软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(11):2625−2638 [doi: 10.3724/SP.J.1001.2011.04005]

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

基于事件确定有限自动机的 UML2.0 序列图描述与验证*

张 琛[1,2], 段振华[1,2+], 田 聪[1,2]

[1](西安电子科技大学 计算理论与技术研究所,陕西 西安 710071)

[2](西安电子科技大学 ISN 国家重点实验室,陕西 西安 710071)

Specification and Verification of UML2.0 Sequence Diagrams Based on Event Deterministic Finite Automata

ZHANG Chen[1,2], DUAN Zhen-Hua[1,2+], TIAN Cong[1,2]

[1](Institute of Computing Theory and Technology, Xidian University, Xi'an 710071, China)

[2](State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China)

+ Corresponding author: E-mail: zhhduan@mail.xidian.edu.cn, http://www.xidian.edu.cn

Zhang C, Duan ZH, Tian C. Specification and verification of UML2.0 sequence diagrams based on event deterministic finite automata. *Journal of Software*, 2011,22(11):2625−2638. http://www.jos.org.cn/1000-9825/4005.htm

Abstract: To ensure the reliability of UML2.0 sequence diagrams (SD), which are used in software analysis and design, propositional projection temporal logic (PPTL) model checking is adopted in this paper. First, event deterministic finite automata (ETDFA) are proposed and used to describe the formal models of SD. Furthermore, an algorithm for model checking ETDFA with PPTL formulas being the properties is presented. Finally, based on the implementation of PPTL model checker, the ETDFA models of SD are verified. Experimental results show that the proposed method is useful in ensuring the reliability of SD.

Key words: UML2.0 sequence diagram; event deterministic finite automata; model checking; propositional projection temporal logic; verification

摘 要: 为了确保软件分析与设计阶段 UML2.0 序列图模型的可靠性,采用命题投影时序逻辑(propositional projection temporal logic,简称 PPTL)模型检测方法对该模型进行分析和验证.提出了事件确定有限自动机(event deterministic finite automata,简称 ETDFA),并使用该自动机为序列图建立形式化模型,通过给出的基于 ETDFA 的 PPTL 模型检测算法得到验证结果.该方法可以在基于 Spin 的 PPTL 模型检测器的支持下实现.实例结果表明,该方法可以验证序列图的性质并保证其可靠性.

关键词: UML2.0 序列图;事件确定有限自动机;模型检测;命题投影时序逻辑;验证

中图法分类号: TP311 **文献标识码**: A

统一建模语言(united modeling language,简称 UML)是对象管理组织(object management group,简称 OMG)

* 基金项目: 国家自然科学基金(60433010, 60873018, 60910004, 91018010, 61003078, 61003079, 61133001); 国家重点基础研究发展计划(973)(2010CB328102); 国家教育部博士点基金(200807010012); 中央高校基本科研业务费专项资金(JY10000903004)

收稿时间: 2010-10-23; 修改时间: 2010-12-09; 定稿时间: 2011-02-17

提出的第三代面向对象建模语言.它定义了多种图元,从不同视角和层次描述了系统的静态结构和动态特性,是一种定义良好、易于表达、功能强大且普遍适用的建模语言,广泛应用于各种领域,并已得到了工业界的支持.UML 序列图(sequence diagram,简称 SD)描述了对象间消息传递的时间顺序,用来表示用例中的行为顺序.序列图的主要目的是定义事件序列,并产生一系列期望的输出,其重点不是消息本身,而是消息产生的顺序.序列图强调了对象之间的消息交互,与 UML1.x 相比,UML2.0 序列图增加了组合片段的概念,大大增强了其建模能力.然而为了提高软件的正确性和可靠性,需要在软件开发的分析与设计阶段对所建立模型的正确性进行验证.

模型检测是一种形式化验证技术,其基本思想是,用状态迁移系统 S 表示系统的行为,用时序逻辑公式 F 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 S 是否是公式 F 的一个模型”,用公式表示为 $S\models F$.近年来出现的 UML 序列图模型检测方法包括:文献[1]给出的通过创建序列图的 Promela 模型,使用 Spin(simple promela interpreter)验证线性时序逻辑(linear temporal logic,简称 LTL)描述的性质;文献[2]给出的采用定理证明的方法验证序列图等.本文提出的基于事件有限自动机模型的 UML2.0 序列图模型检测方法,在验证序列图新特征对对象间更加复杂的交互行为建模的同时,使用命题投影时序逻辑[3]描述系统性质.PPTL 拥有强大的表达能力和自然直观的表达形式,在现实环境中更易于使用.序列图自动机模型的产生,也为进一步研究基于模型的测试奠定了基础.

1 相关工作

UML 序列图可以使用不同的形式化方法描述和验证.目前常见的序列图形式化方法包括:文献[4]给出的基于 B 方法的形式化描述,B 方法属于基于模型规约说明语言的范畴,是一种基于对象的形式化语言,它可以使整个程序和程序的规格说明处于一个统一的数学框架下,按照抽象机的方式理解系统.选择 B 方法作为 UML 模型的形式规约方法,具有以下优点:B 方法具有面向对象的类似特征,可以使 UML 模型到 B 方法形式规约的转换过程比较直观;B 方法有一整套严格的理论分析方法和工具,弥补了 UML 模型缺乏验证工具的问题;B 方法支持软件开发的几乎全过程,包括规约、精化、代码生成、正确性验证等.文献[5]提出了一种利用 Z 语言对序列图进行语义分析的方法,在序列图 Z 规范的基础上,用属性集表示对象状态,并将上下文表示为 Z 形式约束,通过检查上下文约束与对象状态间的一致性对序列图进行语义分析.而通过以上两种方法所产生的序列图形式化模型在使用过程中都存在对象间交互路径表达不清晰的问题.文献[6]提出了基于抽象状态机(abstract state machine,简称 ASM)的序列图形式化描述方法,ASM 能够在准确定义语义的同时避免时序逻辑与 Z 语言的复杂性.使用 ASM 对序列图语义进行建模的目的是,准确地描述模型特征,改进大型系统的测试过程.时序逻辑或 Z 语言能够给出序列图的语义描述,但对逻辑的依赖,使得该方法很难在工业界得到广泛应用.形式化语言通常在学习与使用过程中有相当大的难度,因此序列图语义的形式化定义通常很难看出其与程序语义之间的联系.在使用 ASM 建立简单序列图模型时,其逻辑语义清晰;但对于复杂的 UML2.0 序列图建模,由于缺乏精确的定义,给验证工作带来挑战.文献[7]给出的基于 XYZ/E 时序逻辑语义的序列图形式化描述,以线性时序逻辑为基础,所有程序单元均为合式公式,并且它在统一时序逻辑框架下既能表示系统的动态语义又能表示其静态语义,但是缺乏直观性.与 B 方法、ASM 等其他形式化方法相比,使用 XYZ/E 表示的语义范围更加宽泛.文献[1]给出了使用 Promela 语言描述 UML2.0 序列图的方法.该方法能够方便地实现使用模型检测器验证序列图的目的,但对所生成的 Promela 程序代码不利于进一步使用,特别是从该模型中产生测试用例.文献[8]将序列图中的事件动作及执行序列映射为进程代数中的进程表达式,利用进程代数语义框架构建序列图的形式语义.

