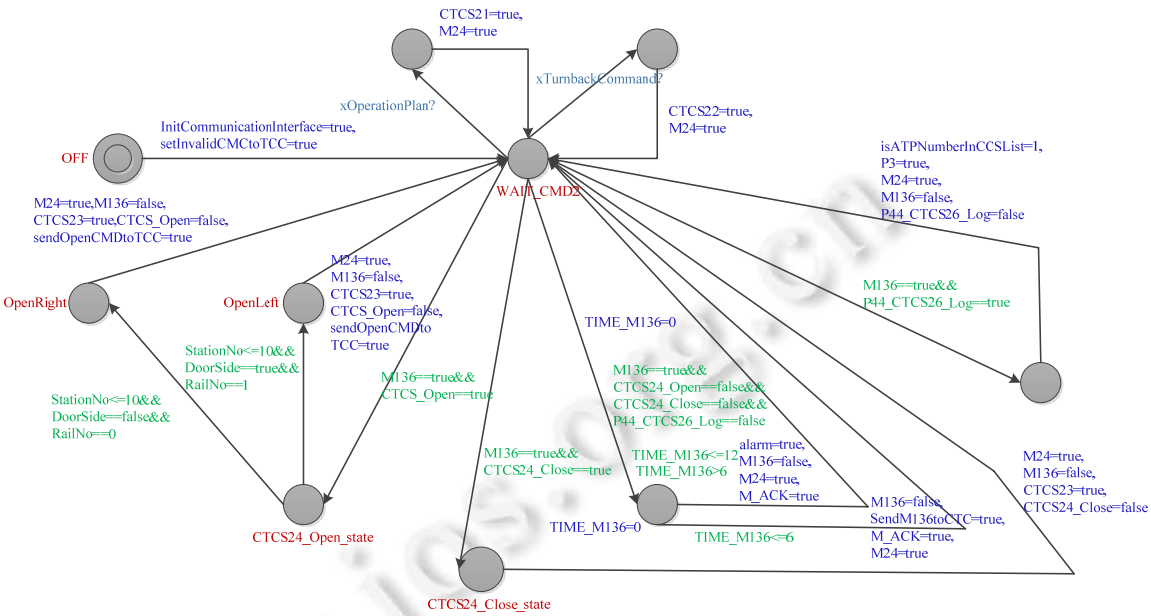


Fig.8 State transition diagram of CCS

图 8 CCS 的状态迁移图

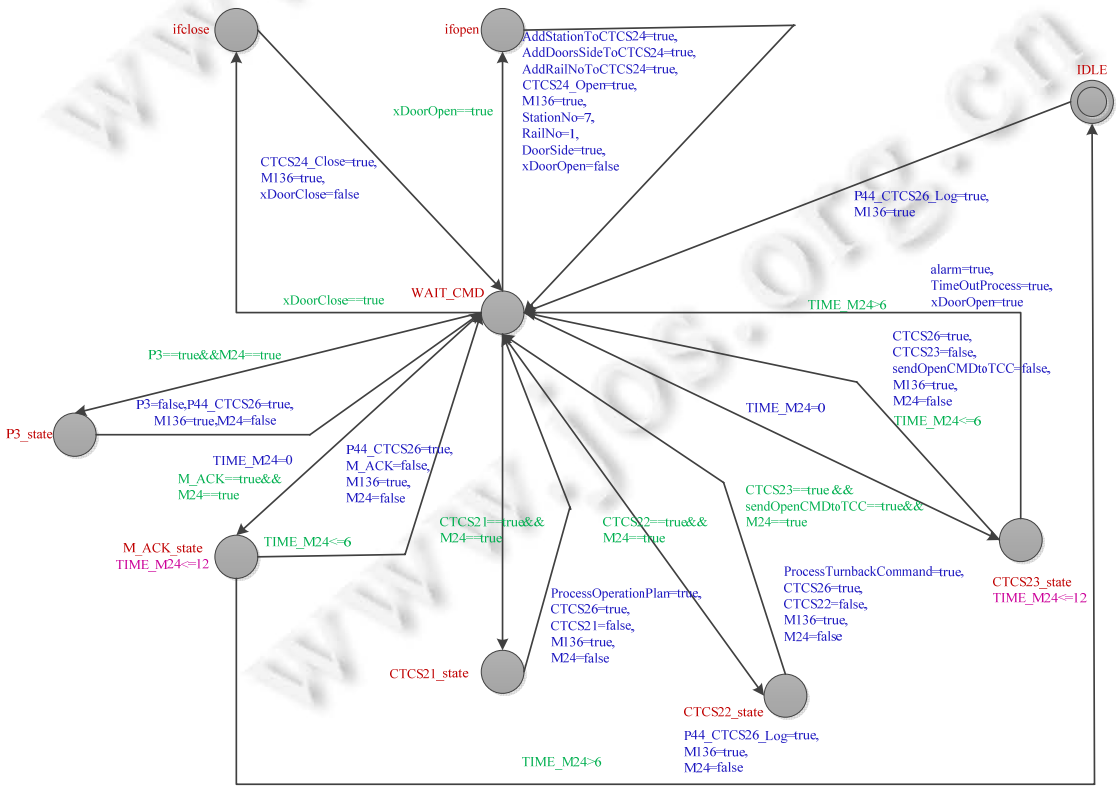


Fig.9 State transition diagram of ATP

图 9 ATP 的状态迁移图

Table 3 Experimental results

表 3 实验结果

属性	Z3			UPPAAL	
	步数	时间	结果	时间	结果
Unreachability 1	50	0.001	*unsat*	0	不能验证
Unreachability 2	50	0.001	*unsat*	0	不能验证
Static 1	50	0.242	*unsat*	0.02	满足
Static 2	50	0.326	*unsat*	0.04	满足
Static 3	50	0.028	*unsat*	0.032	满足
Static 4	32	0.034	*sat*	2.92	不满足
Dynamic 1	15	0.025	*sat*	3.46	不满足
Dynamic 2	50	0.352	*unsat*	0.236	满足
Dynamic 3	50	0.243	*unsat*	0.147	满足
Logic 1	50	0.234	*unsat*	0.258	满足
Logic 2	50	0.262	*unsat*	0.323	满足
Time 1	50	0.042	*unsat*	0.126	满足
