ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.4, April 2006, pp.670−681
DOI: 10.1360/jos170670

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

Statecharts 的组合语义与求精*

朱雪阳+, 唐稚松

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

Compositional Semantics and Refinement of Statecharts

ZHU Xue-Yang+, TANG Zhi-Song

(Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62622753, Fax: 86-10-62563894, E-mail: zxy@ios.ac.cn, http://www.ios.ac.cn

Zhu XY, Tang ZS. Compositional semantics and refinement of Statecharts. *Journal of Software*, 2006, 17(4):670−681. http://www.jos.org.cn/1000-9825/17/670.htm

Abstract: Statecharts is widely used as a behavioral modeling language for reactive systems for its concise and intuitive expression. It can represent the system behavior at different levels of abstraction, and therefore can represent the result of every refinement step in the process of system modeling. However, it's beyond its capability to reason about whether the model semantics of the lower level preserves that of the higher level and whether the models they describe satisfy some properties. In this aspect, the formal language XYZ/E can be used complementarily. The XYZ/E is an executable linear temporal logic. It can express both the properties and behavior of systems. In this paper, the semantics of Statecharts is defined inductively using the Basic Transition System, and its temporal semantics is expressed by an XYZ/E formula. The semantics we give is modularly compositional. The semantic preserving of different refinement steps can be guaranteed by the semantic definition directly. Models that Statecharts specify and their properties are represented in the same logic, so the assertion that a model meets its specification is expressed by logical implication.

Key words: Statecharts; temporal logic; XYZ/E; formal semantics; composition; refinement

摘 要: 由于简洁、直观的表达能力,Statecharts 被用于许多反应系统的行为建模.Statecharts 可表示不同抽象层次的系统行为,因而可用来表示逐步求精建模中各步的结果.但对于求精过程中下层是否保持了上层的语义、所建模型是否满足某些性质的问题,却难以在其自身的框架下进行讨论.在这方面,形式化语言 XYZ/E 可与其互补.XYZ/E 是一种可执行线性时序逻辑语言,既可表示系统的性质,又可表示系统的行为.递归地在基本迁移系统上解释 Statecharts 语义,用 XYZ/E 公式表示它的时序语义.这一语义是模块级可组合的.求精过程的语义保持,可直接从语义定义得到保证.Statecharts 所描述的系统行为模型和性质在同一个逻辑中表示,因此,系统行为是否满足所需性质的问题可由逻辑蕴涵式表示.

关键词: Statecharts;时序逻辑;XYZ/E;形式语义;组合;求精

* Supported by the National Natural Science Foundation of China under Grant Nos.60273025, 60223005 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2004AA1Z2100 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002cb312200 (国家重点基础研究发展规划(973))

Received 2004-07-29; Accepted 2005-10-20

中图法分类号: TP311 文献标识码: A

Statecharts 是适合为反应系统的行为建模的图形语言[1],在实际软件开发中得到广泛应用.它对传统的有限状态机作了层次、并发和广播通信 3 个方面的扩展,能够简洁、结构化地描述复杂反应系统的行为.它通过状态嵌套支持对不同抽象层次系统行为的描述,可以方便地进行逐步求精的系统建模,但对于求精过程中下层是否保持了上层的语义、所建模型是否满足所需性质的问题,却难以在其自身的框架下进行讨论.在这方面,具有严格语义的形式化语言可与其互补——用形式化语言描述模型及性质,再用相应的形式化技术(逻辑推理、模型检测等)对模型是否满足性质进行验证.所以,如果能为 Statecharts 所描述的模型定义精确的语义,建立起 Statecharts 与形式化语言之间的语义联系,就有可能利用形式化技术对上述问题进行严格的检测.线性时序逻辑(linear temporal logic,简写为 LTL)[2]是建立在一阶逻辑上的模态逻辑,在对反应系统的性质描述方面有着直观、表达能力强等优点.XYZ/E[3]是对 LTL 的扩充,引入了状态转换机制,使性质和行为可在同一个逻辑中表示.在 XYZ/E 框架下,考察系统行为是否满足性质的问题变成考察逻辑蕴涵式是否成立的问题.本文在基本迁移系统(basic transition system,简写为 BTS)[2]上解释 Statecharts 的语义,用 XYZ/E 公式表示 BTS 的时序语义[4],从而给出了相应的 Statecharts 的时序语义.我们以最简单的 Statecharts(没有层次和并发)为语义定义的基本元素,递归地定义两个组合操作(并发和嵌入)语义,得到完整的 Statecharts 语义.这一语义是在模块(或系统)级可组合的,求精的语义保持问题可直接从语义定义得到保证.更进一步的目标是建立起以逐步求精为指导思想,以 Statecharts 和 UML 活动图等图形语言为前端建模语言,以 XYZ/E 为严格的形式化语义基础的体系结构描述[5,6]框架,结合软件工程中常用方法(图形建模)与形式化方法(严格的语义及逻辑推理)的优点,提高软件的可靠性.

本文第 1 节简单介绍线性时序逻辑 XYZ/E 与基本转换系统.第 2 节介绍 Statecharts 并定义形式语法.第 3 节是全文的主体,递归地定义 Statecharts 的时序语义.第 4 节是关于 Statecharts 语义问题的进一步讨论.第 5 节比较相关工作.最后总结全文.

