UML状态机模型元素的RSL形式化定义

郭艳燕 刘惊雷

(烟台大学计算机学院 烟台 264005)

摘 要 UML状态机作为UML动态描述机制的重要组成部分,在描述系统及模型的动态行为时扮演着重要的角色,但已有的UML动态语义缺乏准确的形式化描述。首先将UML状态机抽象成图;再将图通过传统的有穷自动机进行语义扩展,同时增加状态分层,形成一个基于UML状态机的有穷自动机;然后用RAISE规约语言RSL对扩展后的自动机进行形式化定义,使UML状态机中的模型元素的语义更加清晰、精确,为后期的UML状态机的操作语义形式化研究打下基础。

关键词 统一建模语言(UML),状态机,形式化方法,有穷自动机,RAISE规约语言(RSL)

中图法分类号 TP301.2 **文献标识码** A

Formalization for Model Element of UML Statechart in RSL

GUO Yan-yan LIU Jing-lei

(School of Computer Science and Technology, Yantai University, Yantai 264005, China)

Abstract UML statechart plays an important role in describing the dynamic behavior of system and model as an important part of UML dynamic description mechanism. Existing dynamic semantics of UML are lack of accurate formal description. UML Statechart was defined as the abstract syntax graphs, which were expanded into a new finite automaton based on UML statechart through the traditional finite automaton and increased state hierarchy. Then, this paper formalized the model elements of UML statechart through RAISE specification language(RSL). The formal semantic of the model elements of UML statechart is more clear and accurate, which is the base of the later operation semantic study of UML statechart.

Keywords Unified modeling language(UML), Statechart, Formal method, Finite automata, RAISE specification language(RSL)

统一建模语言(Unified Modeling Language, UML)[1]是一种可视化的面向对象建模语言,已经成为了OMG的标准。UML提供了多种建模机制,从不同角度和应用层次刻画系统的特性和复杂的运行环境,以及从静态和动态两个方面来为软件系统建模。

目前UML的语义仍是半形式化的,它使用元模型和较为形式化的对象约束语言(Object Constraint Language, OCL)描述静态语义[2],而动态语义基本上完全用自然语言来描述。采用这种方法描述的动态语义存在着不完全、不一致、模糊性等缺陷,难以支持对复杂系统的模型进行严格的语义分析和正确性验证。

UML状态机作为UML动态描述机制的重要组成部分,在描述系统及模型的动态行为时扮演着重要的角色。为UML状态机构造形式化的语义不仅有利于准确而无二义性地理解其所表示对象的行为,而且有利于系统的代码生成及优化,同时更有助于对系统的正确性和安全性进行形式化验证和证明[3],以及软件设计模型间的不一致性的自动检测与跟踪[4]。

具有面向对象特征的RAISE的规约语言RSL(RAISE Specification Language, RSL)[5]是混合的面向性质和模型的规约语言,具有较复杂的面向对象的复合方式,并且提供表达并发的方式。RSL既可以用于书写非常抽象的初级的规范,也可以用于书写易于甚至能自动转换成程序语言的更具体的规范,同时具有较强的不同抽象层次的表述。既然UML可以为软件系统开发的不同阶段、不同抽象层次和不同描述角度进行建模,那么用RSL对UML模型的语义进行形式化就非常适合。

本文的主要工作是采用RSL对UML的状态机的模型元素进行形式化定义。在进行UML状态机模型元素的形式化语义之前,先将UML状态机抽象成图,再将图通过传统的有穷自动机进行语义扩展,产生一种新的基于UML状态机的新型有穷自动机,然后使用RSL进行形式化定义。本文主要对UML状态机的模型元素以及分层状态结构进行了形式化定义,对状态机的语义操作的形式化将作为今后进一步研究的内容。本文在定义语法与语义时借鉴了文献[3,6,10,12,13,15,16]的一些基本思想。

到稿日期:2012-07-19 返修日期:2012-10-17 本文受国家自然科学基金(61170224),山东省自然科学基金(ZR2011FL018),山东高等学校科研计划项目(J10LG27)资助。

郭艳燕(1980—),女,研究生,讲师,主要研究方向为软件工程、理论计算机科学,E-mail:smallgyy@sina.com;刘惊雷(1970—),男,博士,副教授,主要研究方向为人工智能、理论计算机科学。

本文第1节介绍UML状态机形式化定义的相关研究；第2节介绍基于UML状态机的有穷自动机的构建及其形式化定义；第3节将对UML状态机的模型元素进行形式化定义；最后总结本文，并对进一步的工作进行讨论。

1 UML状态机形式化定义的相关研究

从形式化技巧角度来看，UML的形式化方法[6]大致可以分为以下两大类。

(1)直接为UML模型定义形式化的语义，在此基础上对模型进行语义分析和正确性验证。文献[3]通过把UML状态机中的状态映射到一种项代数上，然后把状态项映射到一种加标记的变迁系统LTS上，用Plotkin风格的结构操作语义规则归纳地给出满足组合性的UML状态机语义的形式化方法。文献[7]采用基于Petri网的UML状态图的形式化方法，提出了一种可以准确描述UML状态图动态特征的形式化模型。文献[8]将UML状态图的操作映射到一个特定形式的自动机，使用基于自动机理论的模型检验方法来验证UML Statecharts的线性时态逻辑性质。

(2)结合UML和形式化方法的优点，将非形式化的UML图形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。文献[9]提出在基本迁移系统上建立图形建模语言UML状态机与线性时序逻辑语言XYZ/E的语义联系，用XYZ/E公式表示基本迁移系统的时序语义，从而给出了相应状态机的时序语义。文献[10]提出将UML状态机通过GTDL(Graph Type Definition Language)转换成属性图表，再将属性图表通过对象映射自动机OMA(Object Mapping Automata)映射成形式化语义描述。文献[11]用一阶逻辑的形式化方式定义UML状态图的描述语义。文献[13]使用RAISE规范语言RSL给出了UML状态机视图的形式描述，建立在对UML类图的RSL形式化基础上。文献[14]采用时序描述逻辑方式来形式化UML状态图。