Time 2	50	1.756	*unsat*	0	不能验证

在验证过程中,我们引入一些错误属性来检验本文方法的有效性.实验结果见表 3,属性 *Static* 4 及 *Dynamic* 1 的否定形式经判定为 *sat*,说明这两个属性在本 TSTM 模型中不可满足(这两个属性是根据软件规约设计出的错误属性.在实际建模时,要检测模型设计是否存在问题,因此,通过设计满足软件规约的属性,使用本方法可以检测出 TSTM 模型存在的问题).如果实验结果输出为 *unsat*,则可增加有界模型检测步数 k 直到输出结果为 *sat* 或达到预先设定的检测步数最大值 K(本例中为 50).如果满足,说明在 k 步之内找到了一个反例,则该模型被检测出设计漏洞或错误.模型检测工具可以返回一个反例,通过对反例的解读,可以得到性质不成立的原因,为模型修正提供重要线索;反之,如果 SMT 实例不可满足,则表明待验证系统模型运行到 k 阶段时是安全的、没有错误的.

从验证效率来看,在现有模型规模下,本文的验证方法效率与 UPPAAL 相当(UPPAAL 采用了多种优化策略),对不可满足属性验证时间消耗小于 UPPAAL.从验证能力来看,UPPAAL 无法验证错误单元不可达性,原因在于软件建模时,UPPAAL 无法表示 TSTM 中不可达单元,此外,部分属性也无法通过 UPPAAL 表示(例如属性 *Time* 2).TSTM 状态与事件交叉组合出软件所有可能执行情况,这对软件开发阶段查找需求与设计遗漏非常重要,从这个角度来看,带有时间约束的 TSTM 建模能力要优于 UPPAAL,同时也更便于软件开发人员理解与使用.