1 时序逻辑与基本转换系统简介

LTL 是以一阶逻辑为基础,在线性时间序列上解释语义的模态逻辑,常用于描述反应系统的性质.一阶逻辑公式的真假值在单个取值上计算,而时序逻辑公式的真假值在无限取值序列上计算.XYZ/E 是 LTL 的扩充,增加了状态转换等式$\$Ox=e$ 作为逻辑公式,从而具有了描述系统行为的能力.

首先说明几个要用到的符号.设 V 是变量集,D 是数据域,保持类型的映射 $\sigma:V\to D$ 是对 V 中变量的取值,$\Sigma(V)$ 表示 V 上所有取值的集合,在 V 明确的情况下,直接用 Σ 表示.$\sigma(e)$($\sigma(p)$)是指 V 上表达式 e(一阶逻辑公式 p)在取值 σ 下的值.例如,若 $V=\{x,y\}$,D 为整数域,$\sigma=\{x=1,y=2\}$,则 $\sigma(x+y)=3$,$\sigma(x>y)$=false.

设 r 是一个无限取值序列:$\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_i,\ldots$,

(r,i)表示 r 中截去前 i 个取值后剩下的取值序列:$\sigma_i,\sigma_{i+1},\ldots$

设 p 为一阶逻辑公式,q,q_1,q_2 为时序逻辑公式,x 为变量,e 为表达式.下面是本文用到的时序算子解释:

$(r,i)\models p$	当且仅当	$\sigma_i(p)$=true	/*一阶逻辑公式在当前取值上解释
$(r,i)\models \$Oq$	当且仅当	$(r,i+1)\models q$	/*下一时刻 q 为真
$(r,i)\models \Box q$	当且仅当	对于所有的 $k\geq i$:$(r,k)\models q$	/*从当前时刻起 q 总是为真
$(r,i)\models \Diamond q$	当且仅当	存在 $k\geq i$:$(r,k)\models q$	/*从当前时刻起 q 终将为真
$(r,i)\models \$Ox=e$	当且仅当	$\sigma_{i+1}(x)=\sigma_i(e)$	/*下一时刻 x 的值等于当前时刻 e 的值
$(r,i)\models \neg q$	当且仅当	$(r,i)\not\models q$	
$(r,i)\models q_1\wedge q_2$	当且仅当	$(r,i)\models q_1$ 并且 $(r,i)\models q_2$	
$(r,i)\models q_1\vee q_2$	当且仅当	$(r,i)\models q_1$ 或 $(r,i)\models q_2$	

一个时序逻辑公式可以表示在一定取值域(σ_i 的取值域)下,满足该公式的所有取值序列(r)的集合.时序逻辑公式的$\wedge,\vee,\neg$分别对应集合的交、并、补运算.

我们将用基本迁移系统来定义 Statecharts 的语义.下面先给出基本迁移系统的抽象模型.

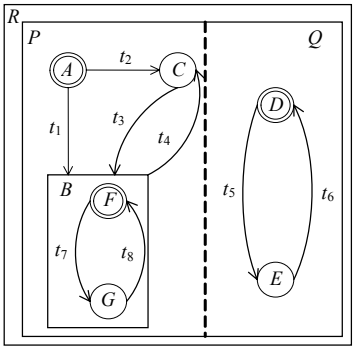
定义 **1**. 基本迁移系统 BTS 是一个四元组$(\Pi,\Sigma,\theta,\Delta)$,其中,$\Pi=\{u_1,u_2,\ldots,u_n\}$是系统的变量集;$\Sigma$是$\Pi$中变量取值的集合;$\theta$是初始条件,表示系统初始值应满足的条件;$\Delta=\{\delta|\delta:\Sigma\to\Sigma\}$是迁移关系集合,$\delta$表示系统的一个迁移动作,可能引起变量取值的变化,每个δ用一个动作发生前后变量取值的关系断言ρ_δ来表示,其中每个变量 u 有两个版本:一个是动作发生前的,用 u 表示;另一个是动作发生后的,用$\$Ou$ 表示.$\$O\Pi=\{\$Ou_1,\$Ou_2,\ldots,\$Ou_n\}$,$\rho_\delta\equiv_{def}C_\delta(\Pi)\wedge(\$Ou_1=e_1)\wedge\ldots\wedge(\$Ou_n=e_n)$,其中 $C_\delta(\Pi)$表示δ能够发生的必要条件,$\$O\Pi$是动作发生的效果.

BTS 的行为是它所有可能的计算(computation)集合,计算是一个取值的无限序列 $r:\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_i,\ldots$,满足条件:1) $\sigma_0(\theta)$=true,即 $r|=\theta$;2) $\forall i\in N,\exists\delta\in\Delta,^{st.}\delta(\sigma_i)=\sigma_{i+1}$,即$(r,i)|=\rho_\delta$.

BTS 的所有计算集合是 $Comp(BTS)=\{r|r|=\theta\wedge\Box(\vee_{\delta\in\Delta}\rho_\delta)\}$.所以,在变量集及取值确定的情况下,用时序逻辑公式 $bs_TLF(BTS)$表示基本迁移系统的时序语义:$bs\_TLF(BTS)\equiv_{def}\theta\wedge\Box(\vee_{\delta\in\Delta}\rho_\delta)$.

2 Statecharts 及其抽象语法

Statecharts 由状态和转换组成.它对传统的有限状态机进行了 3 个方面的扩充:层次——通过状态嵌套实现,一个状态可由多个状态组成,即状态可求精成一个 Statechart;并发——允许多个复合状态并发执行;广播通信——并发状态通信是通过事件广播来实现的.



