面向模型检验的 UML 状态机语义

周 颖,郑国梁,李宣东
(1. 南京大学计算机软件新技术国家重点实验室,江苏南京 210093; 2. 南京大学计算机科学技术系,江苏南京 210093)

摘 要: UML 状态机(SM)是 UML 中用来对系统各种元素的离散行为建模的图. 它丰富的表示符号提供了强大的描述机制,但也降低了其结构的模块性,提高了对其分析验证的难度. 模型检验是自动检验有限状态并发系统的技术. 通过模型检验 SM 描述的不同系统元素的行为是否满足某些性质,能尽早发现设计中的错误. 为了将模型检验技术应用于 SM 的验证,本文用 kripke 结构定义 SM 的操作语义. 与已有的 SM 语义定义不同,本文考虑到了 SM 中包含的不确定因素,用 kripke 结构描述系统所有可能的演化轨迹. 通过检验从 SM 翻译得到的 kripke 结构达到模型检验 SM 的目的.

关键词: UML; 状态机; 操作语义; Kripke 结构; 模型检验
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2003) 12A-2091-05

An Operational Semantics for UML State Machines in Model Checking Context

ZHOU Ying, ZHENG Guo-liang, LI Xuan-dong
(1. *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu* 210093, *China*;
2. *Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu* 210093, *China*)

Abstract: The state machine formalism in UML is used for modeling discrete behavior of various system elements. Its rich notation provides a powerful description mechanism which meanwhile decreases modality of its structure and increases verifying complexity. Model checking is a technique for automatically verifying finite-state concurrent systems. By model checking the behavior of various system elements described by SM, we can find design errors as soon as possible. This paper defines an operational semantics for SM using Kripke structure in order to model checking SM. Different from existing SM semantics definitions, this paper takes undetermined factors in SM into consideration . SM is translated into kripke structure that describes all possible evolving traces on which model checker checks.

Key words: UML; SM; operational semantics; Kripke structure; model checking

1 引言

UML(Unified modeling language)是用来记录、交流系统设计思想的建模语言,它用图描述系统模型的静态结构和动态行为. SM(State machine)是其中一种用来对单个实体(如类)行为或实体间交互(如合作)建模的图. SM 将系统动态行为离散化为格局及格局间的瞬时迁移. 格局类似系统的快照,是相对稳定,可见的;而迁移是格局之间的跳转,是瞬时的,不可见的. 用 SM 建模,系统演化更抽象,易于理解和控制. SM 准确直观地描述系统动态行为模型,但它并非是不言自明的,它的表示方法也限制了模型分析的途径. 为了消除 SM 模型的二义性,为了将 SM 转化为更适合分析检验的表示形式,需要定义 SM 的动态语义.

SM 是 Harel 经典状态图面向对象的变种. Harel 经典状态机是基于状态的图形规约语言,它在有限自动机的基础上增加了层次、并发和广播通信机制. 而 SM 在事件、冲突转换和进出入状态时执行动作的处理上与 Harel 状态图有所不同. 检验 SM 描述的不同系统元素的行为是否满足某些性质能尽早发现设计中的错误. 但 SM 中的组合状态,层间转换,历史转换等在丰富了描述手段的同时也破坏了结构的模块性,而且转换结果也与上下文相关. 这使得直接对 SM 模型检验非常困难. 为此,本文为 SM 定义操作语义,将 SM 翻译为等价的 Kripke 结构后进行模型检验.

Kripke 结构是模型检验中的一种模型描述语言,常被用于描述并发系统的模型. Kripke 结构中的状态描述系统的状态,状态间的关系表示系统可能从其中一个状态改变为另一个状态. 一个状态与多个状态存在关联说明处于该状态的系统的演化方向具有不确定性,例如,在并发程序下一步执行的代码可能是不确定的,系统的下一状态因此有多个可能. SM 中不确定性来源主要有两个:一是事件产生的不确定,二是不

收稿日期:2003-09-30;修回日期:2003-12-10
基金项目:863 项目(No. 2001AA113203,No. 2002AA116090);自然科学基金项目(No. 60207036);973 项目(No. 2002CB312001)

确定的随机转换的存在. 它们都在 kripke 结构中得到体现并接受模型检验.