Petri 网具有严格的数学定义和直观的图形表达方式,具有相对成熟的语义和可执行性,系统结构分析技术比较完善,可以对活性、可达性、有界性等性质进行有效验证,文献[9,10]分别给出了 Petri 网对序列图进行形式化描述的方法.在逻辑流程验证方面,Petri 网具有独特的优点:它综合了数据流、控制流和状态转移,能够自然地描述并发、同步、资源争用等特性,而且本身包含执行控制机制,集规范表示与执行于同一模型中.然而对于 Petri 网模型检测的可判定性,文献[11]指出,对于 1-safe 的 Petri 网,所有 LTL 和计算树逻辑(computation tree logic,简称 CTL)表达的属性都是可判定的;但对于任意的库所/变迁系统,其可判定性要考虑到表达属性的逻辑是基于

状态还是基于动作.

基于自动机的序列图形式化建模与验证方法也有不少研究成果.文献[12−14]在使用自动机形式化描述序列图的过程中,将 SD 所表示的系统看作一个对象,其优点是模型比较直观,在验证过程中状态数相对较少;但缺点是对 SD 描述的系统中每个对象状态的变迁刻画得不清晰,对象之间的交互表达的不明确.将有限自动机与 Petri 网进行比较:两者在交运算下都是封闭的;虽然 Petri 网的表达能力比有限自动机的表达能力更强,但 Petri 网的计算要比有限自动机的计算具有更高的空间复杂度.如果限制 Petri 网有界,则可以降低 Petri 网的计算空间复杂度.但对有界 Petri 网而言,其可达图可以用复杂的有限自动机来表示,所以有界 Petri 网并没有超过有限自动机的表达能力.标号迁移系统(labeled transition system,简称 LTS)[15]定义了状态、行为、迁移以及初始状态集,以四元组的形式给出了状态的迁移过程.与自动机相比,在 LTS 中没有给出可接受状态的定义,并且 LTS 可以描述无穷分支,因此更适合于对实时系统行为进行建模.I/O 自动机(input/output automata)模型是 Mark Tuttle 和 Nancy Lynch 提出的一种应用范围很广泛的自动机模型[16],主要用来对并发分布式离散事件系统进行建模,它能够描述几乎所有的同步或异步并发系统.I/O 自动机模型提供了一个框架,能够推理由交互的异步组成部分构成的系统的各种性质.I/O 自动机的签名由自动机的输入、输出和内部动作构成.在本文中,针对序列图描述的对象之间的交互,所发生的事件均被看作自动机的内部动作.传统的 I/O 自动机具有输入使能的特性,它是指一个 I/O 自动机在任何状态下都必须能够接受所有的输入动作.而在使用自动机描述 UML2.0 序列图所包含的选择、循环等组合片段时,首先需要判断事件发生的条件,即对动作执行条件存在约束.因此,I/O 自动机不适合描述序列图的交互过程.

确定有限自动机(deterministic finite automata,简称 DFA)是一类能够实现状态转移的有限状态自动机.对于一个给定的属于该自动机的状态和一个属于该自动机字母表Σ的字符,它都能根据事先定义的转移函数到达下一个状态,并且该状态是确定的.为了准确表达序列图中各对象在发送、接收消息后的状态变迁和对象之间在消息交互过程中的状态迁移,以及序列图新特征中组合片段对迁移关系产生的影响,本文采用基于事件扩展的 DFA,即事件确定有限自动机描述 UML2.0 序列图,对使用 PPTL 所描述的序列图性质进行验证.并且,基于自动机的验证技术相对成熟,为本文的研究工作提供了基础.

本文第 1 节给出相关工作的研究进展情况.第 2 节主要介绍 UML2.0 中序列图增加的新特征以及序列图的语法和语义.第 3 节详细讨论序列图的有穷状态自动机模型,并给出事件确定有限自动机的构造算法.第 4 节介绍基于 PPTL 的序列图验证方法以及 ETDFA 的 PPTL 性质模型检测算法.第 5 节以一个实例说明该方法的应用.第 5 节对本文工作进行总结.

2 序列图定义

UML2.0 序列图组合片段的增加,大大增强了其在面向对象系统交互需求分析与设计中的建模能力.通过交互对象间传递的消息序列描述用例图中表达的场景,每个消息序列代表该用例的一个可能或无效的事件流.

2.1 UML2.0序列图的新特征

UML2.0序列图中,添加了12种组合片段[17],其中,*loop*,*opt*,*alt*,*break*,*par*,*neg*,*ref*会使执行路径产生分支,因此具有十分重要的作用,将作为本文研究的重点.

loop:指明循环执行的最小、最大次数,或当条件为真时执行循环.

opt:存在单一运算单元,条件为真时执行,类似于 if 语句无 else.包含在此片段中的交互,只有在监护条件为真时才会执行.

alt:条件为真的运算单元执行,可以使用 else 关键字代替布尔表达式.

break:如果监护条件为真,运算单元被执行,而不是该交互的其余部分.*break* 操作符具有单一监护条件,如果它为真,*break* 主体被执行,并且 *loop* 被终止.其要点是 *break* 之后 *loop* 的剩余部分不执行.

par:并发组合片段,该组合片段并发区域中的事件可交替执行.

neg:运算单元显示无效的、不必发生的交互.

ref:组合区引用其他交互.

2.2 序列图的语法

为了能够准确地描述序列图的动态交互过程,抽象对象生命周期内事件发生的序列,本文给出了 UML2.0 中序列图的形式化定义,其中包括了事件之间的二元关系以及 UML2.0 序列图所增加的新特征.

定义 1(序列图). 序列图 SD 用一个十三元组表示:$SD=(P,E,S,R,M,FG,OP,C,msg,obj,frag,\rightarrow,\prec)$,其中,

P 是有穷对象的集合.

E 是有穷事件的集合.

S 是发送事件集.

R 是接收事件集,$E=S\cup R$,并且 $S\cap R=\varnothing$.

M 是有穷消息的集合.每个消息 $m\in M$,与两个存在因果关系的事件相连,一个是消息发送事件$!m\in S$,一个消息接收事件$?m\in R$,$!m$ 和$?m$ 构成了一对同步事件.在 UML2.0 中有 5 种类型的消息,分别是:同步消息、异步消息、返回消息、参与者创建消息和参与者销毁消息.分析模型中,异步消息和同步消息之间的区别无关轻重,往往不关心消息发送的详细语义,仅关心消息被发送的事实.本文主要考虑同步消息,这是最受约束的情况.

FG 是组合片段的集合,同名组合片段分别用添加自然数后缀进行区分.

OP 是组合片段操作域集合,使用组合片段名与执行条件表示,*par* 等无执行条件操作域分别由从 1 开始的自然数标识.

C 是组合片段执行条件的集合,执行条件可为空,表示为ε.

msg 是从 E 到 M 的一个函数关系.$msg(e)\in M$,表示事件 e 所对应的消息.

obj 是从 E 到 P 的一个函数关系.$obj(e)\in P$,表示事件 e 所对应的对象.对象 P_i 上所有事件的集合记为 E_i,$E_i=\{e|e\in E\wedge obj(e)\in P_i\}$.

$frag$ 是从 E 到 FG 的一个函数关系.$frag(e)\in FG$,表示事件 e 所属的组合片段及其所在片段的操作域,存在于多个组合片段的事件可用嵌套片段标识,对于每个操作域中的第 1 个事件 e 使用标识符ⓢ表示,最后一个事件 e 使用标识符ⓕ表示.

$\rightarrow$是消息集合 M 上的一个全序关系.表示序列图中的消息在纵向时间轴上的先后关系.即在一个对象的生命线上,位于上方的事件先于下方的事件发生.

$\prec$是发送事件与接收事件之间的一个二元关系.在两个对象之间,消息 m 的发送事件先于接收事件发生,即$\{(!m,?m)|m\in M\}=\prec$.