本文对UML状态机的形式化研究主要采用第二种形式化方法，与文献[13]一样，都是使用RSL对UML状态机进行形式化描述。文献[13]的形式化工作是建立在UML类图的RSL形式化基础上；本文的形式化工作是建立在对有穷自动机的UML扩展上，并且关注的重点是对UML状态机中的模型元素的形式化定义，因此对模型元素的种类划分更加细致，尤其是状态分层和转换类型，而这些并未出现在文献[13]的研究工作中。在对状态类型和转换类型的形式化定义上，文献[10]将对象状态和伪状态分别进行形式化定义，并且只是简单地将choice伪状态看作junction连接伪状态来实现静态条件分支。本文在状态和转换类型的分类上更加细化和精确，将伪状态融入到对象状态中进行统一的形式化定义，并将junction连接伪状态划分为choice选择伪状态和junction连接伪状态，来实现动态条件分支，使转换的形式化更加具体和准确。

2 UML状态机的形式化语法

UML状态机描述了一个对象或一个交互在生命期内响应事件所经历的状态序列。一个完整的状态机具有的模型元素有状态、转换、事件、监护条件和动作，如图1所示。UML状态机扩展了传统的有穷自动机，增加了分层状态结构以及规约并发和通信。

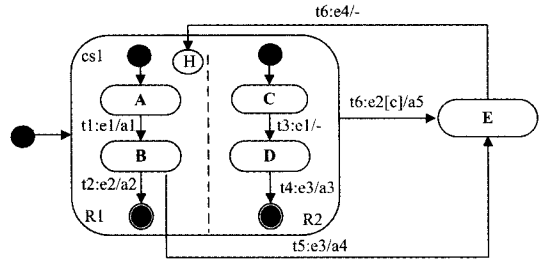

图1 UML状态机图

本文在对UML状态机的模型元素进行形式化定义前，首先对UML状态机的抽象语法进行形式化定义，并对状态机中的模型元素以及连通性进行描述。状态机集中于状态和转换，以及转换上的标记。将UML状态机的抽象语法定义成一个图：图中的顶点(vertice)对应UML状态机中的状态；图中的边(edge)对应状态之间的连线，体现状态之间的连通性；边标记(labs)是触发状态转换的事件(event)、监护条件(condition)和动作(action)的笛卡尔积。再将该图通过传统的有穷自动机进行扩展，同时增加了状态分层，形成了一个基于UML状态机的新型有穷自动机。

定义1 基于UML状态机的有穷自动机形式化为10元组

$Statechart=(VERTICE, EDGE, v\,0, EVENT, CONDITION, ACTION, LABS, SH, \delta_1, \delta_2)$

其中$VERTICE$，$EDGE$，$EVENT$，$CONDITION$和$ACTION$都是有穷集合，并且

1)$VERTICE$是顶点集，对应UML状态机中的状态集。

2)$EDGE$是连接顶点的边集。

3)$v\,0 \in VERTICE$是起始顶点(状态)。

4)$EVENT$是边上属性事件集。

5)$CONDITION$是边上属性监护条件集。

6)$ACTION$是边上属性动作集。

7)$LABS: EVENT \times CONDITION \times ACTION$是边标记集。

8)SH是状态层次结构集。

9)$\delta_1: VERTICE \times EVENT \rightarrow VERTICE$是事件转换函数。

10)$\delta_2: VERTICE \times LABS \rightarrow VERTICE$是标记转换函数。

为了下一步对UML状态机的模型元素进行形式化定义，定义10个以边集合为定义域的函数，如表1所列。

表1 函数描述表

函数名	定义域	值域	函数功能描述
src	EDGE	VERTICE	获取源状态
dst	EDGE	VERTICE	获取目标状态
is_hasEvent	EDGE	Bool	判断边上是否存在触发事件
get_Event	EDGE	EVENT	获取事件
is_hasCondition	EDGE	Bool	判断边上是否存在监护条件
get_Condition	EDGE	CONDITION	获取监护条件
is_hasAction	EDGE	Bool	判断边上是否存在动作
get_Action	EDGE	ACTION	获取动作
is_hasLabs	EDGE	Bool	判断边上是否存在标记
get_Labs	EDGE	LABS	获取边标记

定义2 基于UML状态机的新型有穷自动机用RSL的

形式化定义。

```
Scheme Statechart=
  class
    type
      VERTICE,
      EDGE,
      EVENT,
      CONDITION,
      ACTION,
      LABS= EVENT×CONDITION×ACTION,
      SH
    value
      δ1:VERTICE× EVENT → VERTICE,
      δ2:VERTICE × LABS → VERTICE,
      src:EDGE → VERTICE,
      dst:EDGE → VERTICE,
      is_hasEvent:EDGE→ Bool,
      get_Event:EDGE → EVENT
        pre get_Event(t):is_hasEvent(t),
      is_hasCondition:EDGE→ Bool,
      get_Condition:EDGE → CONDITION
        pre get_Condition(t):is_hasCondition(t),
      is_hasAction:EDGE→ Bool,
      get_Action:EDGE → ACTION
        pre get_Action(t):is_hasAction(t),
      is_hasLabs:EDGE→ Bool,
      get_Labs:EDGE→ LABS
        pre get_Labs(t):is_hasLabs(t)
      end
```

3 UML 状态机模型元素的形式化定义

本文主要对 UML 状态机的模型元素进行形式化定义。以下对模型元素的形式化定义都是建立在第 2 节中介绍的基于 UML 状态机的有穷自动机的构建及其形式化定义上。