4 总结与展望

STM 已广泛应用于嵌入式软件开发,但由于缺乏时间语义,限制了其建模能力.针对这一问题,本文提出了一种 TSTM 形式化建模方法,给出了 TSTM 形式化定义,通过为各单元操作引入时间映射函数和时间约束,使 TSTM 具有显示表示时间问题的能力.此外,针对 TSTM 模型验证,给出了基于 TCTL 的 BMC 模型检测方法,并通过实验证明了建模与验证方法有效性.在未来的工作中,还有几个方面的工作需要进一步研究:一方面,TSTM 建模与验证过程中,c_N 执行不可被中断.未来将研究 c_N 执行可被中断的情况,这更符合嵌入式软件特点.但时间属性验证也更为困难,因为这种情况下 $\Phi(c_N)$ 是非确定的.另一方面,未来将在文献[22,23]的基础上进一步研究 TSTM 符号编码优化方法,提高模型验证效率.

致谢 感谢论文评审专家中肯而有益的评审意见和相关编辑严谨、热情的工作;感谢孔维强教授在论文撰写过程中提供的耐心指导与帮助.

References:

[1] Liu C, Zhang W, Zhao HY, Jin Z. "Use Case+Control Case" driven approach for software analysis and design. Ruan Jian Xue Bao/Journal of Software, 2013,24(4):675−695 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4275.htm [doi: 10.3724/SP.J.1001.2013.04275]

[2] Pnueli A. Verification engineering: A future profession. In: Proc. of the 16th Annual ACM Symp. on Principles of Distributed Computing. New York: ACM Press, 1997. [doi: 10.1145/259380.259407]

[3] Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: MIT Press, 1999. 61−87.

[4] Alur R. Model checking: From tools to theory. In: Proc. of the 25 Years of Model Checking. LNCS 5000, Berlin, Heidelberg: Springer-Verlag, 2008. 89−106. [doi: 10.1007/978-3-540-69850-0_6]

[5] Zhang WH. Model checking with SAT-based characterization of ACTL formulas. In: Butler M, Hinchey M, Larrondo-Petrie MM, eds. Proc. of the 9th Int'l Conf. on Formal Engineering Methods (ICFEM 2007). LNCS 4789, Berlin, Heidelberg: Springer-Verlag, 2007. 191−211. [doi: 10.1007/978-3-540-76650-6_12]

[6] Biere A, Cimatti A, Clarke EM, Zhu Y. Symbolic model checking without BDDs. In: Proc. of the Joint European Conf. on Theory and Practice of Software (ETAPS'99). LNCS 1579, Berlin, Heidelberg: Springer-Verlag, 1999. 193−207. [doi: 10.1007/3-540-49059-0_14]

[7] Alur R, Courcoubetis C, Dill DL. Model-Checking in dense real-time. Information and Computation, 1993,(1):2−34. [doi: 10.1006/inco.1993.1024]

[8] Hansson H, Jonsson B. A logic for reasoning about time and reliability. Formal Aspects of Computing, 1994,(6):512−535. [doi: 10.1007/BF01211866]

[9] Li GY, Tang ZS. A linear temporal logic with clocks for verification of real-time systems. Ruan Jian Xue Bao/Journal of Software, 2002,13(1):33−41 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/33.htm

[10] Aziz A, Sanwal K, Singhal V, Brayton R. Verifying continuous time Markov chains. In: Proc. of the 8th Int'l Conf. on Computer Aided Verification (CAV'96). LNCS 4789, Berlin, Heidelberg: Springer-Verlag, 1996. 269−276. [doi: 10.1007/3-540-61474-5_75]

[11] Bryant RE. Graph-Based algorithms for Boolean function manipulation. IEEE Trans. on Computers, 1986,C-35(8):677−691. [doi: 10.1109/TC.1986.1676819]

[12] Wen YJ, Wang J, Qi ZC. Compositional model checking and compositional refinement checking of concurrent reactive systems. Ruan Jian Xue Bao/Journal of Software, 2007,18(6):1270−1281 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1270.htm [doi: 10.1360/jos181270]

[13] Xu L. Improved SMT-based bounded model checking for real-time systems. Ruan Jian XueBao/Journal of Software, 2010,21(7):1491−1502 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/585. htm [doi: 10.3724/SP.J.1001.2010.00585]

[14] Yang JJ, Su KL, Luo XY, Lin H, Xiao YY. Optimization of bounded model checking. Ruan Jian Xue Bao/Journal of Software, 2009,20(8):2005−2014 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3387.htm [doi: 10.3724/SP.J.1001.2009.03387]