图 1(b)给出了如图 1(a)所示序列图的形式化定义.

2.3 序列图中单个对象的形式化定义

序列图表达了多个对象的消息交互,而单个对象的形式化定义如下:

定义 2(序列图对象). 序列图中的对象由六元组表示:$P=(E,FG,OP,C,num,frag)$,其中,

E 表示对象接收事件和发送事件的集合,$|E|$是事件 E 的个数;

FG 表示对象上事件所在组合片段的集合;

OP 表示组合片段操作域的集合;

C 是组合片段执行条件的集合;

num 是事件发生顺序的列表,第 1 个发生的事件标记为 1,第 2 个发生的事件标记为 2,依此递推;

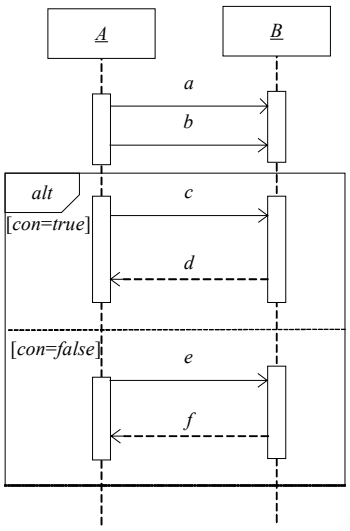
$frag$ 是从事件 E 到组合片段 FG 的一个函数关系.

表 1 给出了六元组之间的关系.

对于出现在同一序列图内的同名组合片段,依次使用“组合片段名+序号”表示,序号为从 1 开始的自然数.

图 1(a)中,对象 A 的形式化描述为:$A=(\{!a,!b,!c,?d,!e,?f\},\{alt\},\{\langle alt,true\rangle,\langle alt,false\rangle\},\{\varepsilon,true,false\},\{1,\ldots,6\},\{\langle !a,null\rangle,\langle !b,null\rangle,\langle !c,alt[true],ⓢ\rangle,\langle ?d,alt[true],ⓕ\rangle,\langle !e,alt[false],ⓢ\rangle,\langle ?f,alt[false],ⓕ\rangle\})$.六元组关系见表 2,“•”表

示事件发生的组合片段与操作域,“*SD*”表示序列图中任意组合片段外发生的事件.



(a) Sequence diagram
(a) 序列图

- $P=\{A,B\}$;
- $E=\{!a,?a,!b,?b,!c,?c,!d,?d,!e,?e,!f,?f\}$;
- $S=\{!a,!b,!c,!d,!e,!f\}$;
- $R=\{?a,?b,?c,?d,?e,?f\}$;
- $M=\{a,b,c,d,e,f\}$;
- $FG=\{alt\}$;
- $OP=\{\langle alt,true\rangle,\langle alt,false\rangle\}$;
- $C=\{\varepsilon,true,false\}$;
- $msg=\{\langle !a,a\rangle,\langle ?a,a\rangle,\langle !b,b\rangle,\langle ?b,b\rangle,\langle !c,c\rangle,\langle ?c,c\rangle,\langle !d,d\rangle,\langle ?d,d\rangle,\langle !e,e\rangle,\langle ?e,e\rangle,\langle !f,f\rangle,\langle ?f,f\rangle\}$;
- $obj=\{\langle !a,A\rangle,\langle ?a,B\rangle,\langle !b,A\rangle,\langle ?b,B\rangle,\langle !c,A\rangle,\langle ?c,B\rangle,\langle !d,B\rangle,\langle ?d,A\rangle,\langle !e,A\rangle,\langle ?e,B\rangle,\langle !f,B\rangle,\langle ?f,A\rangle\}$;
- $frag=\{\langle !a,null\rangle,\langle ?a,null\rangle,\langle !b,null\rangle,\langle ?b,null\rangle,\langle !c,alt[true],ⓢ\rangle,\langle ?c,alt[true],ⓢ\rangle,\langle !d,alt[true],ⓕ\rangle,\langle ?d,alt[true],ⓕ\rangle,\langle !e,alt[false],ⓢ\rangle,\langle ?e,alt[false],ⓢ\rangle,\langle !f,alt[false],ⓕ\rangle,\langle ?f,alt[false],ⓕ\rangle\}$;
- $\rightarrow=\{a\rightarrow b\rightarrow c\rightarrow d\rightarrow e\rightarrow f\}$;
- $\prec=\{!a\prec ?a,!b\prec ?b,!c\prec ?c,!d\prec ?d,!e\prec ?e,!f\prec ?f\}$

(b) Syntax of sequence diagram
(b) 序列图语法

Fig.1 Formal description of sequence diagram
图 1 序列图的形式化描述

Table 1 Relation of 5-tuple
表 1 五元组之间的关系

First fragment		f_1						
Operand/Guard condition		op_1	op_2	…				
Second fragment					f_m			
Operand/Guard condition					op_1	op_2	…	
…								
Num	*Event*	*Frag*						
1	e_1							
2	e_2							
…								

Table 2 5-Tuple of object *A*
表 2 对象 *A* 的五元组关系

First fragment		*SD*	*alt*			
Operand/Guard condition			*True*		*False*	
Num	*Event*	*Frag*				
1	!*a*	•				
2	!*b*	•				
3	!*c*		•	ⓢ		
4	?*d*		•	ⓕ		
5	!*e*				•	ⓢ
6	?*f*				•	ⓕ

2.4 序列图的语义

序列图所描述的场景表现为系统执行路径的事件序列.定义 1 中的二元关系能够充分描述 UML1.x 中基本序列图的语义[18],但不能满足 UML2.0 序列图所增加的新特性,如 *opt*,*loop* 等.针对此问题,下面引入文献[19]给出的迹语义:描述场景执行的事件序列称作迹;可能执行的事件序列称作肯定迹,不希望或禁止执行的事件序列称作否定迹.在 UML2.0 序列图中使用迹表示所有可能的事件序列.

3 序列图的有穷状态自动机模型

结合 UML 和形式化方法的优点,将非形式化的 UML 序列图转换为具有精确语义定义的形式化规范,在非形式化的图形表示与形式化定义之间建立映射关系,为序列图的分析和验证奠定了基础.

3.1 事件确定有限自动机ETDFA

ETDFA 是确定有限自动机的扩展,本文将 ETDFA 作为描述序列图的形式化模型.ETDFA 的状态表达了序列图中事件向对象的映射;ETDFA 的一个状态迁移描述了一次消息交互,迁移的标注是二元组:第 1 项为事件发生条件,第 2 项为所发生的事件.状态迁移表示在消息传递过程中,对象接受或者发送消息之后,从一个状态转移到另一状态.显然,ETDFA 可以给出序列图所描述的交互对象间传递的消息序列,通过多个对象的积自动机能够刻画系统行为的交互过程.

定义 3(ETDFA). 事件确定有限自动机 M 是一个七元组:$M=(Q,C_M,E_M,\Sigma,\delta,q_0,F)$,其中,

Q 表示状态的非空有穷集合,$\forall q\in Q$,q 称为 M 的一个状态;

C_M 是组合片段执行条件的集合,执行条件可为空,表示为ε;

E_M 是有穷事件的集合;

Σ表示输入字母表,输入字符串都是Σ上的字符串,$\Sigma=\{(c,e)|c\in C_M,e\in E_M\}$;

δ表示状态转移函数,$Q\times\Sigma\to Q$;

q_0 表示 M 的初始状态,$q_0\in Q$;

F 表示 M 的终止状态集合,$F\subseteq Q$.

