软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(6):1830−1848 [doi: 10.13328/j.cnki.jos.006252]

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

Ptolemy 离散事件模型形式化验证方法*

陆芝浩[1,2], 王 瑞[1,2], 孔 辉[3], 关 永[1,4], 施智平[1,2]

[1](首都师范大学 信息工程学院,北京 100048)

[2](轻型工业机器人与安全验证北京市重点实验室(首都师范大学),北京 100048)

[3](华为技术上海研发中心,上海 201206)

[4](电子系统可靠性与数理交叉学科国家国际科技合作示范基地(首都师范大学),北京 100048)

通讯作者: 王瑞, E-mail: rwang04@cnu.edu.cn

摘 要: Ptolemy 是一个广泛应用于信息物理融合系统的建模和仿真工具包,主要通过仿真的方式保证所建模型的正确性.形式化方法是保证系统正确性的重要方法之一.提出了一种基于形式模型转换的方法来验证离散事件模型的正确性.离散事件模型根据不同事件的时间戳触发组件,时间自动机模型能够表达这个特征,因此选用 Uppaal 作为验证工具.首先定义了离散事件模型的形式语义;其次设计了一组从离散事件模型到时间自动机的映射规则;然后在 Ptolemy 环境中实现了一个插件,可以自动将离散事件模型转换为时间自动机模型,并通过调用 Uppaal 验证内核完成验证;最后,以一个交通信号灯控制系统为例进行了成功的转换和验证,实验结果证实了该方法能够验证 Ptolemy 离散事件模型的正确性.

关键词: 形式化验证;Ptolemy 离散事件模型;模型自动转换;时间自动机

中图法分类号: TP311

中文引用格式: 陆芝浩,王瑞,孔辉,关永,施智平.Ptolemy 离散事件模型形式化验证方法.软件学报,2021,32(6):1830−1848. http://www.jos.org.cn/1000-9825/6252.htm

英文引用格式: Lu ZH, Wang R, Kong H, Guan Y, Shi ZP. Formal verification of Ptolemy discrete event model. Ruan Jian Xue Bao/Journal of Software, 2021,32(6):1830−1848 (in Chinese). http://www.jos.org.cn/1000-9825/6252.htm

Formal Verification of Ptolemy Discrete Event Model

LU Zhi-Hao[1,2], WANG Rui[1,2], KONG Hui[3], GUAN Yong[1,4], SHI Zhi-Ping[1,2]

[1](College of Information Engineering, Capital Normal University, Beijing 100048, China)

[2](Beijing Key Laboratory of Light Industrial Robot and Safety Verification (Capital Normal University), Beijing 100048, China)

[3](Huawei Technologies Shanghai R&D Center, Shanghai 201206, China)

[4](National International Science and Technology Cooperation Demonstration Base of Interdisciplinary of Electronic System Reliability and Mathematics (Capital Normal University), Beijing 100048, China)

Abstract: Ptolemy is a modeling and simulation toolkit widely used in cyber physical systems, which ensures the correctness of the models through simulation. Formal verification is one of the important methods to guarantee the correctness of systems. This paper presents a model translation based approach to verify the Ptolemy discrete event model. The discrete event model fires according to the timestamp of different events, and the timed automaton can express this feature. Therefore, Uppaal is a suitable verification tool. First, the semantic of discrete event model is defined. And a set of mapping rules are designed to represent the discrete event model with a network

* 基金项目: 国家自然科学基金(61877040, 61876111); 首都师范大学交叉研究院项目(19530012005)

Foundation item: National Natural Science Foundation of China (61877040, 61876111); Cross Research Institute of Capital Normal University (19530012005)

本文由“形式化方法与应用”专题特约编辑姜宇副教授推荐.

收稿时间: 2020-08-30; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-02-07

of timed automata. Then, a plug-in is implemented in the Ptolemy environment that automatically translated the discrete event model into a network of timed automata, and verifies the network of timed automata by calling the Uppaal validation kernel. Finally, a case of traffic light control system is successfully translated and verified, and the experimental results confirm that the proposed approach is reliable and effective for the verification of Ptolemy discrete event model.

Key words: formal verification; Ptolemy DE model; automatic model translation; timed automata

近些年来,随着计算机技术的发展,物理信息融合系统(cyber-physical system,简称 CPS)已经广泛应用到社会中的各个领域,发挥着举足轻重的作用.航空、军事、医疗等高可靠领域对系统正确性的要求尤为突出.然而,在传统的系统开发过程中,通常采用测试的方法验证系统正确性.测试一般是在系统开发结束才执行,发现问题晚,修改的代价是巨大的.因此,一个有效的设计开发过程必须在早期包含对系统正确性的分析,以确保实现的系统满足正确性需求.

模型驱动开发的核心思想是分离业务分析与业务实现,将开发人员的关注点转移到业务领域建模上,模型在开发过程中起到主线的作用.当模型构建完成后,可以对模型进行验证,这样就可以在整个开发过程的早期发现问题.Ptolemy[1]是由加州大学伯克利分校的团队设计、开发的一个研究并发、实时嵌入式系统建模、仿真的集成环境,其依靠其开源、面向角色以及支持异构等方面的优点,在模型驱动开发过程中,越来越受到业内人员的广泛关注.Ptolemy 在建模、仿真方面拥有强大的功能,但是缺乏形式化验证,只能通过仿真来检测模型正确性.然而,这种方法是不完备的,因为对模型进行仿真,仅仅是检测模型中的某条路径,无法对模型进行穷尽搜索,形式化验证则可以高效全面地检测模型的正确性[2].

模型检测是一种重要的形式化方法[3],其基本思想是:用状态迁移系统表示系统的行为,用时序逻辑公式描述系统的性质.这样,“系统是否具有所期望的性质”就转化为数学问题“状态迁移系统是否是公式的一个模型?”.对有穷状态系统,这个问题是可判定的,即可以用计算机程序在有限时间内自动确定.时间自动机是一种可以指定时间的模型,并拥有相应的分析与验证工具 Uppaal,而且作为模型检测基础的时序逻辑可以方便地表示离散事件或状态是否发生以及发生的先后顺序,所以,本文提出了将 Ptolemy 离散事件模型转换为时间自动机模型进行形式化验证的方法.我们首先定义了 Ptolemy 离散事件模型的语义;其次,设计了一组将离散事件模型转换为时间自动机的映射规则;然后,在 Ptolemy 环境中实现了一个插件,可以自动将离散事件模型转换为时间自动机模型,进而通过调用 Uppaal 验证内核进行形式化验证,验证系统的正确性需求.

本文中,我们的主要贡献在于:(1) 定义了更为完整的 Ptolemy 组件映射规则,可以支持更多的组件;(2) 将 Ptolemy 模型中的各个组件转换为时间自动机,在上层组件转换生成的时间自动机和下层组件转换生成的时间自动机之间设置一个同步通道,用于上下层之间传递信息,保留了模型层次化的特点,有效避免了摊平层次化组件时可能造成的状态空间爆炸;(3) 避免使用 tick 计时,降低验证的复杂度;(4) 在 Ptolemy 环境中设计实现了一个自动转换验证的插件,完善了使用 Ptolemy 进行模型驱动开发的工具链.

本文第 1 节介绍 Ptolemy 离散事件模型.第 2 节介绍时间自动机模型.第 3 节是将 Ptolemy 离散事件模型转换为时间自动机模型的映射规则.第 4 节介绍 Ptolemy 离散事件模型验证框架及其实现.第 5 节是一个具体的案例分析.第 6 节介绍相关工作.第 7 节是本文的总结与展望.

1 Ptolemy 离散事件模型

Ptolemy 是一个开源的建模和仿真工具,与其他建模工具不同,Ptolemy 支持层次化异构建模.Ptolemy 的一个关键目标就是将不同领域之间的语法、语义和语用之间的差异最小化,并将不同领域设计之间的互操作性最大化.

1.1 Ptolemy模型语法

Ptolemy 支持面向组件的层次化建模,如图 1 是一个 Ptolemy 层次化异构模型的示意图.D_1 和 D_2 是指示器(director),指明 Ptolemy 模型的语义域(semantic domain),不同语义域下,组件之间的交互是不同的,体现了

Ptolemy 模型支持异构的特点.A_0~A_6 是 Ptolemy 模型中的组件(actor),其中,A_0 和 A_2 是复合组件(composite actor):A_0 表示当前的 Ptolemy 模型;A_2 是模型内嵌的一个复合组件,体现了 Ptolemy 模型层次化的特点.P_0~P_{11} 分别表示组件的输入和输出接口(port).

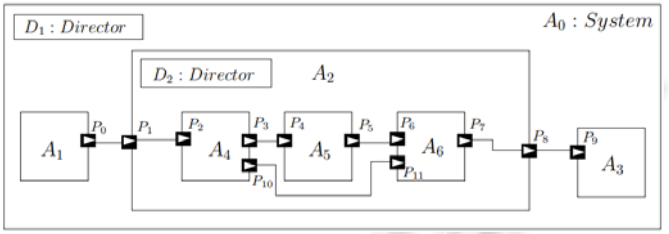


Fig.1 Visual rendition of a Ptolemy hierarchy model

图 1 Ptolemy 层次化模型示意图

Ptolemy 模型中的组件是一个四元组 $A(E,P,Para,R)$,其中,

- E:有限的组件集合;
- P:有限的接口集合,$P=P_{in}\cup P_{out}$,其中,P_{in} 是输入接口集合,P_{out} 是输出接口集合;
- $Para$:=(name,attribute),有限的属性参数集合;
- R:有限的关系(relation)集合,包含两种情况.