[15] Zhou CH, Liu ZF, Wang CD. Bounded model checking for probabilistic computation tree logic. Ruan Jian Xue Bao/Journal of Software, 2012,23(7):1656−1668 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4089.htm [doi: 10.3724/SP.J.1001.2012.04089]

[16] Yan RJ, Li GY, Xu YB, Liu CM, Tang ZS. Reachability checking of finite precision timed automata. Ruan Jian Xue Bao/Journal of Software, 2006,17(1):1−10 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1.htm [doi: 10.1360/jos170001]

[17] Jiang YX, Lin C, Qu Y, Yin H. Research on model-checking based on Petri nets. Ruan Jian Xue Bao/Journal of Software, 2004,15(9):1265−1276 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1265.htm

[18] Bu L, Xie DB. Formal verification of hybrid system. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):219−233 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4535.htm [doi:10.13328/ j.cnki.jos.004535]

[19] Dong W, Wang J, Qi ZC. An approach of model checking UML statecharts. Ruan Jian Xue Bao/Journal of Software, 2003,14(4):750−756 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/750.htm

[20] Zhang C, Duan ZH, Tian C. Specification and verification of UML2.0 sequence diagrams based on event deterministic finite automata. Ruan Jian Xue Bao/Journal of Software, 2011,22(11):2625−2638 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4005.htm [doi: 10.3724/SP.J.1001.2011.04005]

[21] http://www.zipc.com/english/product/zipc/index.html

[22] Kong WQ, Watanabe M, Katayama T, Fukuda A. An SMT-based approach to bounded model checking of designs in communicating state transition matrix. In: Proc. of the 2011 Int'l Conf. on Computational Science and Its Applications (ICCSA). Los Alamitos: IEEE Computer Society, 2011. 159−167. [doi: 10.1109/ICCSA.2011.30]

[23] Kong WQ, Katahira N, Watanabe M, Katayama T, Hisazumi K, Fukuda A. Formal verification of software designs in hierarchical state transition matrix with SMT-based bounded model checking. In: Proc. of the Software Engineering Conf. (APSEC). Los Alamitos: IEEE Computer Society, 2011. 81−88. [doi: 10.1109/APSEC.2011.17]

[24] Biere A, Cimatti A, Clarke E, Fujita M, Zhu Y. Symbolic model checking using SAT procedures instead of BDDs. In: Proc. of the 36th Conf. on Design Automation. New York: ACM Press, 1999. 317−320. [doi: 10.1109/DAC.1999.781333]

[25] Biere A, Cimatti A, Clarke E, Zhu Y. Symbolic model checking without BDDs. In: Cleaveland R, ed. Proc. of the 5th Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems. Berlin: Springer-Verlag, 1999. 193−207. [doi: 10.1007/3-540-49059-0_14]

[26] Penczek W, Wozna B, Zbrzezny A. Bounded model checking for the universal fragment of CTL. Fundamenta Informaticae, 2002, 51(1-2):135−156.

[27] Armando A, Mantovani J, Platania L. Bounded model checking of software using SMT solvers instead of SAT solvers. In: Valmari A, ed. Proc. of the 13th Int'l SPIN Workshop. Berlin: Springer-Verlag, 2006. 146−162. [doi: 10.1007/11691617_9]

[28] Barrett C, Sebastiani R, Seshia SA, Tinelli C. Satisfiability Modulo Theories. Handbook of Satisfiability. Amsterdam: IOS Press, 2009. 825−885.

[29] Emerson EA, Mok A, Sistla AP, Srinivasan J. Quantitative temporal reasoning. In: Clarke EM, Kurshan RP, eds. Proc. of the 2nd Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 1990. 136−145. [doi: 10.1007/BF00355298]

[30] Larsen KG, Pettersson P, Wang Y. UPPAAL in a nutshell. Journal on Software Tools for Technology Transfer, 1997,1(1-2): 134−152. [doi: 10.1007/s100090050010]

附中文参考文献:

[1] 刘春,张伟,赵海燕,金芝.一种"用例+控例"驱动的软件分析与设计方法.软件学报,2013,24(4):675−695. http://www.jos.org.cn/1000-9825/4275.htm [doi: 10.3724/SP.J.1001.2013.04275]