Label: t_1:e_1/a_1, t_2: e_2, t_3:e_5, t_4:e_6
t_5: a_1/e_3, t_6:e_4, t_7:e_3, t_8:e_4

Fig.1 Statechart *SCR*

图 1 Statechart *SCR*

有 3 类状态:基本状态(BASIC)、异或状态(OR)和与状态(AND).基本状态没有子状态,不能被进一步求精,如图 1 所示的 A,C,D,E,F,G.异或状态和与状态都是复合状态.异或状态由一些子状态通过转换连接而成,当该状态为当前活动状态时,有且仅有一个它的子状态为当前活动状态.每个复合状态有一个(或多个)称为缺省状态的子状态,当该复合状态为当前活动状态时,它的缺省状态同时变为活动状态.如图 1 所示 P,Q,B 是异或状态,它们的缺省状态分别是 A,D,F.与状态包含一组异或状态作为它的子状态,也都是它的缺省状态.如图 1 所示的状态 R,当它被激活时,它的所有子状态(P 和 Q)同时都被激活,各子状态表示的活动开始并发执行,通过事件广播进行通信.当系统退出复合状态时,它的所有子状态同时被退出.

转换表示从它的源状态到目标状态的转移关系.其上带的标号由两部分组成.第 1 部分称为触发条件,包括一个事件表达式和一个条件表达式,二者都可缺省.事件和条件本质上都是布尔值,只是刷新机制不同,事件稍现即逝,只对一步起作用,而条件可能是持久的.我们暂不考虑有条件表达式的情况.第 2 部分称为动作,由一组原子事件组成,可为空.当一个转换的源状态是当前状态、环境及系统提供它的触发事件时,该转换可能执行.执行后,将动作中的事件在环境和系统中广播.同时,退出源状态进入目标状态.

设可数无限状态集为 $STATES$,可数无限转换集为Γ;可数无限原子事件集为 $EVENTS$,状态和转换名唯一.以下是 Statecharts 的形式语法定义.

定义 **2**. 一个 Statechart SC 是一个七元组$(S,root,type,children,default,ES,T)$,其中

$S\cup\{root\}\subseteq STATES$,是 SC 中所有状态名的集合,$root$ 是整个 SC 的根状态;

$type:S\cup\{root\}\to\{\text{BASIC,OR,AND}\}$ 是 SC 中状态的类型函数;SC 的所有状态应满足类型约束 $Stype\equiv_{def}(\forall s\in S\cdot type(s)=\text{BASIC}\vee' type(s)=\text{OR}\vee' type(s)=\text{AND})\wedge(type(root)=\text{OR}\vee' type(root)=\text{AND})$.其中$\vee'$表示不可兼析取连接词,$\varnothing$表示空集.

$children:S\cup\{root\}\to 2^S$,定义 SC 中状态子状态的函数,表示状态间的层次关系;不同状态应满足层次约束 $Shierarchy\equiv_{def}\forall s\in S\cup\{root\}\cdot((type(s)=\text{BASIC}\wedge children(s)=\varnothing)\vee(\neg type(s)=\text{BASIC}\wedge children(s)\neq\varnothing))$.

若 $s_2\in children(s_1)$,则称 s_1 是 s_2 的双亲,记为 $s_1=parent(s_2)$,s_2 是 s_1 的孩子.S 中任一状态只有唯一双亲.$root$ 是

唯一没有双亲的状态,满足约束 $Sroot\equiv_{\text{def}}\forall s\in S\cdot root\notin children(s)$.

定义 *children* 的自反传递闭包 $children^*:children^*(s)=\cup_{i\geq 0}children^i(s)$.其中,$children^0(s)=\{s\}$,$children^1(s)=children(s)$,对所有 $i\geq 1$,有 $children^{i+1}(s)=\cup_{s'\in children(s)}children^i(s')$.

若 $s_2\in children^*(s_1)$,则称 s_1 是 s_2 的祖先,s_2 的祖先集合记为 $parent^*(s_2)$,s_2 是 s_1 的子孙.每个状态 s 既是自己的祖先又是自己的子孙.这样,层次关系将 *SC* 组织成一个以 *root* 为根、以基本状态为叶子结点的树型结构,它的内部结点是除 *root* 以外的所有复合状态.

$default:S\cup\{root\}\rightarrow 2^S$,定义 *SC* 中状态的缺省状态集.基本状态没有子状态,因而没有缺省状态;异或状态有唯一缺省状态,与状态的所有子状态都是它的缺省状态;满足约束

$$Sdefault\equiv_{def}\forall s\in S\cup\{root\}\cdot(type(s)=\text{BASIC}\wedge default(s)=\varnothing)$$
$$\vee(type(s)=\text{OR}\wedge\exists_1 s_0\in children(s)\cdot default(s)=\{s_0\})\vee(type(s)=\text{AND}\wedge default(s)=children(s)).$$

$ES\subseteq EVENTS$,表示系统可以接受或生成的原子事件集,不包括系统隐含的进入状态事件($ENTER(s)$)与退出状态事件($EXIT(s)$);

$T\subseteq S\times Label\times S\subseteq I$表示 *SC* 中所有转换集 *Label* 是转换边上所有可能的标号集,$Label=\{Trigger/Action|Trigger\in eExp,Action\subseteq ES\}$;$\lambda$是一特殊事件,表示边上触发事件为空的情况;事件表达式集 *eExp* 是满足条件 1)~4)的最小集:1) $\lambda\in eExp$;2) 如果 $a\in ES$,则 $a\in eExp$;3) 如果 $e\in eExp$,则$\neg e\in eExp$;4) 如果 $e_1,e_2\in eExp$,则 $e_1\wedge e_2\in eExp$ 且 $e_1\vee e_2\in eExp$.