(1) 两个内部组件的端口之间的关系;

(2) 一个组件的端口与一个组件的内部组件的端口之间的关系:

$$R:=\{(p_i,Para_{ij},p_j)|p_i\in P_x,p_j\in P_y,A_x=\{E_x,P_x,Para_x,R_x\}\in E,A_y=\{E_y,P_y,Para_y,R_y\}\in E\}\cup$$
$$\{(p_i,Para_{ij},p_j)|p_j\in P_x,p_i\in P,A_x=\{E_x,P_x,Para_x,R_x\}\in E\}\cup$$
$$\{(p_i,Para_{ij},p_j)|p_i\in P_x,p_j\in P,A_x=\{E_x,P_x,Para_x,R_x\}\in E\},$$

其中,$Para_{ij}$={(name,attribute)}是关系的参数集.

Director 控制组件的执行顺序,协调组件之间的交互.Director 本质上也是一个组件 $A_{dir}=(\varnothing,\varnothing,Para_{dir},\varnothing)$,$Para_{dir}$ 包含一个元素($director$,$semantics$),$semantics$ 是集合{DE,FSM,SR,CT,$PTIDES$}中的一个元素.异构模型由不同层次的不同 $director$ 组成,组件的语义依赖于控制它们的 $director$.一个系统就是一个层次化的组件,它的层次结构是由其内部组件集合递归定义的.如图 1 所示的系统可以表示为 $A_0(E_0,P_0,Para_0,R_0)$,$E_0=\{A_1,A_2,A_3,D_1\}$,$P_0=\varnothing$,$R_0=\{(P_0,P_1),(P_8,P_9)\}$,$A_1$ 是原子组件,A_2 是层次化的复合组件,Ptolemy 中的原子组件还可以通过参数 $Para_{pri}$={($priority$,$value$)}定义组件的优先级.

1.2 Ptolemy离散事件语义

Ptolemy 中的语义域定义了组件交互的“物理定律”,它为组件之间的并发执行以及两个组件之间的通信提供了管理规则[4].在离散事件(discrete-event,简称 DE)域,组件通过位于同一时间轴的事件进行通信.每个事件都有一个值和时间戳,并且组件对事件的处理是按照时间顺序的.由组件产生的输出事件在时间上不得早于被消耗的输入事件.该模型按照一个全局的事件队列进行执行.当一个组件产生输出事件时,该事件根据其时间插入队列.在 DE 域模型的每次迭代中,最小时间戳的事件从全局事件队列中删除,触发它的目标组件执行.

DE 域用于并发组件之间定时、离散的交互建模.一次交互即为一个事件,事件在概念上可以理解为从一个组件发送给另一个组件的瞬时消息.定义 DE 模型中组件执行顺序的原则是:向组件 B 发送事件的组件 A 会更早的触发执行.

算法 1 是 Ptolemy DE 模型语义描述.第 1 行定义一个空的事件队列;第 2 行~第 4 行初始化模型中的每个组件,若组件产生事件,则插入到事件队列中;第 5 行~第 12 行表示当事件队列不为空时,不断地取出队列中的头事件,然后根据该事件获取目标组件,触发目标组件执行.目标组件执行过程中会产生新的事件,插入事件队列中.

当事件队列为空时,说明模型的一次执行结束.

算法 1. Semantics of Ptolemy Discrete Event Model.

```
1:  Q:=∅;
2:  for each actor A in model do
3:      Ainit(·)
4:  end for
5:  while Q is not empty do
6:      event:=head event of Q
7:      remove event from Q
8:      actorToFire:=getActorByEvent(event)
9:      actorToFire.prefire(·)
10:     actorToFire.fire(·)
11:     actorToFire.postfire(·)
13: end while
```

在 DE 模型中,消息通过组件之间的连接进行瞬时传递,模型中时间的推移是通过时钟组件实现的.DE 模型按照时间顺序进行执行,首先执行时间最早的事件(拥有最早的时间戳).随着事件的处理,在模型内和外部世界的时间都会向前推移,因此逻辑时间和物理时间之间会产生混淆.逻辑时间是模型中的时间,物理时间是模型执行时现实世界中消耗的时间.逻辑时间可能会比物理时间走的更快或更慢.

DE 控制器中的 *synchronizeToRealTime* 参数设置为 true 时,可以在某种程度上让这两个时间同步,它通过延缓模型的执行以便物理时间追上逻辑时间.当然,这只有在计算机足够快地执行这个模型使得逻辑时间远远快于物理时间的情况下才起作用.当这个参数设置为 true 时,逻辑时间值以秒为单位,否则时间单位是任意的.

2 时间自动机模型

时间自动机[5]是在传统自动机上进行实时扩展.设 V 是有限变量集合,C 是有限时钟变量集合,X 是有限数据变量集合,有 $V=C\cup X$ 且 $C\cap X=\varnothing$.所有的时钟变量同步向前演进.我们用 $\psi(V)$ 表示位置约束和守卫函数的集合,有 $\psi::=e|\psi\cap\psi,e$ 是形如 $c\sim n$ 或者 $x\sim n$ 的表达式,其中,$c\in C,x\in X,\sim\in\{\leqslant,\geqslant,=,<,>\}$.赋值操作定义为 $v:=e,v\in V$.我们用 U 表示所有的赋值函数.

2.1 时间自动机语义

时间自动机是一个六元组 $\lambda(L,l_0,K,V,I,T)$,其中,

- L:有限的位置集合;
- l_0:初始位置;
- K:有限的动作集合;
- V:有限的变量集合;
- $I:L\to\psi(V)$是位置约束函数;
- $T:T\subseteq L\times\psi\times K\times U\times L$ 是有限的边的集合.

每条边有一个源位置 l、一个目标位置 l'.当边上的守卫 $g\in\psi$满足时,迁移发生并且变量集 V 中的部分变量通过函数 $r\in U$ 更新.边 $t\langle l,g,k,r,l'\rangle$常写作 $l\xrightarrow{g,k,r}l'$.

自动机初始状态位于初始位置 l_0,且所有时钟变量初始化为 0.随着时间流逝,所有的时钟变量按照相同的频率增长,且满足位置约束 $I(l_0)$.当该位置上的变量满足边上的使能守卫 g 时,系统可以继续停留在这个位置或者迁移到位置 l'.当动作 k 发生时,变量根据函数 r 的定义发生改变[6].

时间自动机 $\lambda(L,l_0,K,V,I,T)$的语义定义为一个标号变迁系统 $S(\lambda)=\langle S,s_0,\to\rangle$.$S\subseteq L\times U$ 是状态的集合,s_0 是初始状态,$\to\subseteq S\times(U\cup K)\times S$ 是迁移的集合,分为两类.

- 时间流逝迁移:对 $d\in R^+$,有 $(l,u)\xrightarrow{d}(l,u+d)$,当$\forall d'\leqslant d$,$u$ 和 $u+d'$满足 $I(l)$;
- 位置变化迁移: $(l,u)\xrightarrow{k}(l',u')$,如果$\exists t(l,g,k,r,l')\in T$,$u'=r(u)$,$u$ 满足守卫 g,且 u'满足 $I(l')$,u 表示时间自动机中时钟的当前时间.

嵌入式系统通常由多个模块交互组成,因此需要引入时间自动机网 $\bar{\lambda}=\lambda_1\|...\|\lambda_n$,且 $\lambda_i=(L_i,l_i^0,K_i,V_i,I_i,T_i)$,$n$ 是自动机的个数,这些自动机共享一组相同的动作集合.向量 $\bar{l}=(l_1,...,l_n)$ 是时间自动机网 $\bar{\lambda}$ 的位置向量.位置约束函数 $I(\bar{l})$ 是所有自动机λ_i约束的并集, $I(\bar{l})=\wedge_i I_i(l_i)$,其中, $\bar{l}[l_i'/l_i]$ 表示 $\bar{L}$ 中的 l_i 由 l_i' 取代.

时间自动机网的语义定义为迁移系统 $S(\bar{\lambda})=\langle S,s_0,\rightarrow\rangle$,$S=(L_1\times\ldots\times L_n)\times U$ 是状态集合,s_0 是初始状态,$\rightarrow\subseteq S\times S$ 是迁移关系,定义为:

- $(\bar{l},u)\xrightarrow{d}(\bar{l},u+d)$,其中,$d\in R^+$,如果$\forall d'\leqslant d$,$u$ 和 $u+d'$满足 $I(l)$;
- $(\bar{l},u)\xrightarrow{k}(\bar{l}[l_i'/l_i],u)$,如果$\exists t(l_i,g,k,r,l_i')\in T$,$u'=r(u)$,$u$ 满足守卫 g,且 u'满足 $I(\bar{l})$;
- $(\bar{l},u)\xrightarrow{k}(\bar{l}[l_j'/l_j,l_i'/l_i],u')$,如果存在一条边 $l_i\xrightarrow{g_i,sync?,r_i}l_i'$ 和另一条边 $l_j\xrightarrow{g_j,sync!,r_j}l_j'$,$u'=r_i\cup r_j(u)$,$u$ 满足守卫 $g_i\wedge g_j$,且 u'满足 $I(\bar{l})$,*sync* 是同步信号,*sync*?表示接收到信号 *sync*,*sync*!表示发送信号 *sync*.

2.2 时间自动机模型验证工具Uppaal

Uppaal[7]是一个用于实时系统建模、仿真和验证的集成环境,由瑞典乌普萨拉大学和丹麦奥尔堡大学联合开发.它适用于对带有时间的系统进行建模,并通过同步通道或共享变量实现各个时间自动机之间的通信.

Uppaal 主要由 3 部分组成:描述语言、模拟器和模型检查器.描述语言是一种具有简单数据类型(如整数、布尔型等)的非确定性保护命令语言,可以将系统行为描述为由时钟和数据变量扩展的时间自动机网络.模拟器是一种验证工具,它可以通过仿真检查系统可能的动态执行情况,从而在对模型进行形式化验证之前提供一种简单的故障检测手段;而且当模型检查器检查出错误时,可以在此追溯路径.模型检查器是通过探索系统的状态空间来检查系统的不变性和活性,即通过约束表示的符号状态进行可达性分析,从而对模型进行形式化验证.Uppaal 中常用的数据类型包括整形(int)、布尔型(bool)、时钟(clock)和同步通道(chan),广播使用 broadcast chan 声明.