[9] 李广元,唐稚松.带有时钟变量的线性时序逻辑与实时系统验证.软件学报,2002,13(1):33−41. http://www.jos.org.cn/1000-9825/13/33.htm

[12] 文艳军,王戟,齐治昌.并发反应式系统的组合模型检验与组合精化检验.软件学报,2007,18(6):1270−1281. http://www.jos.org.cn/1000-9825/18/1270.htm [doi: 10.1360/jos181270]

[13] 徐亮.改进的以 SMT 为基础的实时系统限界模型检测.软件学报,2010,21(7):1491−1502. http://www.jos.org.cn/1000-9825/585.htm [doi: 10.3724/SP.J.1001.2010.00585]

[14] 杨晋吉,苏开乐,骆翔宇,林瀚,肖茵茵.有界模型检测的优化.软件学报,2009,20(8):2005−2014. http://www.jos.org.cn/1000-9825/3387.htm [doi: 10.3724/SP.J.1001.2009.03387]

[15] 周从华,刘志锋,王昌达.概率计算树逻辑的限界模型检测.软件学报,2012,23(7):1656−1668. http://www.jos.org.cn/1000-9825/4089.htm [doi: 10.3724/SP.J.1001.2012.04089]

[16] 晏荣杰,李广元,徐雨波,刘春明,唐稚松.有限精度时间自动机的可达性检测.软件学报,2006,17(1):1−10. http://www.jos.org.cn/1000-9825/17/1.htm [doi: 10.1360/jos170001]

[17] 蒋屹新,林闯,曲扬,尹浩.基于 Petri 网的模型检测研究.软件学报,2004,15(9):1265−1276. http://www.jos.org.cn/1000-9825/15/1265.htm

[18] 卜磊,解定宝.混成系统形式化验证.软件学报,2014,25(2):219−233. http://www.jos.org.cn/1000-9825/4535.htm [doi:10.13328/j.cnki.jos.004535]

[19] 董威,王戟,齐治昌.UML Statecharts 的模型检验方法.软件学报,2003,14(4):750−756. http://www.jos.org.cn/1000-9825/14/750.htm

[20] 张琛,段振华,田聪.基于事件确定有限自动机的 UML2.0 序列图描述与验证.软件学报,2011,22(11):2625−2638. http://www.jos.org.cn/1000-9825/4005.htm [doi: 10.3724/SP.J.1001.2011.04005]

附 录

模型中的变量:所有事件(布尔型)、单元格中的变量(实型或布尔型)以及单元格中的函数(验证时转为布尔型,因为只需验证其是否发生).TSTM 模型的部分 SMT-LIB 2.0 公式如下:

(set-logic QF_RDL)

(set-info :smt-lib-version 2.0)

;;;;; initial state information

(declare-fun *x*0_0 (·) Bool)

(declare-fun *x*1_0 (·) Bool)

//省略部分定义…

(declare-fun *x*38_0 (·) Real)

(declare-fun *x*39_0 (·) Real)

(assert (and (not *x*0_0) (not *x*1_0) (not *x*2_0) (not *x*3_0) (not *x*4_0) (not *x*5_0) (not *x*6_0) (not *x*7_0) (not *x*8_0) (not *x*9_0) (not *x*10_0) (not *x*11_0) (not *x*12_0) (not *x*13_0) (not *x*14_0) (not *x*15_0) (not *x*16_0) (not *x*17_0) (not *x*18_0) (not *x*19_0) (not *x*20_0) (not *x*21_0) (=*x*22_0 0) (not *x*23_0) (=*x*24_0 0) (not *x*25_0) (not *x*26_0) (not *x*27_0) (not *x*28_0) (=*x*29_0 0) (=*x*30_0 0) (=*x*31_0 0) (not *x*32_0) (not *x*33_0) (not *x*34_0) (not *x*35_0) (=*x*36_0 0) (=*x*37_0

0) (=*x*38_0 0) (=*x*39_0 0)))

;;;;; variables at step 1

;;;;; step variable information

(declare-fun *x*0_1 (·) Bool)

(declare-fun *x*1_1 (·) Bool)

//省略部分定义…

(declare-fun *x*38_1 (·) Real)

(declare-fun *x*39_1 (·) Real)

(declare-fun *BoundTaskStm*_1 (·) Real)