为 SM 定义语义必须考虑状态层次的表示,并发状态的表示,当前状态的体现,层间转换的表示,冲突转换的判断,冲突优先级的定义,有优先关系的冲突的排除,无优先关系的冲突的处理,所有可能的最大可触发转换集的构造,层间转换的执行,历史转换的执行.

2 SM 的词法

文献[1]中用元模型表示图的抽象词法,即组成图的元素及其之间的关系. SM 的抽象词法包括的元素有:状态机(StateMachine),状态(State),转换(Transition),条件(Guard),事件(Event),动作序列(Action). 本文用元组表示部分概念.

(1) 状态机(StateMachine): $sm = (S, T, S_{top})$,其中 S 为状态集,T 为转换集,S_{top} 为最外层状态.

(2) 原子状态(Basic-State): $s = (n, ac, en, ex)$,其中 n 为状态名,在状态机中是唯一的,ac, en, ex 分别为处于该状态,进入,退出该状态时 SM 执行的动作序列. Type(s) = Basic.

(3) 或状态(Or-State): $s = (n, (s_1, s_2, \cdots, s_k), l, Ts, ac, en, ex)$,其中 n 为状态名; s_i 为 s 的子状态; i 为该子状态在 s 中的编号; l 是处于激活状态的子状态的编号,其默认值为 1,即 s_1 是 s 的默认激活子状态; T_s 是 s 的所有子状态间转换的集合. 对处于激活状态的或状态 s,它的子状态中有且只有一个 s_l 处于激活状态. 或状态的子状态称为是串行的. Type(s) = Or.

(4) 与状态(And-State): $s = (n, (s_1, s_2, \cdots, s_k), ac, en, ex)$,其中 n 为状态名; s_i 为 s 的子状态; i 为该子状态在 s 中的编号;对处于激活状态的与状态 s,它的所有子状态都处于激活状态. 与状态的子状态称为是并发的. Type(s) = And.

(5) 转换(Transition): $t = (n, i, sou, e, g, ac, tar, j, ht)$,其中 n 为转换名; i, j 为转换的最外层源状态、靶状态的标号; 层间转换由 sou 和 tar 描述,sou 是受限源状态(Source restriction),tar 是确定靶状态(Target determinator),分别表示 t 的确切源状态和靶状态; e 是 t 的触发事件; g 是 t 的触发条件; ac 是转换时执行的动作序列; ht 是 t 的历史类型. SM 提供历史结点(一种伪状态结点)表示进入组合状态时对其子状态的激活方式. 历史结点分为 None, Shallow, Deep 三种类型. None 指进入组合状态时激活其默认子状态; Shallow 指对进入组合状态时对直接子状态恢复最近的活动配置,而层层进入组合状态的子状态则激活其默认子状态; Deep 指进入组合状态时层层恢复最近的活动配置.

SM 中有多个状态处于激活状态,从状态的元组中是不能判断该状态是否处于激活状态的,必须通过从顶层状态出发逐层追随 1 的值来计算处于激活状态的状态集合. 如果某状态处于激活状态那么从该状态开始到它的内层状态中所有被激活的状态组成的集合称为该状态的配置(准确定义见 $conf(s)$).

SM 的当前状态(CS)有多种表示方法,如所有状态元组集 S,所有处于激活状态的原子状态组成的集合,顶层状态的配置. 用状态元组集表示的当前状态包含更丰富的信息,在存在历史转换的情形下,也只有它能准确描述系统状态. 本文默认地选第一种表示方法.