3.2 自动机的构造

序列图表达了系统中多个对象的消息交互,而针对单个对象,以下给出不包括组合片段 *neg* 的自动机构造算法,所构造的自动机为事件确定有限自动机.

定义 4(ETDFA 构造算法). 令对象六元组 $P=(E,FG,OP,C,num,frag)$,对应的事件确定有限自动机 AM_p 可表示为一个七元组 $AM_p=(Q,C_M,E_M,\Sigma,\delta,q_0,F)$,其中,

$\forall p\in P$ 对应于一个自动机 AM_p;

创建初始状态:$q_0\in Q$ 为初始状态,在该状态没有记录事件,即$\forall p\in P$,$q_0(p)=\varepsilon$;

输入字母表Σ是 C_M,E_M 的集合.$C_M=\{c|c\in C\}$,$E_M=\{e|e\in E\}$,$\Sigma=\{(c,e)|c\in C_M,e\in E_M\}$;

δ为状态迁移的集合,$Q\times\Sigma\to Q$,每个迁移可表示为 C_M/E_M,即(标记执行条件/消息事件);

F 为 AM_p 的接受状态,在描述不包含组合片段的序列图时,可接受状态是唯一的;

创建非初始状态:Q 为状态的集合,$\forall q_i\in Q(i\geqslant 1)$对应于第 i 步上,事件序列到对象 p 的映射.

算法 1. 构造序列图对象的事件确定有限自动机.

输入:序列图 SD 中对象 p 的形式化定义.

输出:对象 p 的事件确定有限自动机 ETDFA.

(1) 创建初始状态 q_0.

(2) 当 $num=1$ 时,创建状态 q_1:

① 若 e_1 不在任意组合片段内或在组合片段 *par* 内,$\delta(q_0,q_1)=e_1$;

② 若 e_1 在组合片段 *opt*,*alt*,*loop*,*break* 内,$\delta(q_0,q_1)=[con=$组合片段执行条件$]/e_1$;

③ 若 e_1 在嵌套的多个组合片段内,则$\delta(q_0,q_1)=[con=$嵌套组合片段执行条件的交集$]/e_1$;

④ 若 e_1 为该对象的最后一个事件,q_1 为 AM_p 的可接受状态.

(3) 当 $num=i+1$ 时,创建状态 q_{i+1}:

① 若 e_i 与 e_{i+1} 均不在任意组合片段内,$\delta(q_i,q_{i+1})=e_{i+1}$;

② 若 e_{i+1} 在组合片段 *par* 内,e_i 在该组合片段外,$\delta(q_i,q_{i+1})=e_{i+1}$;

③ 若 e_i 与 e_{i+1} 在组合片段 *opt*,*alt*,*loop*,*break*,*par* 的相同操作域内,$\delta(q_i,q_{i+1})=e_{i+1}$;

④ 若 e_{i+1} 在单操作域组合片段 *opt*,*loop*,*break* 内,而 e_i 在该组合片段之外,则$\delta(q_i,q_{i+1})$=[*con*=组合片段执行条件]/e_{i+1};

⑤若 e_{i+1} 在多操作域组合片段 *alt* 内,e_i 在该组合片段外,则$\delta(q_i,q_{i+1})$=[*con*=组合片段 *alt* 中 e_{i+1} 所在操作域的执行条件]/e_{i+1};

⑥ 若 e_i 为多操作域组合片段 *alt* 的任意有效操作域内发生的最后一个事件,e_{i+1} 在 e_i 所在操作域的下一有效域操作内,则创建状态 q_{i+1}(迁移关系在步骤(4)中实现),其中,有效操作域是指该操作域中存在发生的事件;

⑦ 若 e_i 为组合片段 *par* 当前操作域内发生的最后一个事件,e_{i+1} 为组合片段 *par* 下一有效操作域内发生的第 1 个事件,则$\delta(q_i,q_{i+1})=e_{i+1}$;

⑧ 若 e_i 为组合片段 *opt*,*loop* 内发生的最后一个事件,e_{i+1} 为组合片段外的第 1 个事件,则$\delta(q_i,q_{i+1})=e_{i+1}$;

⑨ 若 e_i 为组合片段 *par*,*alt* 最后一个有效操作域内发生的最后一个事件,e_{i+1} 为组合片段外发生的第 1 个事件,则$\delta(q_i,q_{i+1})=e_{i+1}$;

⑩ 若 e_i 为组合片段 *break* 内发生的最后一个事件,e_{i+1} 为 *break* 外的第 1 个事件,则$\delta(q_{i-|break|},q_{i+1})=e_{i+1}$,其中,|组合片段名|表示组合片段所包含事件的个数;

⑪ 若 e_{i+1} 为组合片段 *alt*,*opt*,*loop*,*break*,*par* 内发生的最后一个事件,且 e_{i+2} 不存在,则 q_{i+1} 为 AM_p 的可接受状态;

⑫ 若 e_{i+1} 不在任意组合片段内,且 e_{i+2} 不存在,则 q_{i+1} 为 AM_p 的可接受状态;

⑬ 若 e_{i+1} 为组合片段 *alt*,*par* 任意有效操作域内发生的最后一个事件,且该片段之后无事件发生,则 q_{i+1} 为 AM_p 的可接受状态;

⑭ 若 e_{i+1} 在单操作域组合片段 *opt*,*loop*,*break* 内,e_i 在该组合片段之外,$e_{i+|opt|/|loop|/|break|+1}$ 不存在,则 q_i 为 AM_p 的可接受状态;

⑮ 若 e_{i+1} 在多操作域组合片段 *alt* 内,而 e_i 在该组合片段之外,$e_{i+|alt|+1}$ 不存在,且 *alt* 组合片段各操作域执行条件的并集不为全集,则 q_i 为 AM_p 的可接受状态;

⑯ 若 e_{i+1} 为片段 *break* 内发生的最后一个事件,且 *break* 组合片段之外未嵌套其他组合片段,则 q_{i+1} 为 AM_p 的可接受状态.

通过以上步骤依次创建 ETDFA 中所包含的“事件数+1”个状态.

(4) ① 若 e_i 在 *opt* 组合片段外,e_{i+1} 在 *opt* 组合片段内,$e_{i+|opt|+1}$ 为 *opt* 组合片段外的第 1 个事件,则$\delta(q_i,q_{i+|opt|+1})=e_{i+|opt|+1}$;

② 若 e_i 在当前 *alt* 组合片段外,e_{i+1},e_j,e_k 等分别为 *alt* 组合片段中各相邻有效操作域内第 1 个发生的事件,则

$$\delta(q_i,q_j)=[con=\text{组合片段 } alt \text{ 中 } e_j \text{ 所在操作域的执行条件}]/e_j,$$

$$\delta(q_i,q_k)=[con=\text{组合片段 } alt \text{ 中 } e_k \text{ 所在操作域的执行条件}]/e_k;$$

③ 若 e_{j-1},e_{k-1},e_{l-1} 分别是 *alt* 组合片段中各相邻有效操作域内最后一个发生的事件,e_l 为 *alt* 组合片段外的第 1 个事件,则$\delta(q_{j-1},q_l)=e_l$,$\delta(q_{k-1},q_l)=e_l$;

④ 若 e_i 在 *alt* 组合片段外,e_{i+1} 在 *alt* 组合片段内,$e_{i+|alt|+1}$ 为 *alt* 组合片段外的第 1 个事件,且该组合片段各操作域执行条件的并集不为全集,则$\delta(q_i,q_{i+|alt|+1})=e_{i+|alt|+1}$;