3.1 状态

状态(state)是指在对象的生命期中的一个条件或状况，在此期间对象将满足某些条件、执行某些活动或等待某些事件。

3.1.1 状态分类

(1)UML 的状态包括简单状态(simple)、初始状态(initial)、终止状态(final)、复合状态(compose)(顺序复合状态和并发复合状态)和伪状态(pseudo)。为了形式化定义更加准确，增加根状态(root)来表示状态机初始状态的父状态，即状态机中所有状态的顶层父状态。表 2 为本文中对状态的具体分类符号和其对应的类型集合的描述。

表 2 状态分类描述表

状态分类(集合)	状态类型描述
S_s	简单状态
S_i	初始状态
S_f	终止状态
S_c	复合状态
S_{cc}	并发复合状态
S_{cs}	顺序复合状态
S_p	伪状态
S	所有状态

状态分类集合满足以下规则：

①$S_i=\{v0\}$

②$S_i\subseteq S_p$，$S_f\subseteq S_p$

③$S_c=(S_{cc}\cup S_{cs})\wedge(S_{cc}\cap S_{cs}=\phi)$

④$S=S_s\cup S_c\cup S_p\cup\{\text{root}\}$，即 S 是非空有穷状态集

(2)伪状态是在 UML 状态机中具有状态的形式，而其行为却不同于完整状态的顶点。伪状态可以具体分为初始伪状态(即初始状态)、终止伪状态(即终止状态)、历史伪状态(history)(浅历史伪状态和深历史伪状态)、转换连接伪状态(包括表示并发开始的分叉 fork 伪状态、表示并发结束的汇合 join 伪状态、表示选择开始 choice 的分支伪状态和表示选择结束的连接 junction 伪状态)。

伪状态用来连接转换段，到一个伪状态的转换意味着会有一个状态到另一个状态的自动转换，而不需要事件触发。表 3 是伪状态的具体分类符号和其对应的类型集合的描述。

表 3 伪状态分类描述表

伪状态分类(集合)	伪状态类型描述
S_{pi}	初始伪状态
S_{pf}	终止伪状态
S_{ph}	历史伪状态
S_{psh}	浅历史伪状态
S_{pdh}	深历史伪状态
S_{pl}	转换连接伪状态
S_{plf}	分叉(fork)伪状态
S_{plj}	汇合(join)伪状态
S_{plc}	分支(choice)伪状态
S_{plu}	连接(junction)伪状态

伪状态分类集合满足以下规则：

①$S_{pi}=\{v0\}$

②$S_i=S_{pi}$，$S_f=S_{pf}$

③$S_{ph}=S_{psh}\cup S_{pdh}$

④$S_{pl}=S_{plf}\cup S_{plj}\cup S_{plc}\cup S_{plu}$

⑤$S_p=S_{pi}\cup S_{pf}\cup S_{ph}\cup S_{pl}$

表 4 是与状态类型相关的函数描述表。

表 4 与状态类型相关的函数描述表

函数名	定义域	值域	函数功能描述
get_Statetype	S	State_type	获取当前状态的类型
is_Initialstate	S	Bool	判断当前状态是否是初始状态
is_Simplestate	S	Bool	判断当前状态是否是简单状态
is_Composestate	S	Bool	判断当前状态是否是复合状态
get_ComposetateType	S_c	Composestate_type	获取当前复合状态的类型
is_Pseudostate	S	Bool	判断当前状态是否是伪状态
get_PseudostateType	S_p	Pseudostate_type	获取当前伪状态的类型
is_HPseudostate	S	Bool	判断当前状态是否是历史伪状态
get_HPseudostateType	S_{ph}	Historystate_type	获取当前历史伪状态的类型
is_LPseudostate	S	Bool	判断当前状态是否是转换连接伪状态
get_LPseudostateType	S_{pl}	LPseudostate_type	获取当前转换连接伪状态的类型

定义 3 UML 状态机的状态类型用 RSL 形式化定义为：

```
Scheme StateType=
extend Statechart with
  class
    type
      S= Ss∪Sc∪Sp∪{root},
      Sc= Scc∪Scs,
      Sp= Spi∪Spf∪Sph∪Spl,
      Sph= Spsh∪Spdh,
      Spl= Splf∪Splj∪Splc∪Splu,
      Si= Spi,Sf= Spf,
      Spi:VERTICE,Spf:VERTICE,Spsh:VERTICE,
      Spdh:VERTICE,Splf:VERTICE,Splj:VERTICE,
      Splc:VERTICE,Splu:VERTICE,Scc:VERTICE,
      Scs:VERTICE,Ss:VERTICE,
      State_type==simple|initial|final|compose|pseudo
      Composestate_type==and|or,
      Pseudostate_type==initial|final|history|Historystate_type|
                        LPseudostate_type,
      Historystate_type==shallow_history|deep_history
      LPseudostate_type== fork|join|choice|junction
    value
      get_Statetype:S→State_type,
      is_Initialstate:S→Bool,
      is_Simplestate:S→Bool,
      is_Composestate:S→Bool,
      get_ComposetateType:Sc→Composestate_type,
      is_Pseudostate:S→Bool,
      get_PseudostateType:Sp→Pseudostate_type,
      is_HPseudostate:S→Bool,
      get_HPseudostateType:Sph→Historystate_type,
      is_LPseudostate:S→Bool,
      get_LPseudostateType:Spl→LPseudostate_type
end
```