$t=(s_1,Trigger/Action,s_2)$表示从源状态 s_1 到目标状态 s_2 的转换,带标号 *Trigger*/*Action*.为了叙述方便,定义几个相关函数:$source(t)=s_1$ 表示 t 的源状态,$target(t)=s_2$ 表示 t 的目标状态,$trigger(t)=Trigger$ 表示 t 的触发部分,$action(t)=Action$ 表示 t 执行时生成的事件集.$action(t)$为空时可将 *Action* 部分省略,写成 $t=(s_1,Trigger,s_2)$形式.不允许转换发生在不同层次的两个状态之间,即转换必须满足约束

$$STran\equiv_{def}\forall t\in T\cdot\exists s\in S\cup\{root\}\cdot source(t)\in children(s)\wedge target(t)\in children(s).$$

一个合法的 Statechart 应满足语法约束:$Stype\wedge Shierarchy\wedge Sroot\wedge Sdefault\wedge Stran$.

如图 1 所示,*SCR* 的抽象表示:$S=\{P,A,B,C,F,G,Q,D,E\}$;$ES=\{e_1,e_2,e_3,e_4,e_5,e_6,a_1\}$;$T=\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8\}$;$root=R$;$type(R)=$AND,$children(R)=\{P,Q\}$,$default(R)=\{P,Q\}$;$type(P)=type(Q)=type(B)=$OR,其他都是 BASIC 型;

$children(P)=\{A,B,C\}$,$children(Q)=\{D,E\}$,$children(B)=\{F,G\}$,其他都是$\varnothing$;

$default(P)=\{A\}$,$default(Q)=\{D\}$,$default(B)=\{F\}$,其他都是$\varnothing$.

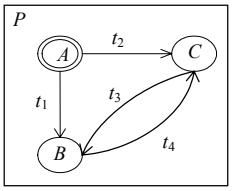
为了定义 Statecharts 的组合语义,我们先定义基本元素——基本 Statecharts(basic Statecharts,简写为 BSC)和两个组合操作——并发(and)和嵌入(embed).

定义 **3**. 基本 Statechart $BSC=(S,root,type,children,default,ES,T)$是满足如下条件的 Statechart:

$$type(root)=\text{OR}\wedge children(root)=S\wedge\forall s\in S\cdot type(s)=\text{BASIC}.$$

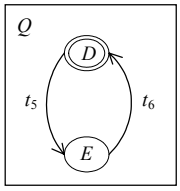
基本 Statecharts 是单层的,简记为 $BSC=(S,root,s_0,ES,T)$,其中 $default(root)=\{s_0\}$.

如图 2 所示的 *BSCP*,*BSCQ*,*BSCB* 都是基本 Statecharts.



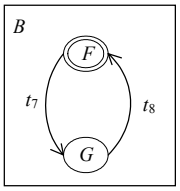
Label: t_1:e_1/a_1, t_2: e_2, t_3:e_5, t_4:e_6

Basic Statechart *BSCP*



Label: t_5: a_1/e_3, t_6:e_4

Basic Statechart *BSCQ*



Label: t_7:e_3, t_8:e_4

Basic Statechart *BSCB*

Fig.2 Basic Statecharts

图 2 基本 Statecharts

定义 **4**. Statechart $SC_1=(S_1,root_1,type_1,children_1,default_1,ES_1,T_1)$,$SC_2=(S_2,root_2,type_2,children_2,default_2,ES_2,T_2)$,则并发组合后的 Statechart and$(SC_1,SC_2)=(S,root,type,children,default,ES,T)$.其中,$S=S_1\cup\{root_1\}\cup S_2\cup\{root_2\}$,$root$是新生成的复合状态,$ES=ES_1\cup ES_2$,$T=T_1\cup T_2$.

$$type(s)=\begin{cases}type_1(s), & s\in S_1\cup\{root_1\}\\ type_2(s), & s\in S_2\cup\{root_2\}\\ \text{AND}, & s=root\end{cases};$$

$$children(s)=\begin{cases}children_1(s), & s\in S_1\cup\{root_1\}\\ children_2(s), & s\in S_2\cup\{root_2\}\\ \{root_1,root_2\}, & s=root\end{cases};$$

$$default(s)=\begin{cases}default_1(s), & s\in S_1\cup\{root_1\}\\ default_2(s), & s\in S_2\cup\{root_2\}\\ \{root_1,root_2\}, & s=root\end{cases}.$$

并发组合后增加一个 AND 型的根状态.

如图 2 所示的 and$(BSCP,BSCQ)$的 $root=R$;$S=\{A,B,C,P,D,E,Q\}$;$ES=\{e_1,e_2,e_3,e_4,e_5,e_6,a_1\}$;$T=\{t_5,t_6,t_7,t_8\}$;$type(R)$=AND,$children(R)=\{P,Q\}$,$default(R)=\{P,Q\}$,其他不变.