或状态 s 状态元组中的 T_s 是 s 子状态间转换的集合,因此 T_s 中所有转换的最小公共祖先(Least common ancestor)都是 s.

图 1 是 SM 的一个例子.

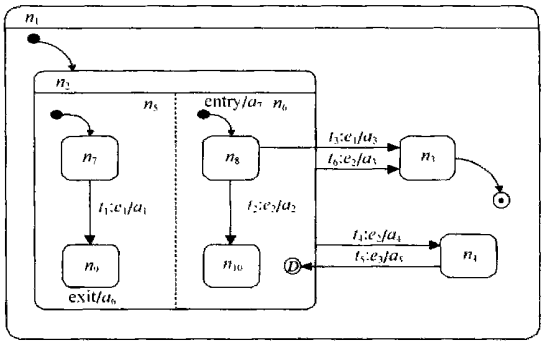

图 1 UML 状态机

图 1 中各状态和转换对应的元组表示如下:

$s_1 = (n_1, (s_2, s_3, s_4), 1, (t_3, t_4, t_5, t_6), \diamond, \diamond, \diamond)$

$s_2 = (n_2, (s_5, s_6), \diamond, \diamond, \diamond)$

$s_5 = (n_5, (s_7, s_9), (t_1), \diamond, \diamond, \langle a_6 \rangle)$

$s_6 = (n_6, (s_8, s_{10}), (t_2), \langle a_7 \rangle, \diamond, \diamond)$

$s_i = (n_i, \diamond, \diamond, \diamond), i = 3, 4, 7, 8, 9, 10$

$t_1 = (t_1, 7, (), e_1, \wedge, \langle a_1 \rangle, (), 9, \text{None})$

$t_2 = (t_2, 8, (), e_2, \wedge, \langle a_2 \rangle, (), 10, \text{None})$

$t_3 = (t_3, 2, (s_5, s_6, s_8), e_1, \wedge, \langle a_3 \rangle, (), 3, \text{None})$

$t_4 = (t_4, 2, (), e_2, \wedge, \langle a_4 \rangle, (), 4, \text{None})$

$t_5 = (t_5, 4, (), e_3, \wedge, \langle a_5 \rangle, (), 2, \text{Deep})$

$t_6 = (t_6, 2, (), e_2, \wedge, \langle a_3 \rangle, (), 3, \text{None})$

3 SM 的静态语义

UML 静态语义定义结构的实例如何与其他实例联系以确保其具有意义[1],即抽象词法中元素应遵循的良形式规则(Wellformedness rules). 在元模型中静态语义用 OCL 描述. SM 中同时有多个状态处于激活状态,但并不是任意状态子集都能同时激活. 要保证 SM 有意义,必须遵守一组良形式规则,这组规则限制了合法的状态激活,也为算法奠定了前提. 关于组合状态的良好形式规则有:组合状态最多有一个 Deep 和 Shallow 历史结点;并发组合状态至少有两个组合子状态;并发组合状态的子状态只能是组合状态,组合状态的子状态只能是该组合状态的一部分[1].

除去转换执行时刻,状态配置总满足下面的条件:(1)如果一个组合状态是活动的并且是非并发的,那么它的直接子状态只有一个是活动的;(2)如果组合状态是活动的并且是并发的,它所有的子状态区都是活动的. 上述条件在或状态和与状态的元组结构中已经隐含,因为或状态中只用标号 1 指示

激活的子状态,与状态中则没有指示激活子状态的标号.

4　Kripke 结构

Kripke 结构的形式定义为 $K=(S,S_0,R,L)$,其中 $S=\{\langle s,e\rangle \mid s\in S(SM),e\in E(SM)\}$,$S_0=\langle \text{defconf}(S_{top}(SM)),0\rangle$,$R(s,s')=(s=\langle s_1,0\rangle \wedge s'=\langle s_1,e\rangle \wedge e\in PES(s_1)) \vee s=\langle s_1,e\rangle \wedge s'=\langle \text{NextState}(s_1,T),0\rangle \wedge T\in \text{SETS}(s_1,e))$,$L=S(SM)\rightarrow 2^{AP}$,$AP$ 是状态相关的性质集.