⑤ 若 e_i 为组合片段 *loop* 内发生的最后一个事件,e_h 为该组合片段内发生的第 1 个事件,则$\delta(q_i,q_h)$=[*con*=组合片段执行条件]/e_h;

⑥ 若 e_i 在 *loop* 组合片段外,e_{i+1} 在该组合片段内,$e_{i+|loop|+1}$ 为 *loop* 组合片段外的第 1 个事件,则$\delta(q_i,q_{i+|loop|+1})=e_{i+|loop|+1}$;

⑦ 若 e_i 在 *par* 组合片段外,e_{i+1},e_j,e_k 分别为 *par* 组合片段中各相邻有效操作域内第 1 个发生的事件,则$\delta(q_i,q_j)=e_j$,$\delta(q_i,q_k)=e_k$;

⑧ 若 e_{j-1},e_{k-1},e_{l-1} 等分别是 *par* 组合片段中各相邻有效操作域内最后一个发生的事件,e_l 为 *par* 组合片段外的第 1 个事件,则$\delta(q_{j-1},q_l)=e_l,\delta(q_{k-1},q_l)=e_l$;

⑨ 若 e_i 在组合片段 *par* 的任意操作域内,$\delta(q_i,q_j)$="e_i 所在操作域内的、事件 e_i 之后的首个消息事件,或者 *par* 组合片段中其他有效操作域内尚未发生的、且在消息的全序关系中对应于最先发生的消息事件";

⑩ 若 e_i 为组合片段 *break* 内发生的最后一个事件,e_k 是 *break* 组合片段所在的封闭交互序列外的第 1 个事件,且 e_k 不存在于 *opt*,*alt*,*loop*,*break* 组合片段中,则$\delta(q_i,q_k)=e_k$;

⑪ 若 e_i 为组合片段 *break* 内发生的最后一个事件,e_k 是 *break* 组合片段所在的封闭交互序列外的第 1 个事件,且 e_k 存在于 *opt*,*alt*,*loop*,*break* 组合片段中,则$\delta(q_i,q_k)$=[*con*=组合片段执行条件]/e_k.

(5) 通过以上步骤,得到序列图单个对象的 ETDFA.

在构造自动机时,若序列图包含 *neg* 组合片段,则对应于该片段内消息事件所产生的状态均为无效状态且不可执行.

4 基于 PPTL 的序列图验证方法

模型检测是对有限状态系统的一种形式化确认方法.具体做法是:采用一种形式语言描述系统的规范说明,构造一种算法来遍历根据系统规格说明设计的实现模型,确认实现模型是否满足系统的规范说明.

投影时序逻辑(projection temporal logic,简称 PTL)是一种用于描述离散区间或时段的逻辑系统,作为命题区间时序逻辑的一个扩展,其拥有强大的表达能力和自然直观的表达形式,在现实应用中更容易使用,并且命题投影时序逻辑的可判定性问题已经得到证明[20],这也使得对 PPTL 公式的模型检测变得现实可行.本节通过讨论系统自动机是否满足给定命题投影时序逻辑公式所描述的性质,来验证序列图的正确性.

4.1 序列图属性描述

在模型检测过程中,序列图的属性需要一种性质描述语言来表达,而 Spin 的性质规范语言是命题线性时序逻辑(propositional linear temporal logic,简称 PLTL),其表达能力有限,不能表达所有的正则表达式,使得一些性质,如"命题 P 必须在每个偶数状态成立"的验证无法自动完成.与 PLTL 相比,PPTL 则可以表达所有的正则表达式,能够描述更多的性质,更适合作为软件需求的性质描述语言.因此,本文采用命题投影时序逻辑描述待验证的系统性质.PPTL 是 PTL 的命题形式,不包含变量、谓词和量词等一阶成分.

用 *Prop* 表示原子命题的集合,PPTL 公式归纳定义如下:

$$P::=p|P_1\vee P_2|\neg P_1|\bigcirc P_1|(P_1,\ldots,P_m)\ prj\ Q,$$

其中,$P\in Prop$,$P_1,\ldots,P_m$ 和 Q 为 PPTL 公式.

PPTL 公式包括的时序操作符有:○(next),□(always),◇(sometimes),⊙(weak next),;(chop),$\overline{;}$(chop in the past),*prj*(projection).

常用的推导公式有:$empty\equiv\neg\bigcirc true$,$more\equiv\bigcirc true$,$\Diamond P\equiv true;P$,$\diamondsuit P\equiv P\,\overline{;}\,true$,$\Box P\equiv\neg\Diamond\neg P$,$\odot P\equiv empty\vee\bigcirc P$,$true\equiv P\vee\neg P$,$false\equiv P\wedge\neg P$ 等.

4.2 基于PPTL的序列图模型检测

使用模型检测器验证序列图是否满足系统属性,需要将待验证的性质用模型检测器可接受的形式表示.如果模型检测器得到该属性的反例,则说明模型和性质是不一致的.根据反例的路径,可以找出序列图在建模过程中存在的问题;如果没有得到反例,则说明模型可以满足序列图所表达的性质.模型检测器作为一种自动化的验证工具,具有验证速度快、效率高,并且可以得到违反属性的执行路径的优点.

本文在文献[21−23]等研究成果基础上给出了序列图模型的 PPTL 验证方法,如图 2 所示.并采用改进后的 Spin(simple promela interpreter)作为验证工具,验证步骤如下:

(1) 使用 UML2.0 序列图表示软件建模的非形式化模型;

(2) 采用 ETDFA 构造算法得到序列图中各个对象的形式化语义;

(3) 采用 ETDFA 的 PPTL 性质模型检测算法得到验证结果.

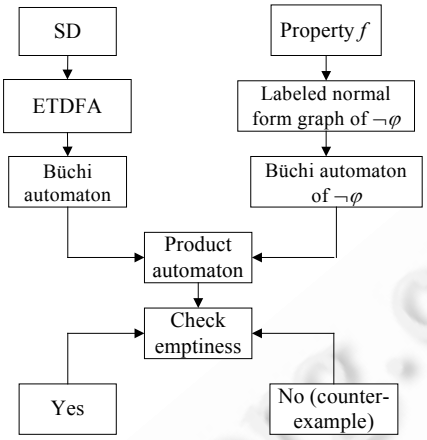


Fig.2 ETDFA based model checking PPTL

图 2 基于事件有限自动机的 PPTL 模型检测

定义 5(ETDFA 的 PPTL 性质模型检测算法). 令事件确定有限自动机 $AM=(Q,C_M,E_M,\Sigma,\delta,q_0,F)$,描述系统属性的 PPTL 公式为 φ,验证结果由是否产生反例给出.

算法 2. 事件确定有限自动机的 PPTL 性质模型检测.

输入:事件确定有限自动机 AM,待验证性质 φ.

输出:验证结果.

(1) 在模型检测过程中,性质描述语言采用 Büchi 自动机表示,因此需要将表达系统模型的 ETDFA 转换为 Büchi 自动机,通过判断两个 Büchi 自动机之间的包含关系来确认模型是否满足性质.

根据 Stutter 扩展规则给出与 ETDFA 等价的 Büchi 自动机 BA_e,$BA_e=(Q^*,\Sigma^*,\delta^*,q_0^*,F^*)$,其中,

① Q^*表示状态的非空有穷集合,且 $Q^*=Q$;

② Σ^*表示输入字母表,$\Sigma^*=\Sigma\cup\varepsilon$,$\varepsilon$为无效字符;

③ q_0^*表示初始状态;

④ F^*表示可接受状态的集合,$F^*\subseteq Q^*$,$\exists f\in F^*, f\times\varepsilon\to f$;

⑤ δ^*表示状态转移函数,$\delta^*=\delta\cup\delta_f$,$\delta_f=\{f\times\varepsilon\to f\}$,说明可接受状态 f 在读入 ε 后仍停留在该状态,即执行了空操作.