定义 **5**. Statecharts $SC_1=(S_1,root_1,type_1,children_1,default_1,ES_1,T_1)$,$ps\in S_1$ 并且 $type_1(ps)$=BASIC,$SC_2=(S_2,root_2,type_2,children_2,default_2,ES_2,T_2)$,则嵌入组合后的 Statechart embed$(SC_1,ps,SC_2)=(S,root,type,children,default,ES,T)$.其中:$S=S_1\cup S_2$;$root=root_1$;$ES=ES_1\cup ES_2$;$T=T_1\cup T_2$.

$$type(s)=\begin{cases}type_1(s), & s\in S_1\cup\{root_1\}-\{ps\}\\ type_2(root_2), & s=ps\\ type_2(s), & s\in S_2\end{cases};$$

$$children(s)=\begin{cases}children_1(s), & s\in S_1\cup\{root_1\}-\{ps\}\\ children_2(root_2), & s=ps\\ children_2(s), & s\in S_2\end{cases};$$

$$default(s)=\begin{cases}default_1(s), & s\in S_1\cup\{root_1\}-\{ps\}\\ default_2(root_2), & s=ps\\ default_2(s), & s\in S_2\end{cases}.$$

嵌入组合后,语法上看 $root_2$ 被丢弃了,实际上是 ps 发生改变,$root_2$ 代替了原来的 ps.

如图 2 所示的 embed$(BSCP,B,BSCB)$的 $root=P$;$S=\{A,B,C,F,G\}$;$ES=\{e_1,e_2,e_5,e_6,e_7,e_8,a_1\}$;$T=\{t_1,t_2,t_3,t_4,t_7,t_8\}$;$type(B)$=OR;$children(B)=\{F,G\}$;$default(B)=\{F\}$,其他不变.

定理 **1**. 任一 Statechart 或者是基本 Statechart 或者可从基本 Statecharts 通过上述两个操作组合而成,即

$$SC::=BSC|\text{and}(SC,SC)|\text{embed}(SC,ps,SC).$$

如图 1 所示的 SCR 可由图 2 所示的 3 个基本 Statecharts 组合而成,即,SCR=and(embed$(BSCP,B,BSCB)$, $BSCQ$).

3 Statecharts 组合语义定义

3.1 语义定义说明

状态相关函数 $type$,$children$ 和 $default$ 用于语法约束,$root$ 表示将整个 Statechart 框起来的复合状态,不在语义中考虑.当转换的触发条件不满足时,系统在源状态中等待,实际上是在作踏步(不改变任何系统可改变量取值的迁移动作).所以在语义上,每个状态上隐含了一条踏步转换边.语义定义需要的所有转换,记为 AT.$AT=T\cup T'\cup T_{\text{end}}$,其中 T 是图中原有的显式转换.

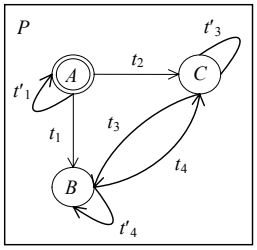
$T'=\{(s,Trigger,s)|\exists t\in T\cdot s=source(t)\wedge Trigger=(\wedge_{source(t)=s}\neg trigger(t))\}$ /*为有出边的状态定义一个到自身的转换.

$T_{end}=\{(s,Trigger,s)|\forall t\in T\cdot s\neq source(t)\wedge Trigger=\lambda\}$ /*为没有出边的状态定义一个到自身的转换,没有触发条件.

T'是每个有出边状态的踏步转换集,由于 Statecharts 的实时性要求:转换条件一旦满足并且具有最高优先级(不确定的情况下选择一个),转换必须执行.所以踏步是有条件的.T_{end} 是无出边状态的踏步转换集,对于每个没有出边的状态,定义一个无条件踏步转换.如图 3 所示,*BSCP* 的 $AT=\{t_1,t_2,t_3,t_4,t'_1,t'_3,t'_4\}$.

所以,语义定义要用到的基本 Statecharts 的结构是一个四元组(S,s_0,ES,AT),其中 s_0 是根的缺省状态.

关于基本迁移系统:在上下文明确的情况下,变量取值集合(Σ)省略.将变量分为两个部分:一部分是系统控制变量集 *CV*;另一部分是可观察变量,有两个:一个是当前系统生成的事件集合变量 *se*;反应系统不断保持与环境的交互,常常和环境一起作为一个完整的闭系统来考虑,所以还要有一个当前环境事件集合变量 *env*.系统当前的触发事件由 *se* 和 *env* 共同提供.



Label: t_1:e_1/a_1, t_2:e_2, t_3:e_5, t_4:e_6
t'_1:$\neg e_1\wedge e_2$, t'_3:$\neg e_5$, t'_4:$\neg e_6$

Fig.3 All transitions

图 3 所有转换示例

关于 Statecharts 语义:考虑离散时间域,即时间被看成是时刻的序列.事件是离散的,仅对一步起作用.一个事件出现时,它或者触发 Statecharts 的某个转换,或者被抛弃.每一步的动作仅在下一步起作用,即当前生成的事件,在下一步才作为触发事件.Statecharts 中有两类不确定性:一类可通过设定优先级来解决.例如,图 1 的 *SCR* 中,当系统处于状态 *G*(同时也处于 *B*)、外部提供事件$\{e_4,e_6\}$时,t_8,t_4 都具备执行条件,冲突产生.若设定外层优先策略,则 t_4 执行;设定内层优先策略,则 t_8 执行.我们采用外层优先策略;一类只能保留.例如,在 *SCR* 中,当系统处于状态 *A*、外部提供事件$\{e_1,e_2\}$时,t_1,t_2 都可执行,选择哪个是不确定的.语义定义独立于时间模型,同步或异步时间模型体现在对 *env* 的约束上,这一点将在第 4 节中详细说明.对于 $source(t)=target(t)$的转换 *t* 的执行,认为没有状态退出和进入的动作,不发出状态进入和退出事件.只有 $source(t)\neq target(t)$的转换 *t* 的执行时,系统发出 EXIT($source(t)$)和 ENTER($target(t)$)事件.