Kripke 结构也可图形化表示,用椭圆表示状态,用状态间的连线表示状态间存在关系.

状态表示系统在某个时刻可观察的状况,状态间的连线表示状态的演化.从同一状态引出的不同连线表示程序演化的不确定性,即演化有多种可能性.

Kripke 结构的状态与状态机的状态有所不同:前者更全面、严格、机械,把确定的因素都包含在状态向量中;而后者更偏重于建模的抽象考虑,加上了层次,而事件、条件往往在状态的视野之外.

UML 状态机中的所谓并发状态中的转换并不引入不确定性,因为依据 UML 文档给出的状态机语义,并发状态中具备触发条件的转换将在同一个 RTC 步骤中完成,即在 kripke 结构中的一条连线上完成.UML 状态机中的不确定性有两个来源:一是事件的发生,它是不受状态机执行的控制的;二是同时使能的若干不确定转换,一个 RTC 步骤只能随机选择其中一个执行.UML 状态机的这两种不确定性在 kripke 结构中都反映为从同一状态引出多条连线通向多个状态,意味这系统从某一状态不确定地演化到若干候选状态之一.本文认为这种转换的不确定性也应该在模型检验的考虑范围之内,因此在 kripke 结构中应有所反映,所以本文中并非只考虑随机选择的一个最大无冲突使能转换集中转换的执行,所有的不确定转换的结果都必须计算,在 kripke 结构中构造相应的候选状态.

在本文的 kripke 结构中略去了对当前状态无影响的事件的处理.如果需要考虑,在 kripke 中加入描述其余事件发生的状态即可.

5　SM 的操作语义

用 kripke 结构定义 SM 语义的时候,二者的抽象词法并非是一一对应的.Kripke 结构的状态对应于 SM 的状态和事件.kripke 结构中没有显式对应于 SM 转换的元素,SM 中转换的执行结果体现在 kripke 结构中状态间的关联上.

这与语义定义的目的有关,kripke 结构是用来模型检验的,关心的是系统所有可能的演化轨迹.依据文[1]中 SM 的 RTC(Run To Completion)语义,RTC 具体地说就是处于某个状态的 SM 对分发的事件,将它能触发的最大的优先级最高的互不冲突的转换合成一个转换集,SM 一一触发其中的转换,完成相应的动作序列后,才能称为完成一个 RTC 广义步.可见,系统演化是从 SM 对事件的整体反应上来考虑的,对单个转换以及转换的执行顺序并不关心.

下面给出将 UML 状态机翻译为 kripke 结构的方法,其中涉及的未定义算法在下节定义.

令 UML 状态机的当前状态等于初始状态,即默认状态:$CS=\text{defconf}(S_{top})$.

SM2KRIPKE(CS)定义如下:

(1)对 UML 状态机的当前状态 CS,如果在 kripke 结构中已存在状态$(CS,0)$,则退出;否则构造对应的 kripke 结构的状态$(CS,0)$,计算其可能事件集 $PES(CS)$.对 PES 中的每个事件 e,构造对应的 kripke 状态(CS,e),连接$(CS,0)$与(CS,e).

(2)对 UML 状态机的当前状态 CS,对 PES 中的每个事件 e,计算其所有可能的最大可触发转换集的集合 SETS.依据 RTC 语义,对 SETS 中的每个转换集 ETS:

①执行 ETS 中的每个转换,到达状态机的下一状态 $NS=\text{NextState}(CS,ETS)$.

②调用 SM2KRIPKE(NS).

③连接(CS,e)与$(NS,0)$.

通过这个语义定义,SM 被翻译为 kripke 结构.Kripke 结构描述系统所有可能的演化.图 2 所示是图 1 状态机对应的 Kripke 结构的一部分.