(2) 给出性质 φ 的非,基于文献[23]提出的方法,通过带标记的范式图表示 $\neg\varphi$ 的范式.

(3) 在 PPTL 判定过程的基础上,将带标记的范式图转换为 Büchi 自动机 $BA_{\neg\varphi}$.

(4) 求 BA_e 与 $BA_{\neg\varphi}$ 的积自动机.

(5) 积自动机代表了违反性质 φ 的所有计算,通过对积自动机进行判空,来判定系统模型是否满足公式 φ. 若模型满足所期望的性质,则给出相应的确认信息;若模型不满足性质,则通过调用模拟模式给出相应的反例.

5 实例分析

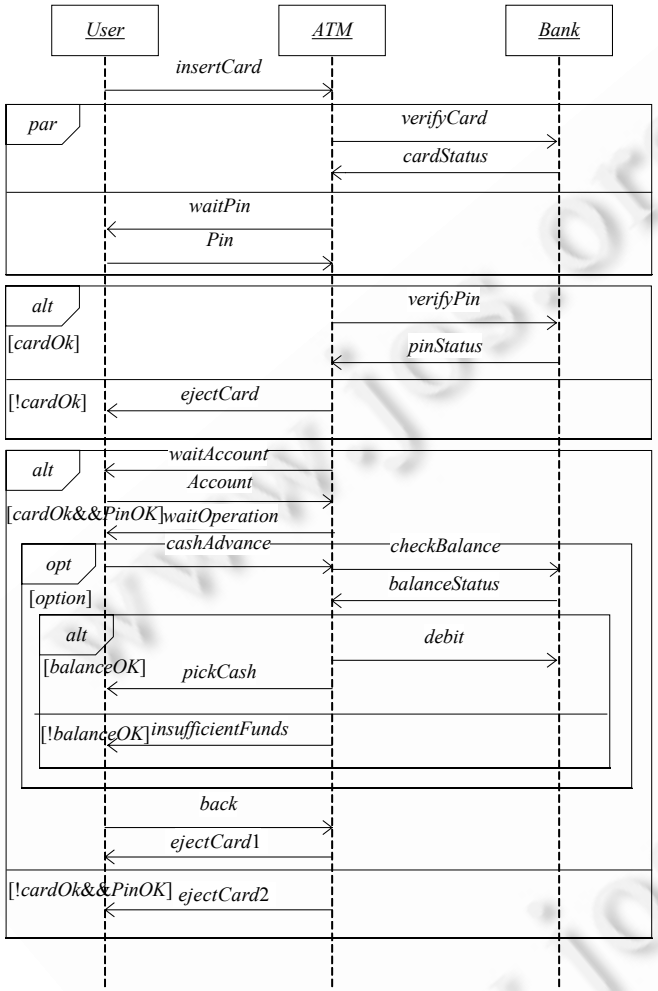
目前,我们已实现了一个基于 PPTL 的模型检测器原型 XSpin,下面通过一个 ATM 实例[1]来说明 UML2.0 序列图的自动机表示与模型检测过程.

5.1 ATM的序列图描述

图 3(a)所示序列图描述了使用 ATM 机进行操作经历的几种状态,以及各状态之间的转换条件.其中,ATM 与 *User* 和 *Bank* 两个对象进行交互.*User* 插入卡后,ATM 验证卡信息以及用户密码,若验证失败,则卡退出;若验证正确,则进行存取款操作.序列图的十三元组定义为

$$SD=(P,E,S,R,M,FG,OP,C,msg,obj,frag,\rightarrow,\prec),$$

其中,各元组部分定义如图 3(b)所示.



(a) ATM sequence diagram
(a) ATM 序列图

- P={*User*,*ATM*,*Bank*};
- E={!*insertCard*,!*verifyCard*,!*cardStatus*,!*waitPin*, !*Pin*,!*verifyPin*,!*pinStatus*,!*ejectCard*, !*waitAccount*,!*Account*,!*waitOperation*, !*cashAdvance*,…}∪{?*insertCard*,?*verifyCard*, ?*cardStatus*,?*waitPin*,?*Pin*,?*verifyPin*,…};
- S={!*insertCard*,!*verifyCard*,!*cardStatus*,!*waitPin*, !*Pin*,!*verifyPin*,…};
- R={?*insertCard*,?*verifyCard*,?*cardStatus*,?*waitPin*, ?*Pin*,?*verifyPin*,…};
- M={*insertCard*,*verifyCard*,*cardStatus*,*waitPin*, *Pin*,*verifyPin*,*pinStatus*,*ejectCard*, *waitAccount*,*Account*,*waitOperation*, *cashAdvance*,*checkBalance*,…};
- FG={*par*,*alt*1,*alt*2,*opt*,*alt*3};
- OP={⟨*par*,1⟩,⟨*par*,2⟩,⟨*alt*1,*cardOK*⟩, ⟨*alt*1,!*cardOK*⟩,⟨*alt*2,*cardOK*&&*PinOK*⟩, ⟨*alt*2,!*cardOK*&&*PinOK*⟩, ⟨*alt*2(*cardOK*&&*PinOK*)[*opt*],*option*⟩, ⟨*alt*2(*cardOK*&&*PinOK*)[*opt*(*option*)[*alt*3]], *balanceOK*⟩, ⟨*alt*2(*cardOK*&&*PinOK*)[*opt*(*option*)[*alt*3]], !*balanceOK*⟩};
- C={*ε*,*cardOK*,!*cardOK*,*cardOK*&&*PinOK*, !*cardOK*&&*PinOK*,*option*,…};
- msg={⟨!*insertCard*,*insertCard*⟩,⟨?*insertCard*, *insertCard*⟩,⟨!*verifyCard*,*verifyCard*⟩, ⟨?*verifyCard*,*verifyCard*⟩,…};
- obj={⟨!*insertCard*,*User*⟩,⟨?*insertCard*,*ATM*⟩, ⟨!*verifyCard*,*ATM*⟩,⟨?*verifyCard*,*Bank*⟩, ⟨!*cardStatus*,*Bank*⟩,⟨?*cardStatus*,*ATM*⟩,…};
- $frag$={⟨!*insertCard*,*SD*⟩,⟨?*insertCard*,*SD*⟩, ⟨!*verifyCard*,*par*(1),*par*[ⓢ]⟩, ⟨?*verifyCard*,*par*(1),*par*[ⓢ]⟩, ⟨!*cardStatus*,*par*(1),*par*[ⓕ]⟩, ⟨?*cardStatus*,*par*(1),*par*[ⓕ]⟩,…};
- $\rightarrow$={*insertCard*→*verifyCard*→*cardStatus*→ *waitPin*→*Pin*→*verifyPin*→*pinStatus*→ *ejectCard*→*waitAccount*→*Account*→ *waitOperation*→*cashAdvance*→ *checkBalance*→*balanceStatus*→*debit*→ *pickCash*→*InsufficienceFunds*→*back*→ *ejectCard1*→*ejectCard2*};
- $\prec$={!*insertCard*≺?*insertCard*, !*verifyCard*≺?*verifyCard*,…}.

(b) Syntax of ATM
(b) ATM 语法

Fig.3 Formal description of ATM sequence diagram

图 3 ATM 序列图的形式化描述

表 3 给出了对象 *User* 定义的六元组关系.