(3)对于 UML 状态机中某一时刻,对象处于状态 s 时,称 s 此时为活动状态。活动状态在 UML 状态机的操作语义中起到非常大的作用,因此在状态的形式化定义中,需要加入状态当前是否是活动状态的判断。由于复合状态存在,当复合状态是活动状态时,需要获取复合活动状态的活动子状态集,为后期的操作语义研究打下基础。与活动状态有关的函数如表 5 所列。

表 5 与活动状态相关的函数描述表

函数名	定义域	值域	函数功能描述
is_Activestate	S	Bool	判断当前状态是否是活动状态
is_inActivestate	S	Bool	判断当前状态是否是非活动状态
is_stateEqual	S×S	Bool	判断两个状态是否相等,即通过状态标示符进行判断
get_Activestates	Sc	S 的幂集	获取复合活动状态的活动子状态集

3.1.2 事件

事件(event)是对一个在时间和空间上占有一定位置的、有意义的、事情的规格说明。在 UML 状态机的语境中,事件触发一个状态转换。

事件可以是内部事件或外部事件。将部事件触发状态的转换。内部事件是在状态内部发生,可以产生一些动作并执行内部转换,但不触发当前状态的转换。在状态图中,主要关心能激发状态转换的外部事件。将不能触发当前转换的事件,并且留到下一个状态来处理的事件称为延迟事件。

与事件有关的函数描述如表 6 所列。

表 6 与事件相关的函数描述表

函数名	定义域	值域	函数功能描述
is_outerEvent	STATE×EVENT	Bool	判断在活动状态下发生的事件是否是外部事件
is_deferEvent	STATE×EVENT	Bool	判断在活动状态下发生的事件是否是延迟事件

定义 4 UML 状态机的事件用 RSL 形式化定义为:

```
Event=
  extend Statechart with
    class
      type
        STATE:S,
        EVENT,
        EVENT_ID:Text,
        Event_type==outer_Event|inner_Event|defer_Event
            value
              is_outerEvent:STATE×EVENT→Bool,
              is_deferEvent:STATE×EVENT→Bool
      end
```

3.1.3 动作

按照动作在状态内执行还是在进行状态转移时执行将其分为状态动作(state action)和转移动作(transfer action)。状态动作又分为入口动作(entry action)、出口动作(exit action)和内部活动(do action)。

当进入状态时,入口动作先发生,然后执行状态的内部活动;当触发状态改变的事件发生时,执行状态的出口动作后,执行状态转移。内部活动发生在状态内部,由内部事件触发,不发生状态的转移。

与动作有关的函数描述如表 7 所列。

表 7 与动作相关的函数描述表

函数名	定义域	值域	函数功能描述
is_stateAction	STATE×ACTION	Bool	判断是否是状态动作
is_entryAction	STATE×ACTION	Bool	判断是否是状态的入口动作
is_exitAction	STATE×ACTION	Bool	判断是否是状态的出口动作
is_doAction	STATE×ACTION	Bool	判断是否是状态的内部活动

定义 5 UML 状态机的动作用 RSL 形式化定义为:

```
Action=
extend Statechart with
  class
    type
      STATE:S,
      ACTION,
      Action_ID:Text,
      Action_type== StateAction_type |TransAction,
      StateAction_type==en_Action|ex_Action|do_Action,
    value
      is_stateAction:STATE×ACTION→Bool,
      is_entryAction:STATE×ACTION→Bool,
      is_exitAction:STATE×ACTION→Bool,
      is_doAction:STATE×ACTION→Bool
end
```

3.1.4 状态描述

有了状态类型、事件、动作的形式化定义后,状态的描述

可以更加具体化。状态主要由状态名、状态类型、入口动作和出口动作、内部活动、内部转换(内部事件触发)、延迟事件6部分组成。

定义6 UML状态机的状态形式化为7元组

$State=(StateID, StateType, En\_Action, Ex\_Action, Do\_Action, IN\_Tr, Defer)$,并且

1)$StateID$是状态的标示符集。

2)$StateType$是状态类型集。

3)En_Action是状态的入口动作集。

4)Ex_Action是状态的出口动作集。

5)Do_Action是状态的内部活动集。

6)IN_Tr是状态的内部转换集。

7)$Defer$是状态内的延迟事件集。

定义7是在状态类型$StateType$的基础上对UML状态机中的状态进行语义扩展。

定义7 UML状态机的状态用RSL形式化定义为:

```
State=
  extend StateType with
    class
      type
        STATE:S,StateID:Text,EVENT,ACTION,
        StateType= State_type∪Composestate_type∪Pseudostate
                   _type ∪ Historystate _ type ∪ LPseudostate _
                   type,
        En_Action={|ac:ACTION,s:STATE · is_entryAction(s,
        ac)|},
        Ex_Action={|ac:ACTION,s:STATE · is_exitAction(s,
        ac)|},
        Do_Action={|ac:ACTION,s:STATE · is_doAction(s,ac)|},
        Defer={|eve:EVENT,s:STATE · is_defer_Event(s,eve)|
              },IN_Tr
      value
        get_StateAction_type:STATE×ACTION→StateAction_
        type,
        is_Activestate:STATE→Bool,
        is_inActivestate:STATE→Bool,
        is_stateEqual:STATE×STATE→Bool,
        get_Activestates:STATE→STATE-set
      axiom
        [active_state_axiom]
        ∀s:STATE,∃act:Action · is_Activestate(s)
        ≡get_StateAction_type(s,act)=en_Action,
        [Inactive_state_axiom]
        ∀s:STATE,∃act:Action · is_inActivestate(s)
        ≡get_StateAction_type(s,act)=ex_Action
end
```

3.2 状态层次结构

状态之间的层次结构体现在状态之间的包含关系上。本文中,状态之间的包含关系是指复合状态与子状态之间的关系,称为复合状态包含子状态或覆盖子状态。UML状态机中复合状态表示的是分层状态结构,是一种状态精化,使得一个状态包含多个子状态。