Uppaal 的主要思想是利用时间自动机来建模、仿真一个系统,优点在于高效性和便捷性.Uppaal 通过快速搜索机制来验证系统规范和可达性,可以快速有效地对系统进行形式化验证.模型检查引擎通过 CTL 公式对系统进行形式化验证[8],公式规则如下.

- *E*<>*p*:存在一条路径,使得 *p* 最终成立;
- *A*[]*p*:对所有的路径,*p* 一直成立;
- *E*[]*p*:存在一条路径,使得 *p* 一直成立;
- *A*<>*p*:对所有的路径,*p* 最终成立;
- *p* imply *q*:当 *p* 成立时,*q* 最终一定成立.

3 映射规则

如图 2 所示.

- A_0 是一个 Ptolemy DE 模型,A_1~A_3 分别是模型中的组件;
- p_0~p_3 是组件的输入输出接口,(p_0,p_1)和(p_2,p_3)分别是 A_1 和 A_2、A_2 和 A_3 间的关系;
- T_0 是一个时间自动机网络,T_1~T_3 分别表示组件 A_1~A_3 转换生成的时间自动机;
- *Temp* 是时间自动机中与组件功能相同的的具体实现;
- p_{01} 和 p_{23} 分别是 T_1 和 T_2、T_2 和 T_3 间的同步通道;
- *GD* 和 *TD* 分别表示组件接口和参数转换生成的全局变量和局部变量.

Ptolemy DE 模型转换生成一个时间自动机网络.模型中的每个组件转换成一个时间自动机,组件的接口和

参数分别转换成时间自动机中的全局变量和局部变量,组件间的关系转换成一组同步通道,实现各个时间自动机间可以通过同步通道通信,从而构成一个时间自动机网络.其中,由于 Ptolemy DE 模型与时间自动机模型在复杂数据类型(如 double、矩阵)和复杂计算(如三角函数、指对数函数)方面的支持力度不同,所以本文中的方法暂未涉及对这些复杂数据类型和复杂计算的转换规则.

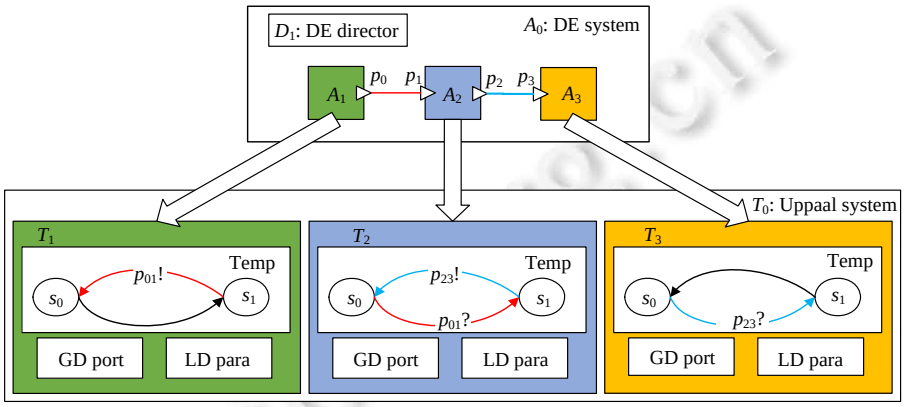


Fig.2 Visual rendition of the map of ptolemy DE model to network of automata

图 2 Ptolemy DE 模型到时间自动机网络的映射关系示意图

Ptolemy DE 模型本质上是一个复合组件 $A=(E,\varnothing,Para,R)$,其映射规则如下:

$$Translate(A)\cong Trans\_Actors(E)\cap Trans\_Paras(Para)\cap Trans\_Relas(R)$$

$$Trans\_Actors(E)\cong Trans\_Actor(E[1])\cap\ldots\cap Trans\_Actor(E[x])$$

$$Trans\_Paras(Para)\cong Trans\_Para(Para[1])\cap\ldots\cap Trans\_Para(Para[y])$$

$$Trans\_Relas(R)\cong Trans\_Rela(R[1])\cap\ldots\cap Trans\_Rela(R[z])$$

参数 x 表示 A 中的组件个数,y 表示 A 中的参数个数,z 表示 A 中的关系个数,$E[i]$,$Para[i]$,$R[i]$分别表示 E,$Para$,R 中的元素.Ptolemy DE 系统的转换包括组件集合 E 的转换、参数集合 $Para$ 的转换和系统中组件间关系 R 的转换.Ptolemy DE 系统最终被转换成一个等价的时间自动机网络.

3.1 数据类型

Ptolemy 支持丰富的数据类型,但是时间自动机却并非如此.为了将 Ptolemy DE 模型转换为等价的时间自动机模型,设计对 Ptolemy 中数据类型的转换显得尤为重要.本文提出:

(1) 将 Ptolemy 中 int 和 bool 类型的数据直接转换为时间自动机中 int 和 bool 类型的数据;

(2) Ptolemy DE 模型中的时间组件具有 *period* 和 *offsets* 两个参数:*period* 表示周期,*offsets* 表示偏移量,是一个长度为 n 的数组($offsets[i]<offsets[j]<period, i<j<n$).时间组件在 *time* 时刻触发,$time=b\times period+offsets[i]$($b\in N$).为了保证时间自动机也表达这一特征,除转换参数 *period*,*offsets* 和 *time* 的计算外,还需为其定义一个局部时钟变量 t,并将 $Edge(l,g,k,r,l')$中 l 状态的 invariant 设置为 $t\leqslant time$,g 设置为 $g\geqslant time$,此时即可满足时间自动机在 *time* 时刻才能发生迁移.k 表示和其他的时间自动机交互通信,r 用于完成对输入和输出接口、参数及 b 和 i 的赋值修改操作.

3.2 关系

两个内部组件的端口或一个组件的端口与一个组件的内部组件的端口之间的关系 $R=(p,Para,q)$:

$$Trans\_Rela(R)\cong Trans\_ROP(p)\cap Trans\_RIP(q)$$

$$Trans\_ROP(p)\cong Edge_a(l_a,\phi,p!,r_a,l'_a)$$

$$Trans\_RIP(q)\cong Edge_b(l_b,\phi,p?,r_b,l'_b)$$

R 是连接组件 A 的输出端口 p 与组件 B 的输入端口 q 的关系,对应转换成一组同步迁移.$Edge_a(l_a,\varnothing,p!,r_a,l'_a)$

是组件 A 转换成的时间自动机中用于发送同步信号的迁移,$Edge_b(l_b,\varnothing,p?,r_b,l_b')$是组件 B 转换成的时间自动机中用于接收同步信号的迁移,各个时间自动机通过这样的同步通道构成一个时间自动机网络.

3.3 数据类型

Ptolemy 是面向组件建模的,因此针对不同类型的组件,定义了不同的映射规则.

3.3.1 特定组件

Ptolemy DE 系统中的组件 $A=(\varnothing,P,Para,R)$,其映射规则如下:

$$Trans\_Actor(A)\cong Trans\_Ports(P)\cap Trans\_Paras(Para)\cap Trans\_Relas(R)\cap Trans\_Func(A)$$

$$Trans\_Ports(P)\cong Trans\_InputPs(IP)\cap Trans\_OutputPs(OP)$$

$$Trans\_InputPs(IP)\cong Trans\_InputP(IP[1])\cap\ldots\cap Trans\_InputP(IP[m])$$

$$Trans\_InputP(IP[i])\rightarrow\langle \text{int},\text{bool}\rangle$$

$$Trans\_OutputPs(OP)\cong Trans\_OutputP(OP[1])\cap\ldots\cap Trans\_OutputP(OP[n])$$

$$Trans\_OutputP(OP[i])\rightarrow\langle \text{int},\text{bool},\text{chan}\rangle$$

$$Trans\_Paras(Para)\cong Trans\_Para(Para[1])\cap\ldots\cap Trans\_Para(Para[y])$$

$$Trans\_Relas(R)\cong Trans\_Rela(R[1])\cap\ldots\cap Trans\_Rela(R[z])$$

$$Trans\_Func(A)\cong Edges(l,g,k,r,l')$$

参数 m 表示组件 A 中的输入接口的个数,n 表示组件 A 中的输出接口的个数.组件的输入接口转换成时间自动机中的一组 int 和 bool 类型的全局变量,输出接口转换成一组 int,bool 和 chan 类型的全局变量,参数转换成时间自动机中的局部变量,根据原子组件功能,生成时间自动机中具体的 template,最终原子组件被转换成一个功能相同的时间自动机.Ptolemy DE 模型语义要求向组件 B 发送事件的组件 A 会更早地触发执行,当组件转换为时间自动机时,为保持语义的一致性,为每个时间自动机定义一个 *depth* 属性.首先计算组件在 Ptolemy DE 模型中的 *level*,时间自动机的 *depth* 由组件的 *level* 计算而来.组件的 *level* 表示该组件到达源组件(没有上游组件)或者延迟组件沿路径历经的最大上游组件个数.但是此时计算出的 *level* 存在相等的情况,这时需要判断相同 *level* 的组件是否具有优先级属性,优先级高的 *depth* 小;若没有优先级,则按照组件加入到模型中顺序进行排序,加入早的组件,其 *depth* 小,由此按照 *level* 将组件排成一个队列,进而网络中的各个时间自动机也是有序的.*depth* 小的时间自动机执行完,*depth* 大的时间自动机才可以执行.

3.3.2 有限状态机组件

Ptolemy DE 模型中的组件可以由有限状态机(finite state machine,简称 *FSM*)来定义,*FSM* 常用于系统的行为建模,系统的状态是指它在特点时间点所处的状况.在 *FSM* 中,系统行为是由状态集以及管理这些状态转移的规则共同建模而成的.Ptolemy 模型中的有限状态机是一个六元组 $FSM=(S,s_0,V,I,O,R)$,其中,

- S:有限的状态集合;
- s_0:初始状态;
- V:有限的参数集合;
- I:输入接口集合;
- O:输出接口集合;
- R: $s_i\xrightarrow{g,o,u}s_j(s_i\in S,s_j\in S)$ 是一组状态迁移,其中,g 是迁移是否满足的约束条件,o 是对输出接口的赋值操作,u 是参数的更新操作.