Table 3 6-Tuple of object *User*

表 3 对象 *User* 的六元组关系

First fragment		*SD*	*Par*		*alt*1		*alt*2				
Operand/Guard condition			1	2	*cardOK*	!*cardOK*	*CardOK&&PinOK*				!*cardOK&&PinOK*
Second fragment								*opt*			
Operand/Guard condition								*option*			
Third fragment									*alt*3		
Operand/Guard condition									*balanceOK*	!*balanceOK*	
Num	*Event*	*Frag*									
1	!*insertCard*	•									
2	?*waitPin*			•	ⓢ						
3	!*Pin*			•	ⓕ						
4	?*ejectCard*					•	ⓢ ⓕ				
5	?*waitAccount*						•	ⓢ			
6	!*Account*						•				
7	?*waitOperation*						•				
8	!*cashAdvance*							•	ⓢ		
9	?*pickCash*								•	ⓢ ⓕ	
10	?*insufficientFunds*									•	ⓢ ⓕ
11	!*back*						•				
12	?*ejectCard*1						•	ⓕ			
13	?*ejectCard*2										• ⓢ ⓕ

5.2 ATM的自动机表示

如图 4 所示的自动机为使用第 3 节描述的构造算法得到的 ATM 序列图中对象 *User* 的 ETDFA.使用同样的规则,分别可以得到对象 ATM 以及对象 *Bank* 的 ETDFA.对象 *User* 的 ETDFA 创建过程如下:

(1) 创建初始状态 q_0.

(2) 输入字母表:

$\Sigma=\{(c,e)|c\in C_M,e\in E_M\}$;

$C_M=\{\langle alt1,cardOk\rangle,\langle alt1,!cardOk\rangle,\langle alt2,cardOk\&\&PinOk\rangle,\langle alt2,!cardOk\&\&PinOk\rangle,\langle opt,option\rangle,$
$\langle alt3,balanceOk\rangle,\langle alt3,!balanceOk\rangle\}$;

$E_M=\{!insertCard,?waitPin,!Pin,?ejectCard,?waitAccount,!Account,?waitOperation,!cashAdvance,$
$?pickCash,?InsufficienceFunds,!back,?ejectCard1,?ejectCard2\}$.

(3) 由构造算法依次创建状态 $q_1,q_2,\ldots,q_{13}$.

(4) 状态迁移函数为δ.

(5) 状态 q_{12},q_{13} 为可接受状态.

通过以上步骤得到了自动机的七元组定义:

$ETDFA=(\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_{10},q_{11},q_{12},q_{13}\},\{C_M\},\{E_M\},\{(C_M\times E_M)\},$
$\{\delta(q_0,!insertCard)=q_1,\delta(q_1,?waitPin)=q_2,\delta(q_2,!Pin)=q_3,\delta(q_3,[con=!cardOk]/?ejectCard)=q_4,$
$\delta(q_4,[con=cardOk\&\&PinOK]/?waitAccount)=q_5,\delta(q_4,[con=!(cardOk\&\&PinOK)]/?ejectCard2)=q_{13},$
$\delta(q_5,!Account)=q_6,\delta(q_6,?waitOperation)=q_7,\delta(q_7,[con=option]/!cashAdvance)=q_8,$
$\delta(q_8,[con=balanceOk]/?pickCash)=q_9,\delta(q_8,[con=!balanceOk]/?insufficienceFunds)=q_{10},$
$\delta(q_9,!back)=q_{11},\delta(q_{10},!back)=q_{11},\delta(q_{11},?ejectCard1)=q_{12}\},\{q_0\},\{q_{12},q_{13}\})$.

5.3 序列图属性的PPTL描述

序列图待验证的部分属性及其 PPTL 性质描述如下:

(1) 若卡无效或用户密码不正确,则卡退出并终止服务.公式为

$$\square((\neg p\vee\neg q)\rightarrow\bigcirc(f\wedge empty)) \quad (1)$$

其中,p 为真表示卡有效,q 为真表示用户密码正确,f 为真表示卡退出.

(2) 检查账户余额并确定余额充足,可进行取款操作.公式为

$$f \rightarrow \Diamond p \wedge fin(q) \tag{2}$$

其中,p 为检查账户余额,q 为余额充足,f 为进行取款.

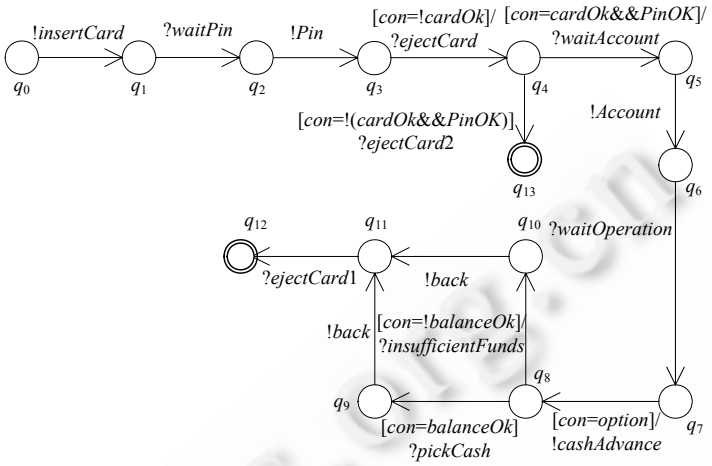


Fig.4 ETDFA of object *User*

图 4 对象 *User* 的 ETDFA

5.4 验证结果

如图 5 所示的自动机为使用 ETDFA 的 PPTL 性质模型检测算法得到的对象 *User* 的 Büchi 自动机.

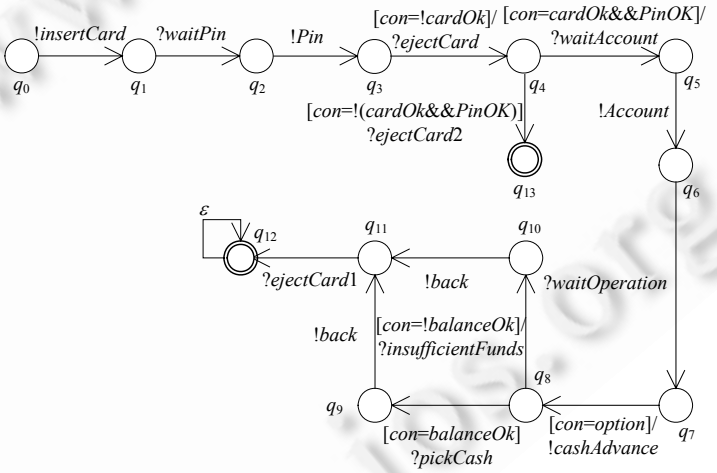


Fig.5 Büchi automaton of object *User*

图 5 对象 *User* 的 Büchi 自动机

将 ATM 中各对象的 Büchi 自动机作为输入,若模型检测器运行该实例后未得到反例,则模型满足性质(1)与性质(2).以性质(1)为例,图 6 给出了验证结果,errors 为 0 说明在验证过程中未发现错误.

6 结 论

实践证明,采用形式化方法和技术对软件系统进行分析、验证是构造可信软件的一条重要途径.本文采用

事件有限自动机描述 UML2.0 序列图,基于该模型并借助模型检测技术,能够在软件开发分析与设计阶段验证所建立模型的正确性,这对提高软件质量、提升软件开发效率具有重要意义.在给出单个对象自动机的基础上,使用积自动机描述系统设计,为验证软件设计模型与需求的一致性奠定了基础;同时,为下一步从 ETDFA 中抽象出软件测试用例提供了有力依据.