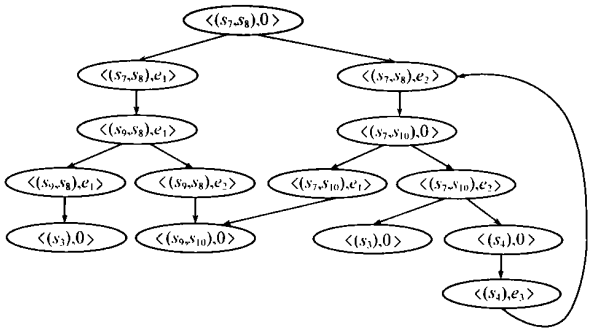

图 2　kripke 结构

图 2 中为简单起见用激活的原子状态集表示当前状态,其实是不对的,在历史转换时会产生不确定的下一状态,在 kripke 结构中引入不可能发生的系统演化轨迹.实际实现中必须采用状态元组集的表示方法才能准确描述系统的当前状态,正确处理历史转换.

6　相关概念和算法

(1)状态的完全配置

$conf((n,ac,en,ex))=\{n\}$

$conf((n,(s_1,s_2,\cdots,s_k),l,T,ac,en,ex)=\{n\}\cup conf(s_l)$

$conf((n,(s_1,s_2,\cdots,s_k),ac,en,ex))=\{n\}\cup\bigcup_{i=1}^{k} conf(s_i)$

(2)状态的所有配置

$confs\leq((n,ac,en,ex))=\{\{n\}\}$

$confs\leq((n,(s_1,s_2,\cdots,s_k),l,T,ac,en,ex)=\{\{n\}\cup confi \mid confi\in confs(s_l)\}\cup\{\{n\}\}$

$confs\leq((n,(s_1,s_2,\cdots,s_k),ac,en,ex))=\{\{n\}\cup\bigcup_{i=1}^{k} confi \mid confi\in confs(s_i)\}\cup\{\{n\}\}$

(3)状态的默认配置

$defconf((n,ac,en,ex))=\{n\}$

$defconf((n,(s_1,s_2,\cdots,s_k),l,T,ac,en,ex)=\{n\}\cup defconf(s_l)$

$defconf((n,(s_1,s_2,\cdots,s_k),ac,en,ex))=\{n\}\cup\bigcup_{i=1}^{k}defconf(s_i)$

(4) 可能事件集(Possible Event Set)

$PES(CS)=\{e(t)|\ sou(t)\subseteq conf(s_{top})\wedge g(t)\}$

(5) 最大使能转换集(Enabled Transition Set)

$ETS(CS,e)=\{t|\ sou(t)\subseteq conf(s_{top})\wedge e(t)=e\wedge g(t)\}$

(6) 最小公共祖先(Least Common Ancestor)

$$\frac{s=(n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex)}{LCA(s_1)=LCA(s_2)=\cdots=LCA(s_k)=s}$$

(7) 冲突转换(Conflicting Transition)

$conflict(t_1,t_2)\Leftrightarrow conf(LCA(t_1))\cap conf(LCA(t_2))\neq\phi$

换言之,当转换 t_1 与 t_2 的退出状态集交集非空时则称 t_1 与 t_2 冲突. 例如,图 1 状态机中,t_1 与 t_3,t_2 与 t_4 是冲突的.

(8) 子状态

$sub((n,ac,en,ex))=\phi$

$sub((n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex))=\{s_1,s_2,\cdots,s_k\}$

$sub((n,(s_1,s_2,\cdots,s_k),ac,en,ex))=\{s_1,s_2,\cdots,s_k\}$

(9) 配置最内层状态(Deepest State)

$ds(conf)=\{s|\ s\in conf\wedge sub(s)\cap conf=\phi\}$

(10) 转换的优先关系

$t\prec t'\Leftrightarrow ds(sou(t))\subset conf(ds(sou(t')))$

冲突的转换中,受限源状态层次较深的转换优先级较高. 图 1 状态机中,$t_1\prec t_3$,$t_2\prec t_4$.