有限状态机的转换包括其参数、输入接口、输出接口及内部状态迁移的转换:

$$Trans\_FSM(\cdot)\cong Trans\_Paras(V)\cap Trans\_InputPs(I)\cap Trans\_OutputPs(O)\cap Trans\_RFS(R)$$

$$Trans\_RFS(R)\cong Trans\_RF(R[1])\cap\ldots\cap Trans\_RF(R[n])$$

$$Trans\_RF(R[i])\cong Trans\_RFT(R[i])\cup Trans\_RFF(R[i])$$

$$Trans\_RFT(R[i])\cong Edge_1(l,g,\phi,r,lc)\cap Edges_2(lc,g,O_j!,r,lc)\cap Edge_3(lc,g,\phi,r,l')$$

$$Trans\_RFF(R[i])\cong Edges_1(l,\phi,I_j?,r,lu)\cap Edges_2(lu,\phi,I_j?,r,lu)\cap Edge_3(lu,g,\phi,\phi,lg)\cap$$
$$Edge_4(lg,g,\phi,r,lc)\cap Edge_5(\lg,!g,\phi,r,l)\cap Edges_6(lc,g,O_j!,r,lc)\cap Edge_7(lc,g,\phi,r,l')$$

有限状态机中的参数、输入接口和输出接口转换成一组变量,有限自动机中的迁移转换成时间自动机中的迁移,有限时间自动机迁移中的输出接口赋值操作和参数更新操作转换为时间自动机迁移中的 r 操作.若迁移 $R[i]$的约束条件是“true”,则执行 $Trans\_RFT(R[i])$,其中,l 是 urgent 状态,lc 是 committed 状态.迁移 $Edge_1$ 用于当满足 *depth* 时,为输出接口赋值和更新参数;$Edge_2$ 是一组为赋值的输出接口产生同步信号的迁移;$Edge_3$ 使时间自动机更新到目标状态.注意:当组件的输出接口连接的是仅用于展示结果的组件时,其赋值的输出接口不再产生同步信号.其他迁移则执行 $Trans\_RFF(R[i])$,其中,lu 是 urgent 状态,lg,lc 是 committed 状态.$Edges_1$ 和 $Edges_2$ 是两组接收同步信号的迁移;$Edge_3$ 用于时间自动机满足 *depth* 时,更新到判断约束条件是否为真的状态.若约束条件为真,则执行 $Edge_4$ 为输出接口赋值并更新参数;若约束条件为假,则执行 $Edge_5$ 回到迁移的源状态.$Edge_6$ 是一组为赋值的输出接口产生同步信号的迁移,$Edge_7$ 使时间自动机更新到目标状态.

3.3.3 模态模型组件

模态模型是层次化的组件,其顶层是 *FSM*,只是状态可以添加细化模型,从而实现分层.在转换模态模型时保留下层次化的特点,从而防止了摊平组件时可能造成的状态空间的爆炸:

$$Trans\_RFT(R[i])\cong Edge_1(l,g,down!,r,ld)\cap Edge_2(ld,\phi,up?,r,lc)\cap Edges_3(lc,g,O_j!,r,lc)\cap Edge_4(lc,g,\phi,r,l')$$

$$Trans\_RFF(R[i])\cong Edges_1(l,\phi,I_j?,r,lu)\cap Edges_2(lu,\phi,I_j?,r,lu)\cap Edge_3(lu,g,down!,r,ld)\cap$$
$$Edge_4(ld,\phi,up?,\phi,lr)\cap Edge_5(lr,g,\phi,r,lc)\cap Edge_6(lr,!g,\phi,r,lro)\cap$$
$$Edges_7(lc,g,O_j!,r,lc)\cap Edge_8(lc,g,\phi,r,l')\cap Edges_9(lro,g,O_j!,r,lro)\cap$$
$$Edge_{10}(lro,!g,\phi,r,l)\cap Edge_{11}(lro,g,\phi,r,lu)$$

细化模型有两种,分别是组件形式的细化和模态模型形式的细化:组件形式的细化是指细化模型中可以任意添加组件,而模态模型形式的细化是指其细化模型中是状态之间的迁移.在模态模型的转换过程中,保留了层次化的特点,只是在相邻两层之间建立两个同步通道:上层发送同步,使得下层时间自动机触发;下层的各个时间自动机执行完之后,发送同步,使得上层时间自动机继续执行.

$Trans\_RFT(R[i])$,其中,l 是 urgent 状态,ld 和 lc 是 committed 状态.迁移 $Edge_1$ 用于当满足 *depth* 时,向下层发送同步信号;$Edge_2$ 则是接收下层发送的同步信号,同时对输出接口赋值和更新参数;$Edges_3$ 是一组为赋值的输出接口产生同步信号的迁移;$Edge_4$ 使时间自动机更新到目标状态,若目标状态包含细化模型,则更新到目标状态的迁移会发送初始化细化模型的广播信号 *init*!.

$Trans\_RFF(R[i])$,其中,lu 是 urgent 状态,ld,lr,lc 和 lro 是 committed 状态.$Edges_1$ 和 $Edges_2$ 是两组接收同步信号的迁移;$Edge_3$ 用于时间自动机满足 *depth* 时,向下层发送同步信号;$Edge_4$ 接收下层发送的同步信号,并更新到判断约束条件是否为真的状态.若约束条件为真,则执行 $Edge_5$ 为输出接口赋值并更新参数;若约束条件为假,则执行 $Edge_6$ 更新到准备产生同步信号的状态.$Edges_7$ 和 $Edges_9$ 是两组为赋值的输出接口产生同步信号的迁移;$Edge_8$ 使时间自动机更新到目标状态;$Edge_{10}$ 表示当细化模型不可以继续执行时,模态模型本次迭代结束,回到迁移的源状态;$Edge_{11}$ 表示细化模型可以继续执行时,时间自动机更新到可以向下层发送同步信号的状态:

$$Trans\_RFSM(R[i])\cong Edge_1(l,\phi,rev?,\phi,lc)\cap Edge_2(lc,g,sen!,r,l')\cap$$
$$Edge_3(lc,!g,sen!,\phi,l)\cap Edge_4(l,\phi,init?,\phi,linit)\cap$$
$$Trans\_RCOM(E)\cong Trans\_NActor(E[1])\cap Trans\_Actor(E[2])\cap\ldots\cap$$
$$Trans\_Actor(E[x-1])\cap Trans\_NActor(E[x])$$

$$Trans\_NActor(E[1]) \cong Edge_1(l,\phi,rev?,\phi,lc) \cap Edge_2(lc,g,\phi,\phi,l') \cap Trans\_Actor(E[1]) \cap Edge_3(lc,!g,sen!,\phi,l)$$

$$Trans\_NActor(E[x]) \cong Trans\_Actor(E[x]) \cap Edge(lc,\phi,sen!,\phi,li)$$

Trans_RFSM(*R*[*i*])表示模态模型形式的细化模型中迁移的转换规则,*lc* 是 committed 状态,*linit* 是细化模型的初始状态.$Edge_1$用于接收前一个细化模型发送的同步,若该细化模型是状态下的第 1 个细化模型,则此迁移用于接收 *down*?;$Edge_2$ 表示当满足约束条件时,对输出接口赋值并更新参数,同时发送同步信号,若该细化模型是状态下的最后一个细化模型,则此迁移用于发送 *up*!;$Edge_3$ 表示当约束条件不满足时,只发送同步信号;$Edge_4$ 表示接收到初始化细化模型的广播,时间自动机回归到初始状态.

Trans_RCOM(*E*)表示组件形式的细化模型中迁移的转换规则,细化模型中 *depth* 最小和最大的组件,需要添加接收或发送同步信号的迁移,其他组件按组件映射规则转换即可.*Trans_NActor*(*E*[1])用于转换 *depth* 最小的组件.$Edge_1$ 用于接收前一个细化模型发送的同步,然后立即判断组件需要的模态模型输入接口信息是否满足:若满足$Edge_2$,更新到*E*[1]对应时间自动机结构的迁移状态,并发送同步信号;若不满足$Edge_3$,则回到接收同步信号的状态,并发送同步信号.*Trans_NActor*(*E*[*x*])用于转换细化模型中 *depth* 最大的组件,当组件对应时间自动机执行完,最后加一条 *Edge* 用于发送同步信号.

3.4 一致性保证

基于模型转换的形式化验证方法,保证转换前后模型的一致性是至关重要的.一般情况下,如果两个模型的语义保证下列行为是一致的,则说明两个模型是等价的[9].(1) 当输入相同时,两个模型有相同的执行路径;(2) 当输入相同时,两个模型对应的参数是相同的.

在 Ptolemy DE 模型中,组件的触发顺序是严格按照事件的时间戳进行的,当两个事件的时间戳一样时,则继续判断两个组件的优先级:若优先级也一样,则按照加入模型的顺序进行触发.Ptolemy DE 模型中组件的触发总是满足该原则:向组件 *B* 提供输入的组件 *A* 会先执行.

根据第 3.1 节数据类型的映射规则,因为时间自动机中支持 int 和 bool 类型,所以可以直接将 Ptolemy 中 int 和 bool 类型的数据转换为时间自动机中 int 和 bool 类型的数据;为了保证时间自动机必然可以在某一时刻发生迁移,为其设置了一个时钟变量,并添加位置上的 invariant 和迁移上的 guard 约束;在 Ptolemy 模型中,事件是从一个组件发送给另一个组件的瞬时消息,即从前一个组件产生输出消息,到后一个组件接收到该消息,并触发组件是瞬时完成的,逻辑时间没有推进.