祖先与子孙状态的进入与退出之间有如下关系(其中"⇒"表示逻辑蕴涵):

ENTER(s)⇒$\wedge_{s'\in parent^*(s)}$ENTER($s'$) /*进入状态 *s* 蕴含进入它的所有祖先状态;

EXIT(s)⇒$\wedge_{s'\in children^*(s)}$EXIT($s'$) /*退出状态 *s* 蕴含退出它的所有子孙状态.

3.2 语义定义

我们用基本迁移系统(BTS)作为 Statecharts 的语义域,Statechart 的时序语义由 BTS 的时序语义定义.以下从基本 Statecharts 开始,递归地给出 Statecharts 到 BTS 的语义映射 *bts* 和 Statecharts 到时序逻辑公式的时序语义映射 *bs_TLF* 的定义.

定义 6. 基本 Statechart $BSC=(S,s_0,ES,AT)$,则 $bts(BSC)=(CV,se,env,\theta,\Delta)$,其中:

$CV=\{LB\}$,*LB* 是系统的控制变量,指示当前活动状态,*LB* 在 *S* 中取值,基本 Statechart 只有一个控制流程,故只有一个控制变量;

se 是系统生成事件集合,在 2^{ES} 中取值;

env 是环境生成事件集合,在 2^{ES} 中取值;

$\theta\equiv_{def} LB=s_0\wedge se=\varnothing\wedge env=\varnothing$;

$\Delta=\{\delta$|每个δ对应一个 $t\in AT$,转换关系为ρ_δ(也用ρ_t表示)$\}$;

$\rho_t\equiv_{def} LB=source(t)\wedge occ(trigger(t))\wedge\$OLB=target(t)\wedge\$Ose=action(t)$;

其中 $occ(e)$解释如下($a\in ES,e\in eExp$):

$occ(\lambda)$=true;

occ(ENTER(s)) 当且仅当 $LB\neq s\wedge\$OLB=s$;

$occ(\text{EXIT}(s))$ 当且仅当 $LB=s\wedge \$OLB\neq s$;

$occ(a)$ 当且仅当 $a\in se\vee a\in env$;

$occ(\neg e)$ 当且仅当 $\neg occ(e)$;

$occ(e_1\wedge e_2)$ 当且仅当 $occ(e_1)\wedge occ(e_2)$;

$occ(e_1\vee e_2)$ 当且仅当 $occ(e_1)\vee occ(e_2)$.

BSC 的时序语义为 $bs\_TLF(BSC)\equiv bs\_TLF(bts(BSC))\equiv\theta\wedge\Box(\vee_{t\in AT}\rho_t)$.

不带$O 算子的部分表示转换发生的条件,带$O 算子的部分表示转换发生的效果;$trigger(t)=\lambda$表示触发条件恒真;Ose=action(t)$说明:(1) 事件仅对一步起作用,因为每一时刻都对 *se* 重新置值;(2) 转换边上的动作对下一步起作用;每一时刻环境的变化体现在环境事件集 *env* 上,系统可观察到 *env* 的变化,但不能改变它;表示时序语义的时序逻辑公式中有冗余,例如,对于 $trigger(t)=\lambda$的转换边 *t*,相应 *t*′的 $trigger(t')=\neg\lambda, occ(\neg\lambda)=\text{false}$, $\rho_{t'}=\text{false}$,这样,*t*′表示的转换永远不会发生,这种情况放在公式简化时考虑;与 $action(t)$中事件不同的是,由于$\rho_t\Rightarrow occ(\text{EXIT}(source(t))\wedge\text{ENTER}(target(t)))$,退出源状态和进入目标状态事件在转换执行的同时起作用.

图 2 中 *BSCP* 的 $S=\{A,B,C\}, s_0=A, ES=\{e_1,e_2,e_5,e_6,a_1\}, AT=\{t_1,t_2,t_3,t_4,t_1',t_3',t_4'\}$.

$$
\begin{aligned}
bs\_TLF(BSCP)\equiv_{\text{def}} & LB_P=A\\
&\wedge\Box(LB_P=A\wedge occ(e_1)\wedge\$OLB_P=B\wedge\$Ose=\{a_1\}\\
&\quad\vee LB_P=A\wedge occ(e_2)\wedge\$OLB_P=C\wedge\$Ose=\varnothing\vee LB_P=A\wedge occ(\neg e_1\wedge\neg e_2)\wedge\$OLB_P=A\wedge\$Ose=\varnothing\\
&\quad\vee LB_P=B\wedge occ(e_6)\wedge\$OLB_P=C\wedge\$Ose=\varnothing\vee LB_P=B\wedge occ(\neg e_6)\wedge\$OLB_P=B\wedge\$Ose=\varnothing\\
&\quad\vee LB_P=C\wedge occ(e_5)\wedge\$OLB_P=B\wedge\$Ose=\varnothing\vee LB_P=C\wedge occ(\neg e_5)\wedge\$OLB_P=C\wedge\$Ose=\varnothing)
\end{aligned}
$$