(11) 可触发转换集(Firing Transition Set)

$FTS(CS,e)=\{t|\ \forall t'\in ETS(CS,e),t'=t\vee conflict(t,t')=\text{False}\vee t\prec t'\}$

某些转换是冲突的却无优先关系. 在一个 RTC 步骤中只能选择它们中的一个,这个选择是随机的,不确定的. 由于不确定转换的存在,当前状态和当前事件对应的可触发转换集也是不确定的. 每种可能发生的情况都是模型检验应该考虑的,因此为每个可能的转换选择构造一个 FTS,构成可触发转换集的集合(Set of Firing Transition Set). 构造 SFTS 的方法的主要思路是构造优先关系链,取链首元素组成候选转换集 T,构造 T 的冲突关系图 G,求其补图 G',G 的每个最大强连通子图的结点集对应一个可能的无冲突的优先级最高的转换集.

优先级关系构成一个偏序,也是若干独立的偏序链. 同一偏序链中的转换优先级从高到低排列,转换的源状态也是依次嵌套的. 每条链中优先级最高的转换互相没有嵌套关系,它们或者不冲突,或者冲突而不存在优先关系. 选取每条偏序链的优先级最高的转换构成的转换集 T 中剔除了造成冲突且优先集较低的转换,即不可能触发的转换,余下的转换至少在一个最大可触发转换集中出现.

转换集 T 中的转换和转换间的冲突关系可以表示为一张图 G:把转换作为结点,两个转换对应的结点间有边当且仅当这两个转换冲突. 对图 G 求其补图 G'. G 的一个最大强连通子图对应 T 的一个最大无冲突转换集. G 的所有最大强连通子图对应 T 的所有可能的最大无冲突转换集. "连通"保证选取的转换之间是无冲突的;"最大子图"保证所有可并发触发的转换都已被选. 形式表示如下.

(12) 可能的最大无冲突可触发转换集的集合

$SFTS(CS,e)=\{NCT(CS,e)\cup\{t(v)|\ v\in V(g_i)\}\}$,$g_i\in CSGS(G')$,其中 $CSGS(G')$ 为 G' 的最大强连通子图集.

(13) 状态对单个转换的下一状态

$$\frac{s=(n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex),t\in T_s}{\text{NextState}(s,t)=(n,(s_1,s_2,\cdots,s_k),lab(s_{j(t)}),T_s,ac,en,ex)}$$

其中 $lab(s)$ 是状态 s 在其父状态中的编号.

$$\frac{s=(n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex)\in tar(t),s_{j(t)}\in sub(s)\cap tar(t)}{\text{NextState}(s,t)=(n,(s_1,s_2,\cdots,s_k),lab(s_{j(t)}),T_s,ac,en,ex)}$$

下面的规则与 $ht(t)$ 取值有关.

$$\frac{ht(t)=\text{None},s=(n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex),s\in defconf(ds(tar(t)))}{\text{NextState}(s,t)=(n,(s_1,s_2,\cdots,s_k),1,T_s,ac,en,ex)}$$

$$\frac{ht(t)=\text{Shallow},s=(n,(s_1,s_2,\cdots,s_k),l,T_s,ac,en,ex),s\in defconf(s_{1(ds(tar(t)))})}{\text{NextState}(s,t)=(n,(s_1,s_2,\cdots,s_k),1,T_s,ac,en,ex)}$$

history(t) = Deep 时,其余状态保持不变.

(14) 状态对转换集的下一状态

NextState(S,T) 是 T 中所有转换迭加作用到当前状态的结果.

7 相关工作

关于 SM 语义,已有一些出于不同目的采用不同手段进行的全面或部分定义.