根据第 3.2 节关系的映射规则,保证了时间自动机为输出接口参数赋值和后继组件转换生成的时间自动机为输入接口参数赋值是同时发生的,并且转换而来的时间自动机与组件功能一致,保证了当输入相同时,时间自动机对参数的更新操作与组件对参数的更新操作是相同的.

根据第 3.3.1 节特定组件的映射规则,将每个组件转换成功能相同的时间自动机,同时,每个时间自动机都有一个 *depth* 属性,各个时间自动机的 *depth* 值不同,时间自动机网络按照 *depth* 从小到大,判断时间自动机是否可以执行,保证了时间自动机始终遵循着 Ptolemy 中“向组件 *B* 提供输入的组件 *A* 会被先执行”这一原则,从而保证了当输入相同时,时间自动机的执行顺序和 Ptolemy 中组件的执行顺序是严格一致的.

根据第 3.3.2 节有限状态机的映射规则,将组件接收和发送事件,转换为时间自动机中的接收同步信号和发送同步信号,有限状态机中的一次状态迁移转换为时间自动机中的一组位置迁移.

第 3.3.3 节指出了层次化模型中,上层组件和下层组件转换生成的时间自动机之间通过交互通道通信,保证了各层之间信息的传递.

4 Ptolemy DE 模型验证框架及实现

4.1 Ptolemy DE模型验证框架

本文提出将 Ptolemy DE 模型转换为可以在 Uppaal 中进行形式化验证的时间自动机模型,从而进行验证,

其整体框架如图 3 所示.验证 Ptolemy DE 模型分为转换模型和形式化验证两部分.

- 在模型的自动转换阶段,将 Ptolemy DE 模型和根据模型提取的属性验证公式作为输入,首先对 Ptolemy DE 模型进行解析,获得各个组件的结构信息;然后根据定义的映射规则,将各个组件转换为对应的时间自动机;最终,将时间自动机与属性验证公式共同构成一个可进行验证的时间自动机网络结构;
- 将内存中的时间自动机网络结构输出文件,即可调用 Uppaal 验证引擎进行验证.

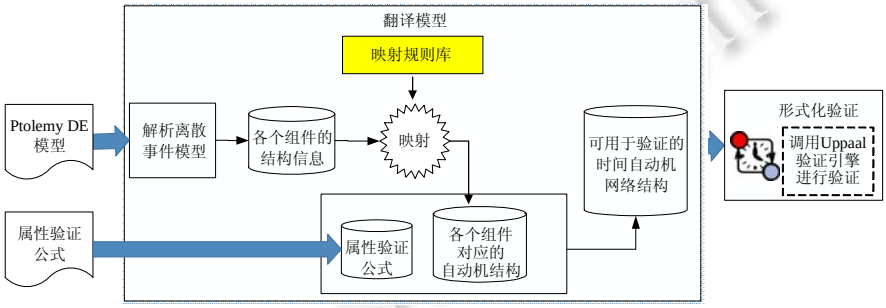


Fig.3 Framework of verifying Ptolemy DE model

图 3 验证 Ptolemy DE 模型的整体框架

4.2 Ptolemy DE模型验证实现

算法 2 是 Ptolemy DE 模型自动化转换的伪代码,以 Ptolemy DE 模型和属性验证公式为输入/输出一个时间自动机网络结构.第 4 行获取 Ptolemy DE 模型中的组件.第 5 行~第 8 行将各个组件转换为时间自动机,并保存到 beans 中.第 6 行 *getAByRule*(*A*)是组件转换的核心函数,其过程是:首先,根据组件类型读取模板库中对应的模板文件;然后,通过获取组件 *A* 中的参数、接口,生成对应的声明字符串;进而对获取的模板文件中的字符串进行替换,从而实现根据模板文件生成一个完整的时间自动机.第 9 行表示将各个时间自动机和属性验证公式一起构成可用于形式化验证的自动机网络结构.

算法 2. Translation of Ptolemy Discrete Event Model.

1: Input: *TP* model, *Formula*;
2: Output: Network of automata.
3: *bean*←∅
4: *actorArr*←*parse*(*TP* model)
5: **for** each *actor A* in *actorArr* **do**
6: *bean*←*getAByRule*(*A*)
7: *bean.add*(*bean*)
8: **end for**
9: *Network*←*getCode*(*beans*,*Formula*)
10: **return** *Network*

图 4 是调用插件的界面,在 Output Type 中选择输出可进行形式化验证的时间自动机文件或者直接进行验证,在 Temporal Formula 中输入属性验证公式,在 Terminal 中返回验证结果.

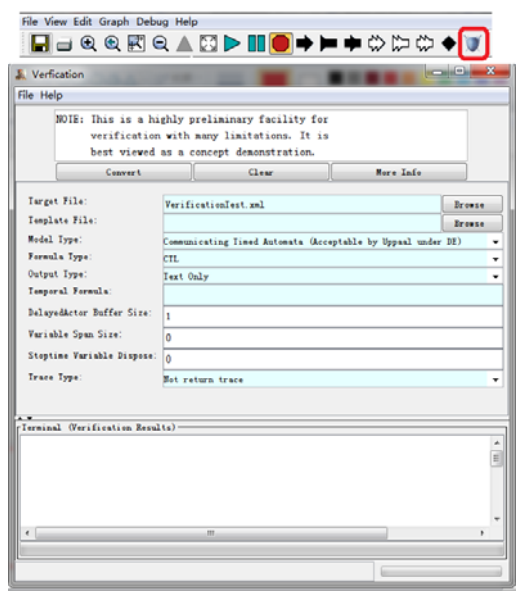


Fig.4 Interface of verifying Ptolemy DE model

图 4 调用自动化转换插件及验证引擎的界面

5 案例分析

5.1 情景描述

如图 5 是一个交通信号灯系统,指示车辆行动的信号灯有红、黄、绿这 3 种,指示行人行动的指示灯有红、绿这两种.当系统正常工作时:

汽车指示灯按照红、黄、绿、黄的顺序变换,分别停留 3,1,2,1 个时间单元;

行人指示灯根据汽车指示灯的变换而变换:当汽车指示灯由红变黄时,行人指示灯变红;当汽车指示灯由黄变红时,行人指示灯变绿.

有时系统会出现故障,此时只有指示汽车暂停行驶的黄灯闪烁,其他指示灯都灭.特殊情况包括:(1) 车辆和行人不能同时行动,即两种指示灯不能同时为绿色;(2) 当信号灯系统出现故障时,只有黄灯闪烁,其他灯不亮.

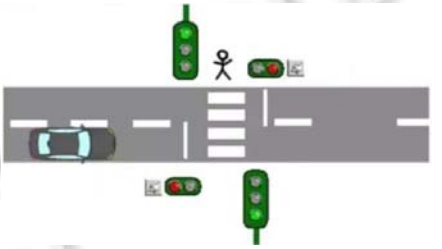


Fig.5 Traffic environment

图 5 交通灯系统实体图

5.2 Ptolemy DE模型

如图 6,Ptolemy DE 模型中定义了 5 个参数:*Pred*,*Pgrn*,*Cred*,*Cyel* 和 *Cgrn*,分别表示不同的交通灯:当值为 1 时,表示其对应的交通灯亮;值为 0 时,表示其对应的交通灯灭.如图中参数 *Pred* 和 *Cyel* 值为 1,则行人红灯和车辆黄灯亮.DiscreteClock 是一个事件源组件,以 1 为周期产生输出信号,Decision 是一个异常生成器,TrafficLight 是交通灯控制器,SetVariable 组件用于更新参数值.

组件 Decision 是一个有限状态机,功能是模拟系统中产生异常,其中包含 Normal,Abnormal 两个状态和一个 number 参数.Decision 在状态 Normal 上停留 15 个时间单元,每个时间单元,Ok 接口都会产生输出表示系统正常;之后,Error 接口产生输出表示系统出现异常,Decision 更新到 Abnormal 状态.在状态 Abnormal 上停留 5 个时间单元,每个时间单元,Error 接口都会产生输出表示系统出现异常;之后,Ok 接口产生输出表示系统恢复正常,Decision 更新到 Normal 状态.Decision 负责维持系统 15 个时间单元正常、5 个时间单元异常,依次交替更新.

组件 TrafficLight 交通灯控制器是一个模态模型,有两个状态 normal 和 error,状态 normal 和 error 各有一个细化模型.TrafficLight 处于状态 normal 时表示系统正常,可以正常控制交通灯;处于状态 error 时表示系统出现异常,控制交通灯显示异常情况,只有黄灯闪烁,其他灯都不亮.TrafficLight 处于 normal 状态,当输入接口 Error 有输入信号时,表示系统出现异常,TrafficLight 更新到 error 状态,同时初始化 error 状态中的细化模型;TrafficLight 处于 error 状态,当输入接口 Ok 有输入信号时,表示系统恢复正常,TrafficLight 更新到 normal 状态,同时初始化 normal 状态中的细化模型.

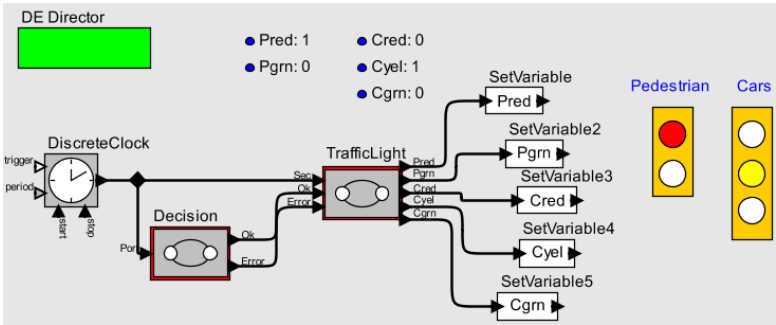


Fig.6 Ptolemy DE model of traffic environment

图 6 交通灯 Ptolemy DE 模型

如图 7,TrafficLight normal 是 TrafficLight 中 *normal* 状态的一个细化模型,表示系统在正常情况下,交通灯控制器的详细模型,包含两个控制器:车辆交通灯控制器 CarLightNormal 和行人交通灯控制器 PedestrianLightNormal.当 CarLightNormal 的输出接口 Pgo 和 Pstop 产生输出信号时,PedestrianLightNormal 会立即作出相应,确保车辆的行为发生改变时,行人的行为也立即发生改变.