定义 **7**. 并发组合 Statecharts SC_1 和 $SC_2, bts(SC_1)=(CV_1,se_1,env_1,\theta_1,\Delta_1), bts(SC_2)=(CV_2,se_2,env_2,\theta_2,\Delta_2)$,则 $bts(\text{and}(SC_1,SC_2))=(CV,se,env,\theta,\Delta)$,其中,$CV=CV_1\cup CV_2$;*se* 的值是 se_1 和 se_2 值的并,即,$\Sigma(se)=\Sigma(se_1)\cup\Sigma(se_2)$;*env* 的值是 env_1 和 env_2 值的并,即$\Sigma(env)=\Sigma(env_1)\cup\Sigma(env_2)$;$\theta=\theta_1\wedge\theta_2$;$\Delta=\{\delta|\exists\delta_1\in\Delta_1,\delta_2\in\Delta_2\cdot\rho_\delta=\rho_{\delta_1}\wedge\rho_{\delta_2}\}$.

$$
\begin{aligned}
&bs\_TLF(\text{and}(SC_1,SC_2))\equiv_{\text{def}}\theta\wedge\Box(\vee_{\delta\in\Delta}\ \rho_\delta)\equiv\theta_1\wedge\theta_2\wedge\Box(\vee_{\delta\in\Delta_1}\rho_\delta\wedge\vee_{\delta\in\Delta_2}\rho_\delta)\\
&\equiv(\theta_1\wedge\Box(\vee_{\delta\in\Delta_1}\rho_\delta))\wedge(\theta_2\wedge\Box(\vee_{\delta\in\Delta_2}\rho_\delta))\equiv bs\_TLF(SC_1)\wedge bs\_TLF(SC_2).
\end{aligned}
$$

嵌入组合本质上也是并发组合,但在 $\text{embed}(SC_1,ps,SC_2)$中,只有当上层的 SC_1 处于被嵌入状态 *ps* 时,子 Statechart SC_2 才活动(并发的 Statecharts 是同时活动的),与父状态 *ps* 的踏步并发;而当 SC_1 不在 *ps* 时,SC_2 处于不活动状态.故需引入一个特殊状态“⊥”,用以表示整个 SC_2 不活动的情况.并且 SC_2 中需增加外部事件 ENTER(*ps*)和 EXIT(*ps*),作为 SC_2 激活和退出的触发事件.在 ENTER(*ps*)发生的同时进入 SC_2 的缺省状态;在 EXIT(*ps*)发生的同时退出 SC_2.所以,在给出嵌入组合的语义之前,先对 SC_2 进行预处理.不妨设 SC_1, SC_2 都是 *BSC*.

定义 **8**. 将 $SC_2=(S_2,s_{02},ES_2,AT_2)$嵌入 $SC_1=(S_1,s_{01},ES_1,AT_1)$的 *ps* 中,对 SC_2 进行预处理,得到基本 Statechart $pre(SC_1,ps,SC_2)=(S,s_0,ES,AT)$,其中

$$S=S_2\cup\{\perp\};\ s_0=\begin{cases}s_{02}, & ps=s_{01}\\ \perp, & ps\neq s_{01}\end{cases}$$

$ES=ES_2\cup\{\text{ENTER}(ps),\text{EXIT}(ps)\}$;$AT=A\,T_2'\cup T_{ex}\cup\{t_{en},t_\perp\}$;

$A\,T_2'=\{(s_1,Trigger/Action,s_2)|\exists t\in AT_2\cdot s_1=source(t)\wedge s_2=target(t)$

$\wedge Trigger=(trigger(t)\wedge\neg\text{EXIT}(ps))\wedge Action=action(t)\}$

/*只有当 *ps* 不退出时,SC_2 转换照常执行.

$T_{ex}=\{(s_1,Trigger,s_2)|\exists s\in S_2\wedge s_1=s\wedge Trigger=\text{EXIT}(ps)\wedge s_2=\perp\}$

/*退出 *ps* 的同时退出 SC_2,保证了外层优先.

$t_{en}=(\perp,\text{ENTER}(ps),s_{02})$ /*进入 *ps* 的同时,激活 SC_2.

$t_\perp=(\perp,\neg\text{ENTER}(ps),\perp)$/*$SC_1$ 不在 *ps* 时,SC_2 不活动,在⊥处踏步.

如图 4 所示,$pre(BSCP,B,BSCB)$的 $S=\{F,G,\perp\}; s_0=\perp; ES=\{e_3,e_4,\text{ENTER}(B),\text{EXIT}(B)\}; AT=\{t_7',t_7'',t_8',t_8'',t_{en},t_{ex1},t_{ex2},t_\perp\}$.

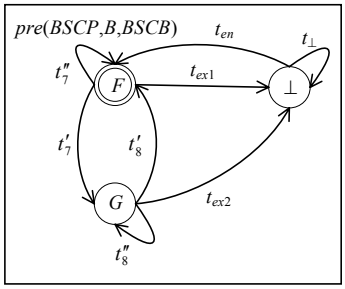
$bs\_TLF(pre(BSCP,B,BSCB))\equiv_{def}LB_B=\perp$

$\wedge\Box(LB_B=\perp\wedge occ(\text{ENTER}(B))\wedge\$OLB_B=F\wedge\$Ose=\varnothing\vee LB_B=\perp\wedge occ(\neg\text{ENTER}(B))\wedge\$OLB_B=\perp\wedge\$Ose=\varnothing$

$\vee LB_B=F\wedge occ(e_3\wedge\neg\text{EXIT}(B))\wedge\$OLB_B=G\wedge\$Ose=\varnothing\vee LB_B=F\wedge occ(\neg e_3\wedge\neg\text{EXIT}(B))\wedge\$OLB_B=F\wedge\$Ose=\varnothing$