通过使用AND和OR可以将复合状态分解为并发子状态和顺序子状态。使用AND来分解的复合状态又叫并发复合状态(AND状态),当该状态是活动状态时,其所有直接子状态都处于活动状态,并发的子状态处于用虚线分割开的区域中;使用OR来分解的复合状态又叫顺序复合状态(OR状态),当该状态是活动状态时,只能有一个直接子状态处于活动状态。

定义8 UML状态机的状态层次结构形式化为一个5元组$SH=(S, subs, cntr, get_H, defult)$,并且满足以下特性:

(1)S是非空状态集。

(2)函数$cntr: S/\{\text{root}\}\rightarrow S$是状态抽象函数,描述状态之间子与父的直接层次关系,获取当前状态的父状态。$cntr(s)=c$表示子状态s直接被复合状态c包含。根状态root是唯一一个没有父状态的OR状态。

定义9 $cntr^*(s)$是$cntr(s)$的自反传递覆盖关系,$cntr^*(s)$定义了状态s的所有祖先,且满足$s\in cntr^*(s)$并且$\forall s_1, s_2 \in S/\{root\} \cdot s_1\in cntr^*(s_2)\Rightarrow cntr(s_1)\in cntr*(s_2)$。

(3)函数$subs: S\rightarrow S\text{-}set$是状态精化函数,描述状态之间的父与子的直接层次关系。$subs(c)=s$表示复合状态c直接包含子状态s。

定义10 $subs^*(s)$表示状态s的包含自身的子孙,即包含层次结构上的自反的传递包含关系。

如果对于状态序列$s_1, s_2, \cdots, s_n$有$subs(s_k)=s_{k-1}(1<k\leqslant n)$,则$subs^*(s_n)=s_1(1\leqslant n)$。即$subs(s_n)=s_{n-1}, subs(s_{n-1})=s_{n-2}, \cdots, subs(s_2)=s_1\Rightarrow subs^*(s_n)\supseteq s_1(1\leqslant n)$。称$s_n$是$s_1$的祖先,$s_1$是$s_n$的子孙。

定义11 $subs^+(s)$表示状态s的不包括自身的子孙,即包含层次结构上的非自反的传递包含关系。

如果对于状态序列$s_1, s_2, \cdots, s_n$有$subs(s_k)=s_{k-1}(1<k\leqslant n)$,则$subs^+(s_n)=s_1(1<n)$。即$subs(s_n)=s_{n-1}, subs(s_{n-1})=s_{n-2}, \cdots, subs(s_2)=s_1\Rightarrow subs^+(s_n)\supseteq s_1(1<n)$。

(4)在UML状态机的层次结构中,所有的状态机都有一个根节点状态root,它是任何状态的祖先。

定义12 root是UML状态机的根节点,其满足$\text{root}\in S, \forall s\in S\Rightarrow \text{root}\notin subs(s)\wedge s\in subs^*(\text{root})$。

(5)UML状态机图中有且只有一个初始状态,初始状态是根状态root的直接子状态。

$\exists c\in S\wedge is\_Initialstate(c)\Rightarrow c\in subs(root)\wedge subs(c)=\phi$

(6)任何一个非根节点状态,有且仅有一个直接父状态。

$\forall s\in S, \exists c_1, c_2\in S\cdot sub(c_2)=s\wedge sub(c_1)=s\wedge s\neq \text{root}$

$c_2=cntr(s)\wedge c_1=cntr(s)\wedge (c_2=c_1)$

(7)任何一个复合状态都具有孩子节点状态(子状态)。

$\forall c\in S\wedge is\_Complestatetype(c), \exists s\in S, subs(c)=s\Rightarrow s=subs(c)$

(8)状态分层中状态分类。

如果$subs(s)\neq\phi\wedge get_Statetype(s)=compose\wedge get_ComposestateType(s)=or$,称$subs(s)$是状态$s$的OR分解,当对象处于复合状态$s$时,实际处于$s$的一个子状态。

如果$subs(s)\neq\phi\wedge get_Statetype(s)=compose\wedge get_ComposestateType(s)=and$,称$subs(s)$是状态$s$的AND分解,当对象处于复合状态$s$时,实际处于$s$的所有子状态上。

如果$subs(s)=\phi\wedge get_Statetype(s)=simple$,则$s$是基本状态。

如果 $subs(s)=\phi \wedge get\_Statetype(s)=pseudo$，则 s 是伪状态。

(9)UML 状态图中的历史状态是针对顺序复合状态来讲的，并发复合状态中不存在历史状态。函数 $get\_History: Sph \rightarrow S$ 获取离开 OR 复合状态时的最后一个活动子状态。为了后期 UML 操作语义的形式化，扩展函数 $get\_History$ 为 $get\_H: S \rightarrow S$，此函数是一个分段函数：定义域分为是历史状态和非历史状态。

$\forall s \in S \backslash S_{ph} \Rightarrow get\_H(s)=s$

$\forall s \in S_{ph} \Rightarrow get\_H(s)=get\_History(s)$

(10)函数 $defult: S_{cs} \rightarrow S$ 获取默认子状态。默认子状态存在于顺序复合状态中。默认子状态的获取需要分为两种情况：一种是对于不存在历史状态或没有历史状态可用的情况下的 OR 状态，其默认子状态即为复合状态中的初始子状态(或者是初始子状态的第一个转换状态)；另一种是对于存在历史状态并且历史状态可用的情况下，其默认子状态为离开复合状态的最后的活动子状态。