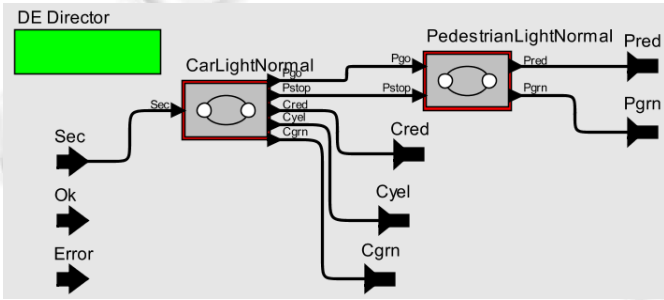


Fig.7 Refinement of the normal mode in TrafficLight

图 7 TrafficLight normal 细化模型

如图 8,CarLightNormal 是一个有限状态机,是车辆交通灯的控制器.控制器首先让输出接口 *Cred* 输出 1,*Cyel*,*Cgrn* 输出 0,是为了初始化车辆交通灯只有红灯亮.CarLightNormal 在 *Cred* 状态停留 3 个时间单元,使得红灯持续亮了 3 个时间单元,之后输出接口 *Cyel* 产生输出,修改参数 *Cyel* 的值为 1,使得黄灯亮;输出接口 *Pstop* 产生输出,使得行人交通灯控制器处于 *Pred* 状态;CarLightNormal 更新到 *Credyel* 状态.紧接着,下一个时刻,CarLightNormal 从 *Credyel* 状态更新到 *Cgrn* 状态,并让输出接口 *Cred*,*Cyel*,*Cgrn* 产生 0,0,1 的输出,使得车辆交通灯只有绿灯亮.CarLightNormal 在 *Cgrn* 状态停留两个时刻,更新到 *Cyel* 状态,使得黄灯亮、绿灯灭.下一时刻,CarLightNormal 由 *Cyel* 状态更新到 *Cred* 状态,并使得车辆交通灯只有红灯亮了,同时输出接口 *Pgo* 产生输出,通知行人交通灯控制器要更新到 *Pgrn* 状态,表示行人可以行动了.如此重复这个过程,使得车辆交通灯在红、黄、绿、黄间不断变化.

如图 9,PedestrianLightNormal 是一个有限状态机,是行人交通灯的控制器.控制器首先让输出接口 *Pred* 输出 1,*Pgrn* 输出 0,是为了初始化行人交通灯只有红灯亮.当 PedestrianLightNorma 的输入接口 *Pgo* 接收到信号时,表示行人可以行动了,因此控制器跳转到 *Pgrn* 状态,并且输出接口 *Pred*,*Pgrn* 分别输出 0,1,使得信号灯变为绿灯亮、红灯灭;当输入接口 *Pstop* 接收到信号时,表示行人不可以行动了,因此控制器跳转到 *Pred* 状态,并且输出接口 *Pred*,*Pgrn* 分别输出 1,0,使得信号灯变为红灯亮、绿灯灭.

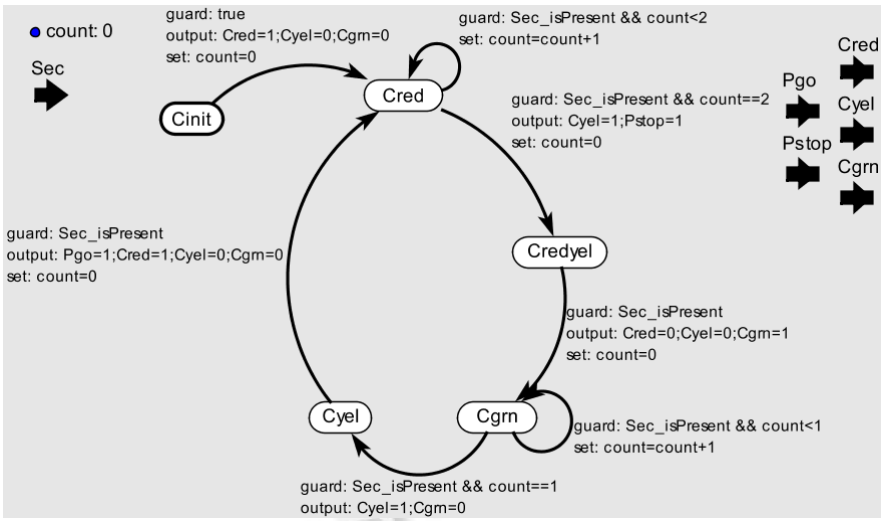


Fig.8 FSMActor CarLightNormal defining the operation of the car light controller in normal mode

图 8 CarLightNormal 车辆交通灯控制器

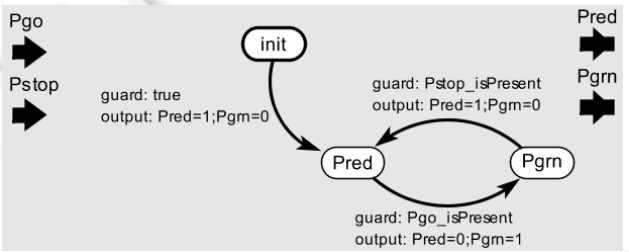


Fig.9 FSMActor PedestrianLightNormal defining the operation of the pedestrian light controller in normal mode

图 9 PedestrianLightNormal 行人交通灯控制器

如图 10,TrafficLight error 是 TrafficLight 中 error 状态的一个细化模型,表示系统在异常情况下,交通灯控制器的详细模型.当 TrafficLight 处于 error 状态时,表示系统出现异常,其细化模型在 YellowOn 状态和 YellowOff 状态之间,以 1 个时间单元为周期来回跳转,输出接口 *Cyel* 交替输出 1,0,导致参数 *Cyel* 的值不断更新为 1,0,产生黄灯闪烁的效果.

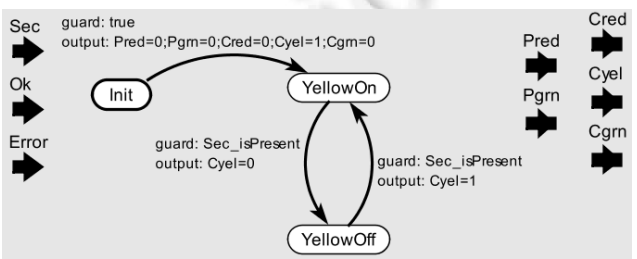


Fig.10 Refinement of error mode in TrafficLight

图 10 TrafficLight error 细化模型

5.3 核心时间自动机

图 11 是由有限状态机 CarLightNormal 转换而来并进行优化的时间自动机,输出接口 *Cred*,*Cyel* 和 *Cgrn* 连接的组件与模型的整体逻辑无关,所以这些输出接口不再产生同步信号.输入接口 *Sec* 直接与上层 *Sec* 相连,共

用一组声明即可,所以只针对输出接口 *Pgo*,*Pstop* 和自定义参数 *count* 产生相应的全局声明.组件中的每条迁移都至少对应时间自动机中的一条迁移,首先 *TrafficLight_normal_down*?接收顶层传来的同步,然后 *TrafficLight_Sec_isPresent* 判断输入接口信息是否满足,再判断迁移的约束条件是否满足:若满足,则对输出接口进行赋值并更新参数,为赋值的输出接口产生同步信号,更新到目的状态;若不满足,则返回迁移的源状态.时间自动机中除初始状态外,每个状态都对应有一条以初始状态为目标状态的迁移 *TrafficLight_Normal_init*?,用于初始化时间自动机.

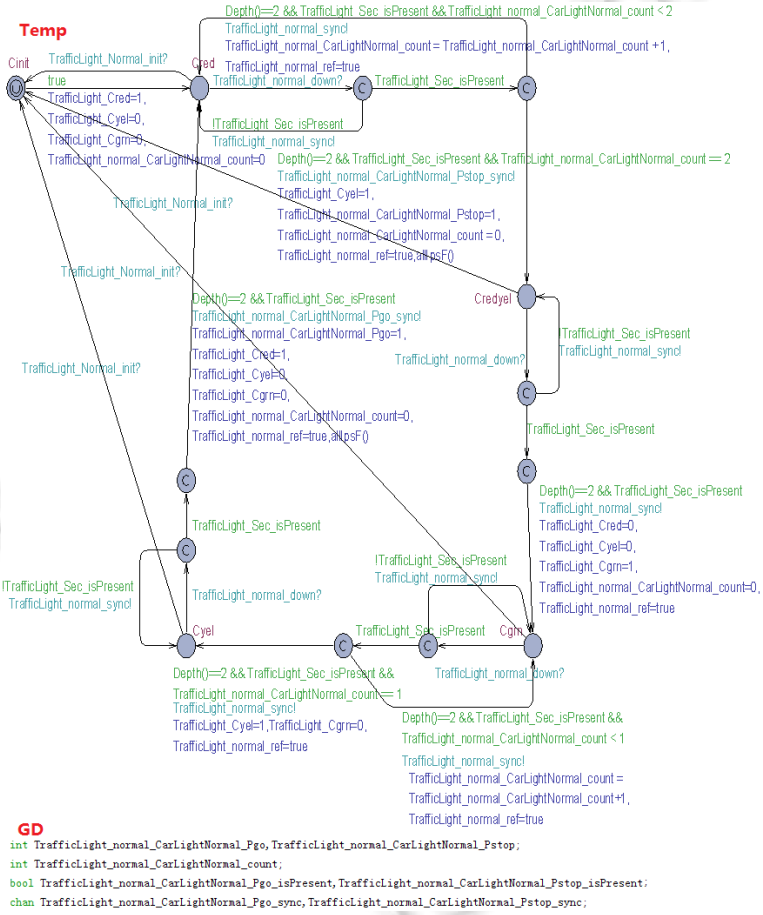


Fig.11 Automaton of CarLightNormal

图 11 CarLightNormal 时间自动机

图 12 是由有限状态机 PedestrianLightNormal 转换而来并进行优化的时间自动机,输出接口 *Pred* 和 *Pgrn* 连接的组件与模型的整体逻辑无关,所以不再为其产生同步信号,只为每个输入接口产生 int 和 bool 的全局声明.每组迁移首先 *TrafficLight_normal_down*?接收顶层传来的同步,然后判断迁移的约束条件是否满足:若满足,则对输出接口进行赋值并修改参数,更新到目的状态,同时产生同步信号 *TrafficLight*_normal_*sync*!与顶层同步;若不满足,则返回迁移的源状态,同时产生同步信号 *TrafficLight_normal_sync*!与顶层同步.时间自动机中除初始状态外,每个状态也都对应有一条以初始状态为目标状态的迁移 *TrafficLight_Normal_init*?,用于初始化时间自动机.