(declare-fun *TranIDStatusEvent*_1 (·) Real)

;;;;; assumptions at step 1 (省略部分公式)

(assert (or (and (=*x*28_0 false) (=*x*0_1 *x*0_0) (=*x*1_1 *x*1_0) (=*x*2_1 *x*2_0) (=*x*3_1 *x*3_0) (=*x*4_1 *x*4_0) (=*x*5_1 *x*5_0) (=*x*6_1 *x*6_0) (=*x*7_1 *x*7_0) (=*x*8_1 *x*8_0) (=*x*9_1 *x*9_0) (=*x*10_1 *x*10_0) (=*x*11_1 *x*11_0) (=*x*12_1 *x*12_0) (=*x*13_1 *x*13_0) (=*x*14_1 *x*14_0) (=*x*15_1 *x*15_0) (=*x*16_1 *x*16_0) (=*x*17_1 *x*17_0) (=*x*18_1 *x*18_0) (=*x*19_1 *x*19_0) (=*x*20_1 *x*20_0) (=*x*21_1 *x*21_0) (=*x*22_1 *x*22_0) (=*x*23_1 *x*23_0) (=*x*24_1 *x*24_0) (=*x*25_1 *x*25_0) (=*x*26_1 *x*26_0) (=*x*27_1 *x*27_0) (=*x*28_1 true) (=*x*29_1 *x*29_0) (=*x*30_1 *x*30_0) (=*x*31_1 *x*31_0) (=*x*32_1 *x*32_0) (=*x*33_1 *x*33_0) (=*x*34_1 *x*34_0) (=*x*35_1 *x*35_0) (=*x*36_1 *x*36_0) (=*x*37_1 *x*37_0) (=*x*38_1 *x*38_0) (=*x*39_1 *x*39_0) (and (=*BoundTaskStm*_1 1025) (=*TranIDStatusEvent*_1 0))) (and (=*x*27_0 false) (=*x*0_1 *x*0_0) (=*x*1_1 *x*1_0) (=*x*2_1 *x*2_0) (=*x*3_1 *x*3_0) (=*x*4_1 *x*4_0) (=*x*5_1 *x*5_0) (=*x*6_1 *x*6_0) (=*x*7_1 *x*7_0) (=*x*8_1 *x*8_0) (=*x*9_1 *x*9_0) (=*x*10_1 *x*10_0) (=*x*11_1 *x*11_0) (=*x*12_1 *x*12_0) (=*x*13_1 *x*13_0) (=*x*14_1 *x*14_0) (=*x*15_1 *x*15_0) (=*x*16_1 *x*16_0) … (=*x*19_1 *x*19_0) (=*x*20_1 *x*20_0) (=*x*21_1 *x*21_0) (=*x*22_1 *x*22_0) (=*x*23_1 *x*23_0) (=*x*24_1 *x*24_0) (=*x*25_1 *x*25_0) (=*x*26_1 *x*26_0) (=*x*27_1 *x*27_0) (=*x*28_1 *x*28_0) (=*x*29_1 *x*29_0) (=*x*30_1 *x*30_0) (=*x*31_1 *x*31_0) (=*x*32_1 *x*32_0) (=*x*33_1 *x*33_0) (=*x*34_1 *x*34_0) (=*x*35_1 *x*35_0) (=*x*36_1 *x*36_0) (=*x*37_1 *x*37_0) (=*x*38_1 *x*38_0) (=*x*39_1 *x*39_0) (and (=*BoundTaskStm*_1 1025) (=*TranIDStatusEvent*_1 4)))))

;;;;; property variable at step 1

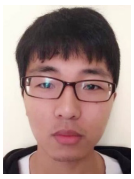
(declare-fun loop1 (·) Bool)

侯刚(1982－),男,辽宁沈阳人,博士生,讲师,CCF 会员,主要研究领域为模型检测.

王洁(1979－),男,博士,讲师,CCF 会员,主要研究领域为可信软件.

周宽久(1966－),男,博士,教授,CCF 高级会员,主要研究领域为可信软件.

李明楚(1963－),男,博士,教授,博士生导师,主要研究领域为信息安全,信任模型.



常军旺(1989－),男,硕士生,主要研究领域为模型检测.