与状态分层相关的函数如表 8 所列，状态分层可以对第 3.1.1 节中定义的状态进行语义扩展。

表 8　与状态分层相关的函数描述表

函数名	定义域	值域	函数功能描述
cntr	S\{root}	S	获取当前状态的直接父状态
cntr*	S\{root}	S	获取当前状态的祖先状态(自反传递)
subs	S	S 的幂集	获取当前状态的直接子状态集
subs*	S	S 的幂集	获取当前状态的子孙状态集(自反传递)
get_Root	S	S	获取当前状态的子孙状态集
is_Root	S	Bool	判断当前状态是否是根节点
is_inRoot	S	Bool	判断当前状态是否是非根节点
get_ Initialstate	S	S	获取 UML 状态机的初始状态
is_Initialstate	S	Bool	判断当前状态是否是初始状态
get_History	S_{ph}	S	获取离开 OR 复合状态时的最后一个活动子状态
get_H	S	S	获取最后一个活动状态
defult	S_{cs}	S	获取默认子状态

定义 13　UML 状态机的状态分层结构用 RSL 形式化定义为：

```
SH=
extend State with
  class
    type
      Sc={s:S • subs(s)≠φ ∧ get_Statetype(s)=compose },
      Scc={s:S • subs(s)≠φ ∧ get_Statetype(s)=compose ∧ get_
          ComposestateType(s)=and },
      Scs={s:S • subs(s)≠φ ∧ get_Statetype(s)=compose ∧ get_
          ComposestateType(s)=or },
      Ss={s:S • subs(s)=φ ∧ get_Statetype(s)=simple },
      Sp={s:S • subs(s)=φ ∧ get_Statetype(s)=pseudo }
      value
        cntr:S\{root}→S,
        cntr≡[s ↦c|s∈S\{root},c∈Sc • cntr(s)=c],
        cntr* :S\{root}→S-set,
        subs:S→S-set,
        subs≡[c ↦s|c∈Sc,s∈S\{root} • cntr(s)=c],
        subs* :S→S-set,
        subs+ :S→S-set
        get_Root:S\{root}→S,
        get_Root≡{c| ∀ s∈S\{root},∃ c∈S • s(subs* (c)}
        Is_Root:S→Bool,
        Is_inRoot:S→Bool,
        get_Initialstate:S→S,
        get_Initialstate ≡
        {init|init∈S • init∈subs(root) ∧ subs(init) =φ},
        is_Initialstate:S→Bool,
        get_History:Sph→S,
        get_H:S→S,
        defult:Scs→S
    axiom
      [cntr* _axiom]
        ∀ s1∈S\{root},∃ s2∈S • s2=cntr* (s1)⇒cntr(s1)∈
        cntr* (s2) ∧ s1∈cntr* (s1),
      [subs_axiom]
        ∀ c∈S∧ Is_complestatetype(c),∃ s∈S • subs(c)=s⇒
        s=subs(c)
      [subs* _axiom]
        ∀ s1∈S,∃ s2∈S • s2=subs* (s1)⇒subs(s2)⊆subs*
        (s1) ∧ s1∉subs* (s1),
      [subs+ _axiom]
        ∀ s1∈S,∃ s2∈S • s2=subs+ (s1)⇒subs(s2)⊆subs+
        (s1) ∧ s1∉subs+ (s1),
      [root_axiom]
        ∀ s∈S\{root},∃ c,c1∈S(sub(c)=s ∧ sub(c1)=s⇒c=
        cntr(s) ∧ c1=cntr(s) ∧ c=c1,
      [get_H_axiom]
        get_H(s)≡
          case is_HPseudostate(s) of
            ture→get_History(s)
            false→s
          end
    end
```

3.3　转换

转换是对象对事件作出的响应，包括源状态、目标状态、事件、监护条件和执行动作。UML 状态机中，转换表示成状态到状态的箭头与箭头标记。箭头和抽象状态机语法图中的边对应，箭头标记与抽象状态机语法图中的边标记对应，由事件 EVENT、监护条件 CONDITION 和动作 ACTION 的组合，在 UML 状态机中用 EVENT[CONDITION]/ACTION 表示。如果没有触发事件或动作，EVENT 和 ACTION 可以用符号"-"表示；如果没有监护条件，[CONDITION]可省略。当事件用符号"-"表示时，表示当源状态的内部活动执行完不需要外部事件就能自动离开当前源状态进入目标状态，称为无触发转换(完成转换)。转换的类型可以分为两大类：有触发转换和无触发转换。在状态图中，大部分的转换都是触发事件引发的转换。

当状态 s 是活动状态时，事件 EVENT 发生并且 CONDITION 为真时，转换发生，并执行相应的动作 ACTION。当转换发生时，对象状态从源状态转换到目标状态。如果源状态是复合状态。则所有的活动子状态退出并离开复合状态。如果目标状态是并发复合状态，则所有的子状态为活动状态。如果目标状态是顺序复合状态，当不存在历史状态时，默认初始子状态为活动状态，当存在历史状态并且历史状态可用时，最后离开目标复合状态的活动子状态为活动状态。可以通过

第 4.2 节中的 defult 函数获取目标复合状态的默认子状态。

根据转换有无触发事件，定义枚举类型 transfer_Type 来表示转换类型，定义函数 is_EventTR 来判断当前转换是否为触发转换。

定义 14 UML 状态机的转换用 RSL 形式化定义为：

```
TRANSITION=
extend Statechart with
  class
    type
      T:EDGE,
      transfer_Type==noEvent_TR| Event_TR
    value
      is_EventTR:T→Bool
    axiom
      is_EventTR(t)≡
        case is_hasEvent(t) of
          true→true
          false→false
        end
    end
```