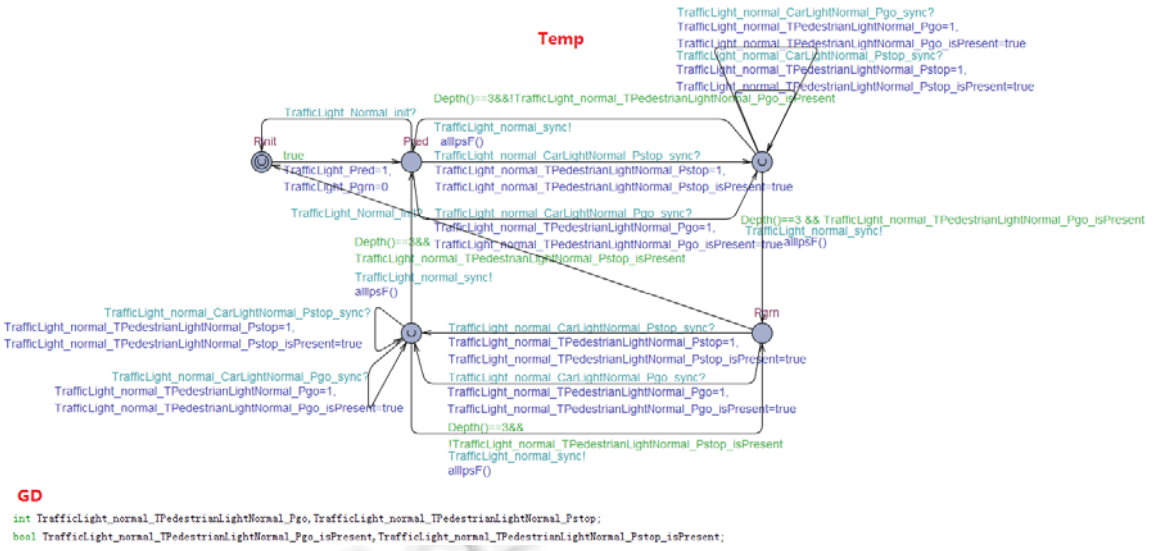


Fig.12 Automaton of PedestrianLightNormal

图 12 PedestrianLightNormal 时间自动机

图 13 是由 TrafficLight error 细化模型转换而来并进行优化的时间自动机,不必为输出接口产生同步信号.每组迁移首先 *TrafficLight_error_down*?接收顶层传来的同步,然后判断迁移的约束条件是否满足:若满足,则对输出接口进行赋值并修改参数,更新到目的状态,同时产生同步信号 *TrafficLight_error_sync*!与顶层同步;若不满足,则返回迁移的源状态,同时产生同步信号 *TrafficLight_error_sync*!与顶层同步.时间自动机中除初始状态外,每个状态也都对应有一条以初始状态为目标状态的迁移 *TrafficLight_error_init*?,用于初始化时间自动机.

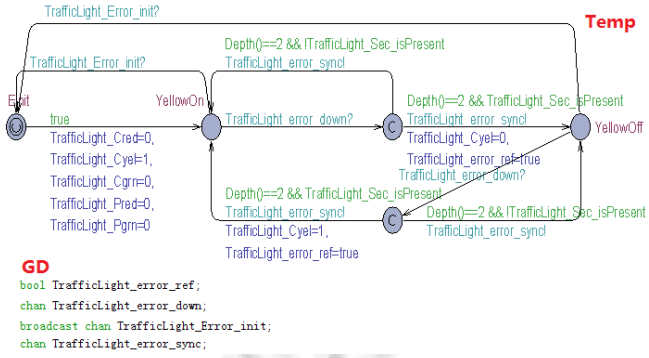


Fig.13 Automaton of refinement of TrafficLight error

图 13 TrafficLight error 细化模型的时间自动机

5.4 属性验证

在此基础上,通过提取 Ptolemy DE 模型中的属性特征,形式化描述为 CTL 公式,进而使用 Uppaal 验证的引擎对模型的可靠性、安全性、死锁进行形式化验证.

(1) *A*[·]!(*CarLightNormal.Cgrn* && *PedestrianLightNormal.Pgrn*)

Cgrn 和 *Pgrn* 分别是车辆交通灯控制器 CarLightNormal 和行人交通灯控制器 PedestrianLightNormal 中表示对应绿灯亮的状态.如果该属性满足,则证明系统不存在车辆交通灯和行人交通灯同时绿灯亮的情况.

(2) *A*[·]!(*Pgrn*==1 and *Cgrn*==1)

Pgrn 和 *Cgrn* 分别是定义的表示行人交通灯绿灯和车辆交通灯绿灯亮灭的参数.当该属性满足时,证明系统不存在车辆交通灯和行人交通灯同时为绿的情况.

(3) *A*[·](*Decision.AbNormal imply*!*TrafficLight.normal*)

AbNormal 是异常生成器 Decision 中表示出现异常的状态,当系统出现异常时,交通灯控制器 TrafficLight 不能处于 normal 状态.如果该属性满足,则证明当系统出现异常时,交通灯控制器 TrafficLight 处于异常状态.

(4) *A*[·](*Decision.AbNormal imply*(*Pgrn*==0 and *Pred*==0 and *Cgrn*==0 and *Cred*==0 and (*Cyel*==0 or *Cyel*==1)))

Decision 处于 AbNormal 状态时,表示系统出现异常,此时,*Pgrn*,*Pred*,*Cgrn* 和 *Cred* 的值都为 0,*Cyel* 的值在 0 和 1 之间交替变换.如果该属性满足,则证明当系统出现异常时,只有指示车辆暂停行动的黄灯闪烁,其他灯灭.

(5) *A*[·] not deadlock

该属性验证公式用于验证系统中是否存在死锁.

表 1 是上述验证公式详细的验证情况,当时间自动机模型中有不满足的属性时,Uppaal 验证引擎可以返回异常路径.

Table 1 Details of verifying formulas

表 1 属性公式验证消耗

验证公式	验证费时(s)	Kernel 费时(s)	总费时(s)	常驻内存(KB)	虚拟内存的使用峰值(KB)	验证结果
(1)	0	0	0.003	6 804	26 228	满足
(2)	0	0	0.004	6 892	26 216	满足
(3)	0	0	0.042	6 930	271 251	满足
(4)	0	0	0.045	7 012	280 134	满足
(5)	81.62	0.218	82.751	137 340	284 156	满足

6 相关工作

在模型驱动开发过程中,模型从各个角度对系统进行了描述[10],是早期对可靠性进行分析的一个重要环节.然而,工业界建模工具中的模型通常缺乏形式语义,无法进行形式验证,所以研究人员采用模型转换的思想[11,12],在保证语义一致的基础上,将构建好的模型转换为可以进行验证的形式化模型,如将 UML,AADL,Yakindu statecharts,POOSL,TASM 或者 Simulink 构建的模型转换为一种形式化模型,进而实现验证.北京大学的研究团队提出了 UML 到 Markov 链的语义转换,通过在 UML 的用况图、活动图、顺序图、构件图中加入可靠性相关的信息,构造出整个系统的构件转移图;根据系统的构件转移图,并结合构件图中描述的单个构件的可靠性,最终构造出用于可靠性分析的 Markov 链[13].空中客车公司的 TOPCASED 项目提出了 AADL[14]到中间语言 Fiacre 的语义转换,主要给出了线程执行和通信的转换原理,所关注的研究子集和转换规则采用自然语言进行描述[15].法国 IRIT 实验室采用动作时序逻辑描述语言 TLA+对 AADL 执行模型的部分语义做了初步研究,包括端口通信、共享变量的构件间通信、抢占式调度策略以及模式的语义.这是 AADL 语义形式化的最早研究工作,其转换规则主要以语义函数的方式给出,但 TLA+的模型检测工具 TLC 的验证能力较弱[16].伊利诺斯理工大学的研究团队提出了 Yakindu statecharts 到时间自动机模型的相关转换、验证工作[9],并给出了判断转换前后模型一致性的条件以及验证属性不满足时追溯原模型中路径的方法;该团队还提出了将 Stateflow 模型转换为时间自动机模型,进而使用 Uppaal 进行验证的方法[17].特文特大学的研究团队提出了一种将简化了的运动控制系统的 POOSL 模型转换为时间自动机模型的方法[8],并验证了其功能行为和最坏情况下的延迟.北京航空航天大学的研究团队给出了 TASM 到时间自动机模型地转换语义,并开发了一个自动转换工具[18],通过 atlas 转换语言,将 TASM 转换为时间自动机模型,进而进行相关的形式化验证工作.以色列的某研究团队设计了一个转换验证工具 TVS,将 Simulink 模型转换成优化的 C 代码[19],并重点介绍各种技术问题,比如特征不变量的自动生成.中国科学院软件研究所的研究团队提出了将 Simulink/Stateflow 图形模型转换成 HCSP 形式模型的方法,并使用 HHL 及其定理证明器形式化验证了转换后的形式模型[20].清华大学的研究团队提供了使用时间自动机描述和验证多时钟系统的方法[21,22].