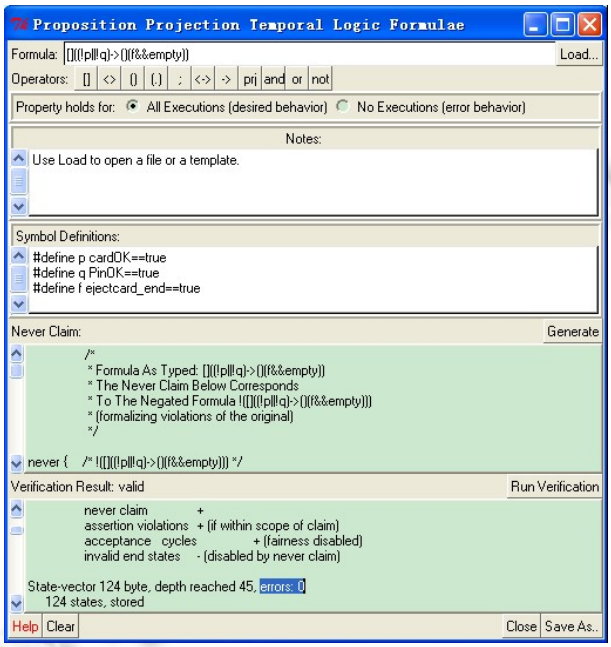


Fig.6 Verification result

图 6 验证结果

致谢 在此,向对本文工作给予支持和建议的同行,尤其是西安电子科技大学计算理论与技术研究所段振华教授领导的讨论组中的同学和老师表示衷心的感谢.

References:

[1] Lima V, Talhi C, Mouheb D, Debbabi M, Wang L, Pourzandi M. Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages. Electronic Notes in Theoretical Computer Science, 2009,254:143−160. [doi: 10.1016/ j.entcs.2009.09.064]

[2] Aredo DB. A framework for semantics of UML sequence diagrams in PVS. Journal of Universal Computer Science, 2002,8(7): 674−697. [doi: 10.3217/jucs-008-07-0674]

[3] Duan ZH. Temporal Logic and Temporal Logic Programming. Beijing: Science Press, 2005.

[4] Ammar BB, Bhiri MT, Souquières J. Incremental development of UML specifications using operation refinements. Innovations in Systems and Software Engineering, 2008,4(3):259−266. [doi: 10.1007/s11334-008-0056-1]

[5] Li JF, Chen P. The Z specification-based method for the semantic analysis of UML sequence diagrams. Journal of Xidian University (Natural Science), 2003,30(4):519−524 (in Chinese with English abstract).

[6] Zhou X, Shao ZQ. ASM semantic modeling and checking for sequence diagrams. In: Proc. of the 5th Int'l Conf. on Natural Computation (ICNC 2009). Washington: IEEE Computer Society, 2009. 527−530. [doi: 10.1109/ICNC.2009.218]

[7] Gan JH, Zhang S, Wen B. Research of modeling method based on UML2.0 and temporal logic. In: Proc. of the 1st Int'l Conf. on Information Science and Engineering (ICISE 2009). Washington: IEEE Computer Society, 2009. 5033−5036. [doi: 10.1109/ICISE. 2009.918]

[8] Tribastone M, Gilmore S. Automatic translation of UML sequence diagrams into PEPA models. In: Proc. of the 5th Int'l Conf. on Quantitative Evaluation of Systems (QEST 2008). Washington: IEEE Computer Society, 2008. 205−214. [doi: 10.1109/QEST.

2008.18]

[9] Li GY, Yao SZ. Research on mapping algorithm of UML sequence diagrams to object Petri nets. In: Proc. of the WRI Global Congress on Intelligent Systems-Volume 04 (GCIS 2009). Washington: IEEE Computer Society, 2009. 285−289. [doi: 10.1109/GCIS.2009.397]

[10] Nematzadeh H, Deris SB, Maleki H, Nematzadeh Z. Evaluating reliability of system sequence diagram using fuzzy Petri net. Int'l Journal of Recent Trends in Engineering, 2009,1(1):142−147.

[11] Esparza J. Decidability and complexity of Petri net problems—An introduction. In: Reisiq W, Rozenberg G, eds. Proc. of the Lectures on Petri Nets I: Basic Models, Advances in Petri Nets. London: Springer-Verlag, 1998. 374−428. [doi: 10.1007/3-540-65306-6_20]

[12] Knapp A, Wuttke J. Model checking of UML 2.0 interactions. In: Kühne T, ed. Proc. of the Models in Software Engineering (MoDELS) 2006 Workshops. LNCS 4364, Berlin, Heidelberg: Springer-Verlag, 2007. 42−51. [doi: 10.1007/978-3-540-69489-2_6]

[13] Harel D, Maoz S. Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling, 2008,7(2):237−252. [doi: 10.1007/s10270-007-0054-z]

[14] Harel D, Kleinbort A, Maoz S. S2A: A compiler for multi-modal UML sequence diagrams. In: Dwyer MB, Lopes A, eds. Proc. of the 10th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE 2007). LNCS 4422, Berlin, Heidelberg: Springer-Verlag, 2007. 121−124.

[15] Broy M, Jonsson B, Katoen JP, Leucker M, Pretschner A, eds. Model-Based testing of reactive systems. LNCS 3472, Berlin, Heidelberg: Springer-Verlag, 2005. 615−616.

[16] Lynch NA, Tuttle MR. An introduction to input/output automata. CWI Quarterly, 1989,2(3):219−246.

[17] Hamilton K, Miles R. Learning UML2.0. United States: O'Reilly Media, 2006.

[18] Li XS, Liu ZM, He JF. A formal semantics of UML sequence diagrams. In: Proc. of the Australian Software Engineering Conf. (ASWEC 2004). Washington: IEEE Computer Society, 2004. 168−177.

[19] Inform C. Ohlhoff CA. Consistent refinement of sequence diagrams in the UML 2.0. Hamburg: Christian- Albrechts-Universität zu Kiel, 2006. http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/aoh-dt.pdf

[20] Duan ZH, Tian C, Zhang L. A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica, 2008,45(1):43−78. [doi: 10.1007/s00236-007-0062-z]

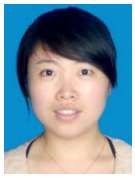
[21] Duan ZH, Tian C. A unified model checking approach with projection temporal logic. In: Liu S, Maibaum T, Araki K, eds. Proc. of the 10th Int'l Conf. on Formal Engineering Methods (ICFEM 2008). LNCS 5256, Berlin, Heidelberg: Springer-Verlag, 2008. 167−186. [doi: 10.1007/978-3-540-88194-0_12]

[22] Tian C, Duan ZH. Model checking propositional projection temporal logic based on SPIN. 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. 246−265. [doi: 10.1007/ 978-3-540-76650-6_15]

[23] Tian C, Duan ZH. Complexity of propositional projection temporal logic with star. Mathematical Structure in Computer Science, 2009,19(1):73−100. [doi: 10.1017/S096012950800738X]

附中文参考文献:

[5] 李景峰,陈平.基于 Z 规范的统一建模语言序列图语义分析方法.西安电子科技大学学报(自然科学版),2003,30(4):519−524.



张琛(1981－),女,陕西西安人,博士生,主要研究领域为软件建模,形式化方法,验证技术,高可信软件开发方法.

田聪(1981－),女,博士,副教授,主要研究领域为形式化方法,时序逻辑,模型检测.

段振华(1948－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络计算,可信软件理论和技术.