根据转换的源、目标类型，可以将转移分为简单转换和复合转换。简单转换是指源状态和目标状态都是简单状态的转换[13]；复合转换是指由伪状态 fork、join、choice、junction 连接的简单转换，它们存在多个源状态和多个目标状态。fork 伪状态将迁移分割为多个到达不同正交状态的转换段；join 伪状态把从多个正交状态出发的转换段连接到 join 伪状态；choice 伪状态将单个转换分割成多个条件转移分支；多条件转移结束用 junction 伪状态连接到一起。图 2 是一个 UML 状态机复杂转换的实例。

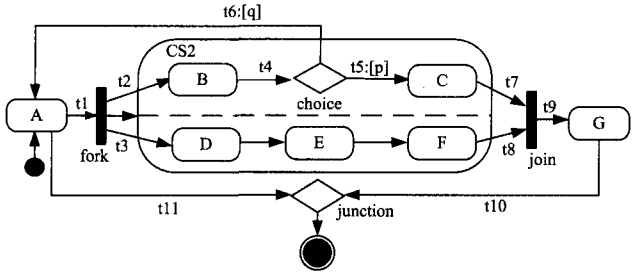

图 2 复合转换实例图

根据不同的连接伪状态连接的转换段，将转换划分为表 9 所列的类型。

表 9 转换分类描述表

状态分类（集合）	状态类型描述	集合描述
T_{ss}	状态到状态的转换	$T_{ss}\equiv\{e\in E\mid src(e)\in S\setminus S_{pl}\wedge dst(e)\in S\setminus S_{pl}\}$
T_{js}	并发结束(join)伪状态到状态的转换	$T_{js}\equiv\{e\in E\mid src(e)\in S_{pj}\wedge dst(e)\in S\setminus S_{pl}\}$
T_{sf}	状态到并发开始(fork)伪状态的转换	$T_{sf}\equiv\{e\in E\mid src(e)\in S\setminus S_{pl}\wedge dst(e)\in S_{pf}\}$
T_{us}	选择连接结束(jounction)到状态的转换	$T_{us}\equiv\{e\in E\mid src(e)\in S_{pu}\wedge dst(e)\in S\setminus S_{pl}\}$
T_{sc}	状态到选择开始(choice)伪状态的转换	$T_{sc}\equiv\{e\in E\mid src(e)\in S\setminus S_{pl}\wedge dst(e)\in S_{pc}\}$
T	所有外部转换	$T=T_{ss}\cup T_{js}\cup T_{sf}\cup T_{us}\cup T_{cs}$
T_i	内部转换	
T_{hs}	浅历史状态到状态的转换	$T_{hs}\equiv\{e\in E\mid src(e)\in S_{psh}\wedge dst(e)\in S\}$

在此转换分类的基础上定义获取转换属性的函数，如表 10 所列。

表 10 与转换相关的函数描述表

函数名	定义域	值域	函数功能描述
src_T	T	S	获取转换的源状态
dst_T	T	S	获取转换的目标状态
get_T_Event	T	EVENT 的幂集	获取转换的触发事件
get_T_Condition	T	CONDITION 的幂集	获取转换的监护条件
get_T_Action	T	ACTION 的幂集	获取转换的动作

定义 15 UML 状态机的转换类型用 RSL 形式化定义为：

```
TR=
extend TRANSITION with
  class
    type
      Tss={|e:E • src(e)∈S\Spl ∧ dst(e)∈S\Spl|}
      Tjs={|e:E • src(e)∈Spj ∧ dst(e)∈S\Spl|}
      Tsf={|e:E • src(e)∈S\Spl ∧ dst(e)∈Spf|}
      Tus={|e:E • src(e)∈Spu ∧ dst(e)∈S\Spl|}
      Tsc={|e:E • src(e)∈S\Spl ∧ dst(e)∈Spc|( }
      T=Tss∪Tjs∪Tsf∪Tus∪Tcs
      Ths={|e:E • src(e)∈Spsh ∧ dst(e)∈S\Spl|}
      Ti
    value
      src_T:T→S
      src_T≡[t ↦src(t)|t∈Tss∪Tsf∪Tsc∪Ti]∪[t ↦src(e)|
            t∈Tus,e∈E(dst(e)=src(t)]∪[t ↦X|t∈Tjs,S⊇X,
            e∈E(dst(e)=src(t)⇒src(e)∈X]
      dst_T:T→S
      dst_T≡[t ↦dst(t)|t∈Tss∪Tjs∪Tus∪Ti]∪[t ↦X|t∈Tsf
            ∪Tsc,S⊇X,e∈E • src(e)=dst(t)⇒dst(e)∈X ]
      get_T_Event:T→EVENT-set
      get_T_Event≡[t ↦get_Event(t)| t∈Tss∪Tjs∪Tsf∪Tsc∪
            Ti] ∪[t ↦get_Event(e)| t∈Tus,e∈E • dst
            (e)=src(t) ]
      get_T_Condition:T→ CONDITION-set
      get_T_Condition≡[t(get_Condition(t)|t∈Tss∪Tjs∪Tsf∪
            Ti]∪[t ↦(get_Condition(e)∧get_Condi-
            tion(t))|t∈Tus∪Tsc,e∈E • dst(e)=src
            (t)]
      get_T_Action:T→ACTION-set
      get_T_Action≡[t ↦Get_action(t)|t∈Tss∪Ti]∪[t ↦(Get
            _action(t)∪Get_action(e))| t∈Tus∪Tjs,e∈
            E • dst(e)=src(t)]∪[t ↦(Get_action(t)∪
            Get_action(e))|t∈Tsf∪Tsc,e∈E • src(e)=
            dst(t)]
    end
```

结束语 本文完成了 UML 状态机的模型元素的 RSL 形式化定义，在以下几个方面有所创新：

(1)先将 UML 状态机的抽象语法定义成一个图，再将图通过传统的有穷自动机进行扩展，最终采用 RSL 规约语言对模型元素进行形式化定义。将 UML 状态机的抽象语法定义成一个图的好处是为今后做一致性检查时进行状态搜索提供方便。