现阶段,关于形式化验证 Ptolemy DE 模型的研究工作有两种,都是根据构建好的 Ptolemy DE 模型,生成另一种可用于形式化验证结构的文件,进而利用相关验证工具实现验证 Ptolemy DE 模型的形式化验证[23].

(1) 根据构建的Ptolemy DE模型生成可以使用RED(regional encoding diagram)进行验证的时间自动机结构.该方法将 Ptolemy DE 模型描述成一个通信时间自动机,通过 RED 工具进行验证.该方法支持的组件有 Clock,FSMActor,ModalModel,TimedDelay.这种方法支持的组件数量少、功能单一,ModalModel 仅支持状态包含一个 FSM 细化模型,在转换时采用摊平的方式展开,易造成状态空间爆炸;

(2) 使用 Real-Time Maude 语言形式化 Ptolemy DE 模型,同时,借助 Ptolemy 自身的代码生成机制,生成可供 Real-Time Maude 工具验证的文件.Real-Time Maude 是一种支持对实时系统和混合系统进行验证的工具,该工具读取的文件需使用 Real-Time Maude 语言、语法编写.Real-Time Maude 基于重写逻辑进行验证,适合于面向对象实时系统的验证.该方法支持的组件有 CompositeActor,FSMActor, ModalModel,TimedDelay,Clock,CurrentTime,Pulse,Ramp,TimedPlotter,SetVariable 和 SingleEvent.这种方式不支持延迟组件处理多个输入事件,并且事件源组件采用 tick 模拟时间的增加,增加了验证的复杂度;同时,该方法是利用 Maude 语言形式化组件,进而进行转换,不利于组件库的扩展.

7 总结与讨论

在系统开发前期,如何对模型进行形式化验证、保证系统的可靠性,是目前学术界和工业界共同关注的热点问题.本文提出一种基于模型转换的形式化方法来验证 Ptolemy DE 模型.Ptolemy DE 模型根据不同事件的时间戳触发组件,时间自动机模型能够表达这个特征,因此,我们选用 Uppaal 作为验证工具.我们首先设:散事件模型转换为时间自动机模型,进而通过调用Uppaal验证内核完成验证.与现有研究工作相比,该方法定义了更为完整的 Ptolemy 组件转换规则,可以支持更多的组件;保留了模型层次化的特点,有效避免了状态空间爆炸;避免使用 tick 计时,降低验证的复杂度.现阶段,该方法支持组件见表 2.

Table 2 Actors that can be translated

表 2 本文方法支持的组件

组件名称	组件名称	组件名称	组件名称	组件名称
CompositeActor	FSMActor	ModalModel	TimeDelay	DiscreteClock
Const	Clock	Ramp	TimedDelay	TimedPlotter
BooleanSwitch	SingleEvent	Pulse	Scale	AddSubtract
RecordAssembler	RecordDisassembler	Expression	Limiter	Maximum
BooleanSelect	Multiplexor	Minimum	SetVariable	Sampler

对于基于模型转换的形式化验证方法,如何验证模型转换的一致性是一个重要问题.在现有的 Ptolemy 模型转换研究工作中,都是采用手工确认的方式,未来工作:

(1) 设计一种方法自动化证明 Ptolemy DE 模型与目标模型的语义一致性;

(2) Ptolemy 提供了丰富的组件库,需要继续丰富映射规则,从而支持更多的组件;

(3) Ptolemy 模型支持复杂数据类型(double、矩阵等)和复杂计算操作(三角函数、对数函数等),需要继续设计这些复杂数据类型及复杂计算操作的映射规则.

References:

[1] Joseph B, Soonho H, Edward L, David M. Ptolemy: A framework for simulating and prototyping heterogeneous systems. Int'l Journal of Computer Simulation, 1995,10.

[2] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019,30(1):33−61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[3] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(z1):1907−1912 (in Chinese with English abstract).

[4] Mudit G, Kienhuis B, Edward L, Liu J. Ptolemy II—Heterogeneous Concurrent Modeling and Design in Java. 1999.

[5] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235.

[6] Behrmann G, David A, Larsen K. A tutorial on uppaal. In: Proc. of the Formal Methods for the Design of Real-Time Systems. Springer, 2004. 200−236.

[7] Bengtsson J, Yi W. Timed automata: Semantics, algorithms and tools. LNCS, 2003,87−124.

[8] Xing JS, Theelen BD, Langerak R, Van JDP, Tretmans J, Voeten JPM. From POOSL to UPPAAL: Transformation and quantitative analysis. In: Proc. of the 2010 10th Int'l Conf. on Application of Concurrency to System Design. 2010. 47−56. [doi: 10.1109/ACSD.2010.21]

[9] Guo C, Ren SP, Jiang Y, Wu PL, Sha L, Richard B, Berlin J. Transforming medical best practice guidelines to executable and verifiable statechart models. In: Proc. of the ACM/IEEE Int'l Conf. on Cyber-Physical Systems. ACM, 2016.

[10] Jiang Y, Liu H, Song H, Kong H, Wang R, Guan Y, Liu S. Safety-Assured model-driven design of the multifunction vehicle bus controller. IEEE Trans. on Intelligent Transportation Systems, 2018,19(10).

[11] Csertan G, Huszerl G, Majzik I, Pap Z, Andras P. VIATRA—Visual automated transformations for formal verification and validation of UML models. In: Proc. of the Automated Software Engineering. 2002. 267−270. [doi: 10.1109/ASE.2002.1115027]

[12] Jordi C, Robert C, Daniel R. UMLtoCSP: A tool for the formal verification of UML/OCL models using constraint programming. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2007). New York: Association for Computing Machinery, 2007. 547−548. [doi: https://doi.org/10.1145/1321631.1321737]

[13] Liu Y, Ma ZY, He X, Shao WZ. Approach to transforming UML model to reliability analysis model. Ruan Jian Xue Bao/Journal of Software, 2010,21(2):287−304 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3792.htm [doi: 10.3724/SP.J.1001.2010.03792]

[14] Yang ZB, Hu K, Zhao YW, Ma DF, Bodeveix JP. Verification of AADL models with timed abstract state machines. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):202−222 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4776.htm [doi: 10.13328/j.cnki.jos.004776]

[15] Berthomieu B, Bodeveix JP, Chaudet C, Zilio SD, Filali M, Vernadat F. Formal verification of AADL specifications in the topcased environment. In: Proc. of the 14th Ada-Europe Int'l Conf. on Reliable Software Technologies. Berlin, Heidelberg: Springer-Verlag, 2009. 207−221. [doi: 10.1007/978-3-642-01924-1_15]

[16] Liang ML, Wang XP, Xue XP, Li G. Formal verification of UML models based on TLA. Computer Engineering, 2011,37(2):72−74.

[17] Jiang Y, Song HB, Yang YX, Liu H, Gu M, Guan Y, Sun JG, Sha L. Dependable model-driven development of CPS: From stateflow simulation to verified implementation. ACM Trans. on Cyber-Physical Systems, 2019,3(1):12:1−12:31.

[18] Hu K, Zhang T, Yang ZB, Gu B, Jiang S, Jiang BC. Formal verification of TASM models by translating into UPPAAL. Journal of Donghua University (English Edition), 2012,29(1):54−57.

[19] Michael R, Strichman O. Translation validation: From simulink to C. In: Proc. of CAV 2009. 2009. 696−701. [doi: 10.1007/978-3-642-02658- 4_57]

[20] Guo DQ, Lü JD, Wang SL, Tang T, Zhan NJ, Zhou DT. Formal analysis and verification of train control system of High-speed railway in China. Scientia Sinica Informationis, 2015,45(3):417−438 (in Chinese with English abstract).

[21] Jiang Y, Zhang HH, Li ZH, Deng YD, Song XY, Gu M, Sun JG. Design and optimization of multiclocked embedded systems using formal techniques. IEEE Trans. on Industrial Electronics, 2015,62(2):1270−1278.

[22] Jiang Y, Zhang HH, Zhang HF, Liu H, Song XY, Gu M, Sun JG. Design of mixed synchronous/asynchronous systems with multiple clocks. IEEE Trans. on Parallel Distributed Systems, 2015,26(8):2220−2232.

[23] Bae K, Olveczky PC, Feng TH, Tripakis S. Verifying ptolemy II discrete-event models using real-time maude. In: Proc. of the Int'l Conf. on Formal Engineering Methods. 2009. 717−736.

附中文参考文献:

[2] 王戟,詹乃军,冯新宇,刘志明.形式化方法概貌.软件学报,2019,30(1):33−61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[3] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(z1):1907−1912.

[13] 柳毅,麻志毅,何啸,邵维忠.一种从 UML 模型到可靠性分析模型的转换方法.软件学报,2010,21(2):287−304. http://www.jos.org.cn/1000-9825/3792.htm [doi: 10.3724/SP.J.1001.2010.03792]

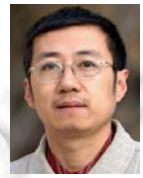
[14] 杨志斌,胡凯,赵永望,马殿富,Bodeveix JP.基于时间抽象状态机的 AADL 模型验证.软件学报,2015,26(2):202−222. http://www.jos.org.cn/1000- 9825/4776.htm [doi: 10.13328/j.cnki.jos.004776]

[20] 郭丹青,吕继东,王淑灵,唐涛,詹乃军,周达天.中国高速铁路列控系统的形式化分析与验证.中国科学:信息科学,2015,45(3):417−438.

陆芝浩(1995－),男,硕士,CCF 学生会员,主要研究领域为形式化验证.

王瑞(1981－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为形式化方法,软件安全验证.

孔辉(1978－),男,博士,CCF 专业会员,主要研究领域为形式化方法,混杂系统验证.



关永(1966－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为形式化验证,高可靠嵌入式系统.

施智平(1974－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为形式化方法,人工智能.