Johan Lilius 等人在文[2]中,用表结构表示状态,状态层次没有独立表示,引入变量处理关于组合状态的转换,用 term 描述状态匹配,计算 exit 集判断冲突状态. 他们定义 SM 语义的目的是 Model Checking,讨论了 UML 文档中关于 SM 的几乎所有概念,但有些并未进行形式化定义,如历史转换.

文[5]的作者用元组表示状态和转换,用一组 SOS 规则定义 UML 状态图的操作语义,将其翻译为 LTS. 本文中状态和转换的表示均沿袭[5].

文[6]中,Sabine Kuske 用图转换技术定义 SM 语义,用图转换规则表示转换,用图转换单元实现生成状态层次,状态配置,构造 fire 集等算法. 但未对历史转换等复杂转换进行定义. 图转换是基于图形的技术,它在为 SM 语义定义提供丰富理论和工具支持的同时,也限制了它的使用.

李等在文[7]中,用 Kripke 结构描述 SM 的操作语义,在抽象机(该文称虚拟机)定义中将事件队列分为内部、完成、外部事件队列,并应用该语义构造对象 SM 的测试用例集.

所有这些 UML 状态机的语义定义,包括那些同样以模型检验为目的的工作,都忽略了对状态机中不确定性因素的处理,只从冲突而无优先关系的转换中随机地选择一个加入 ETS,这样得到的模型是不能满足模型检验要求的. 本文的语义定义考虑了 SM 中的不确定因素,翻译得到的 kripke 结构描述系统所有可能的演化轨迹. 模型检验可基于 kripke 结构描述的系统模型进行.

8 结束语

UML 语义文档中用自然语言指定 SM 的动态语义. 为 SM 提供形式语义不仅能提高描述精确性,减少二义性和不一致性,也是模型执行、程序自动生成、模型检验等工作的重要基础.

本文用 kripke 结构定义 SM 的操作语义. 通过检验从 SM 翻译得到的 kripke 结构达到模型检验 SM 的目的. 本文对动作序列运行结果没有涉及,只适用模型检验状态中隐含的属性. 进一步的工作可考虑针对动作序列进行语义扩展.

参考文献:

[1] OMG Unified Modeling Language Specification, Version 1.3 [S]. available at http://www.rational.com/uml/resources/documentation/index.jtmpl, 1999-06.

[2] Johan Lilius, Iván Porres. Formalizing UML state machines for model checking [A]. UML '99, LNCS1723 [C]. Springer-Verlag Heidelberg, 1999. 430-445.

[3] Stuard Kent, Andy Evans, Bernhard Rumpe. UML semantics FAQ [A]. ECOOP '99 Workshops, LNCS1743 [C]. Springer-Verlag Heidelberg, 1999. 33-56.

[4] Peter Padawitz. Swinging UML: How to make class digrams and state machines amenable to constraint solving and proving [A]. UML2000, LNCS1939 [C]. Springer-Verlag Heidelberg, 2000. 162-177.

[5] Michael von der Beeck. Formalization of UML statecharts [A]. UML2001, LNCS2185 [C]. Springer-Verlag Heidelberg, 2001. 406-421.

[6] Sabine Kuske. A formal semantics of UML state machines based on structured graph transformation [A]. UML 2001, LNCS 2185 [C]. Springer-Verlag Heidelberg, 2001. 241-256.

[7] 李留英,王戟,齐治昌. UML Statechart 图的操作语义 [J]. 软件学报, 2001, 12(12): 1864-1873.

作者简介:

周　颖　女,1974 年生于江苏泰州,博士生,主要研究领域为软件工程,形式化方法.

郑国梁　男,1937 年生于浙江桐乡,教授,博士生导师,主要研究领域为软件工程,软件开发环境.

李宣东　男,1963 年生于湖南邵东,教授,博士生导师,主要研究领域为面向对象技术,形式化方法,软件工程.