(2)将 UML 状态机中的状态类型、转换类型更加细化，

(下转第 205 页)

[5] Zhai C X, Cohen W W, Lafferty J. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval[C]// Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. New York, NY: ACM Press, 2003: 10-17

[6] Zhang B Z, Li H, Liu Y, et al. Improving web search results using affinity graph[C]// Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY: ACM Press, 2005: 504-511

[7] Zhang J, Xu H B, Cheng X Q. GSPSummary: A Graph-based Sub-topic Partition Algorithm for Summarization[C]// Proceedings of 2008 Asia Information Retrieval Symposium. AIRS, 2008: 321-334

[8] Zhu X J, Goldberg A, Gael J V. et al. Improving diversity in ranking using absorbing random walks: Human Language Technologies[C]// The Annual Conference of the North American Chapter of the Association for Computational Linguistics. Rochester, NY, NAACL-HLT, 2007

[9] Lin C Y, Hovy E H. From Single to Multi-document Summarization: A Prototype System and its Evaluation[C]// Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. Morristown, NJ: Association for Computational Linguistics, 2002: 25-34

[10] Harabagiu S, Lacatusu F. Topic themes for multi-document summarization[C]// Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY: ACM Press, 2005: 202-209

[11] Saggion H, Bontcheva K, Cunningham H. Robust generic and query-based summarization[C]// Proceedings of 10th Conference of the European Chapter of the Association for Computational Linguistics. Morristown, NJ: ACL, 2003: 235-238

[12] Conroy J M, Schlesinger J D. CLASSY query-based multi-document summarization[C]// Proceedings of Document Understanding Conference. Vancouver, Canada, 2005

[13] Page L, Brin S, Motwani R, et al. The PageRank Citation Ranking: Bringing Order to the Web [R]. Stanford, California: Stanford University Database Group, 1998

[14] Motwani R, Raghavan P. Randomized algorithms [M]. Cambridge Press, 1996, 28: 33-37

[15] Haveliwala T. Efficient computation of pagerank[R]. Stanford, California: Database Group, Computer Science Department, Stanford University, 1999

(上接第 183 页)

同时增加了状态分层，使模型语义更加精确，同时为今后的 UML 状态机的操作语义研究打下基础。

未来的工作包括：

(1)对本文定义的 UML 状态机的模型元素的形式化描述进行验证，同时考虑验证过程中的性能开销。

(2)在 UML 状态机模型元素语义形式化定义的基础上，进行操作语义的形式化定义。

(3)在 UML 状态机完整语义定义的基础上，进行 UML 图的一致性的自动检测[4]。

参 考 文 献

[1] OMG. UML2.0 Infrastructure Specification [OL]. http://www.omg.org/cgi-bin/doc? formal/2005-07-05.pdf, 2005

[2] OMG. Object Constraint Language. Version 2.3.1 [OL]. http://www.omg.org/cgi-bin/doc? formal/2009-02-02.pdf, 2009

[3] 蒋慧，林东，谢希仁. UML 状态机的形式语义[J]. 软件学报，2002, 13(12): 2244-2250

[4] Egyed A. Automatically Detecting and Tracking Inconsistencies in Software Design Models[J]. IEEE Transactions on Software Engineering, 2011, 37(2): 188-204

[5] Bjorner D. 软件工程卷 1：抽象与建模[M]. 刘伯超，向剑文，译. 北京：清华大学出版社，2010: 18-21

[6] Woodcock J. Formal Methods: Practice and Experience[J]. ACM Computing Surveys, 2009, 41(4): 19: 1-19: 36

[7] 郭峰，姚淑珍. 基于 Petri 网的 UML 状态图的形式化模型[J]. 北京航空航天大学学报，2007, 33(2): 248-252

[8] 董威，王戟，齐志昌. UML Statecharts 的模型检验方法[J]. 软件学报，2003, 14(4): 750-756

[9] 朱雪阳，唐稚松. Statecharts 的组合语义与求精[J]. 软件学报，2006, 17(4): 670-681

[10] Jin Yan, Esser R. A method for describing the syntax and semantics of UML statecharts[J]. Software System Model, 2004, 3(2): 150-163

[11] 单黎君，朱鸿. UML 的形式化描述语义[J]. 计算机工程与科学，2010, 32(3): 96-103

[12] 郭亮，缪淮扣，王晢，等. UML 模型到 FSM 模型的转换[J]. 计算机科学，2009, 36(7): 113-116

[13] Sun Meng, Zhang Nai-xiao. The Formalization for UML statechart Diagrams[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2005, 41(3): 344-356

[14] 李明，杨海波，张其文，等. 基于时序描述逻辑的 UML 状态图语义[J]. 计算机工程，2010, 36(23): 76-78

[15] 李留英，王戟，齐治昌. UML Statechart 图的操作语义[J]. 软件学报，2001, 12(12): 1865-1868

[16] 曾一. 基于形式化规格说明的 UML 状态图提取[J]. 计算机应用研究，2011, 28(5): 1767-1769

[17] Bendraou R. A Comparison of Six UML-Based Language for Software Process Modeling[J]. IEEE Transactions on Software Engineering, 2010, 36(5): 662-675

[18] Whittle J. Synthesizing Hierarchical State Machines from Expressive Scenario Descriptions[J]. ACM Transactions on Software Engineering, 2010, 19(3): 8: 1-8: 40

[19] Bjorner D. 软件工程卷 2：系统与语言规约[M]. 刘伯超，向剑文，译. 北京：清华大学出版社，2010: 406-413

[20] Ershov A P, Itkin V E. Correctness of mixed computation in Algol-like programs[J]. Mathematical Foundations of Computer Science, 1977, 53: 59-77