$\vee LB_B=F\wedge occ(\text{EXIT}(B))\wedge\$OLB_B=\perp\wedge\$Ose=\varnothing\vee LB_B=G\wedge occ(e_4\wedge\neg\text{EXIT}(B))\wedge\$OLB_B=F\wedge\$Ose=\varnothing$

$\vee LB_B=G\wedge occ(\neg e_4\wedge\neg\text{EXIT}(B))\wedge\$OLB_B=G\wedge\$Ose=\varnothing\vee LB_B=G\wedge occ(\text{EXIT}(B))\wedge\$OLB_B=\perp\wedge\$Ose=\varnothing)$



$t'_7:e_3\wedge\neg\text{EXIT}(B),t''_7:\neg e_3\wedge\neg\text{EXIT}(B),t_{ex1}:\text{EXIT}(B)$
$t'_8:e_4\wedge\neg\text{EXIT}(B),t''_8:\neg e_4\wedge\neg\text{EXIT}(B),t_{ex2}:\text{EXIT}(B)$
$t'_{en}:\text{ENTER}(B),t_\perp:\neg\text{ENTER}(B)$

Fig.4 $pre(BSCP,B,BSCB)$
图 4 $pre(BSCP,B,BSCB)$

定义 **9**. 将 SC_2 嵌入 SC_1 的 ps,对应的基本迁移系统为 $bts(\text{embed}(SC_1,ps,SC_2))\equiv bts(\text{and}(SC_1,pre(SC_1,ps,SC_2)))$,时序语义为 $bs\_TLF(\text{embed}(SC_1,ps,SC_2))\equiv bs\_TLF(SC_1)\wedge bs\_TLF(pre(SC_1,ps,SC_2))$.

可见 $\text{embed}(SC_1,ps,SC_2)$仍保持上层 SC_1 的语义,只是对被嵌入的 SC_2 的语义进行约束,使之能够表示整个 SC_2 不活动的情况,并通过 ps 的进入退出事件建立了它与 SC_1 的关联.

例如: $bs\_TLF(\text{embed}(BSCP,B,BSCB))\equiv bs\_TLF(BSCP)\wedge bs\_TLF(pre(BSCP,B,BSCB))$

$bs\_TLF(SCR)\equiv bs\_TLF(\text{and}(\text{embed}(BSCP,B,BSCB),BSCQ))$

$\equiv bs\_TLF(BSCP)\wedge bs\_TLF(pre(BSCP,B,BSCB))\wedge bs\_TLF(BSCQ)$.

定理 **2**. 关于时序语义有如下结论:

1) 若 ps_1 是 SC_1 的基本状态,ps_2 是 SC_2 的基本状态,则

$bs\_TLF(\text{embed}(\text{embed}(SC_1,ps_1,SC_2),ps_2,SC_3))\equiv bs\_TLF(\text{embed}(SC_1,ps_1,\text{embed}(SC_2,ps_2,SC_3)))$;

2) 若 ps_1,ps_2 是 SC_1 的基本状态,则

$bs\_TLF(\text{embed}(\text{embed}(SC_1,ps_1,SC_2),ps_2,SC_3))\equiv bs\_TLF(\text{embed}(\text{embed}(SC_1,ps_2,SC_3),ps_1,SC_2))$;

3) 若 ps 是 SC_1 的基本状态,则

$bs\_TLF(\text{embed}(\text{and}(SC_1,SC_2),ps,SC_3))\equiv bs\_TLF(\text{and}(\text{embed}(SC_1,ps,SC_3),SC_2))$.

这一结论表明:在我们的语义定义下,利用 Statecharts 为系统建模的组合顺序是无关的.建模时可结合自顶向下的逐步求精和自底向上的组合方法,而不破坏最终模型语义.一旦接口(可观察变量)明确,各子系统可并行开发,再作组装.由于用时序逻辑表示语义,可直接得到如下组合证明规则.

定理 **3**. 设 SC_1 和 SC_2 是 Statecharts,p 和 q 是时序逻辑公式表示的性质,"$SC\|=p$"表示系统 SC 满足性质 p,则有:

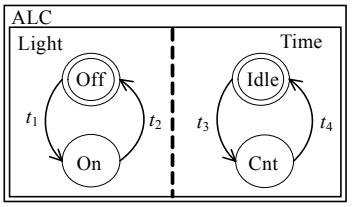
1) 若 $SC_1\|=p,SC_2\|=q$,则 $\text{and}(SC_1,SC_2)\|=p\wedge q$;

2) 若 $SC_1\|=p,pre(SC_1,ps,SC_2)\|=q$,则 $\text{embed}(SC_1,ps_1,SC_2)\|=p\wedge q$.

而且,关于 $SC\|=p$ 是否成立也可表示为逻辑蕴涵式 $bs\_TLF(SC)\Rightarrow p$,在时序逻辑框架下推导.

上述语义描述所采用的是通常的时序逻辑公式表示,只需对时序逻辑公式进行等价变换,就可将它们转换成 XYZ/E 公式.我们用一个完整的实例来说明我们的语义在性质验证方面的优势.

例 1:如图 5 所示,SC_ALC 是一个表示灯光自动开关控制系统的 Statechart.系统由灯光控制(Light)和时间控制(Timer)组成,控制灯亮 5 分钟后自动关闭.Light 的缺省状态是关闭(Off),Timer 的缺省状态是闲置(Idle).按开关(Pressed)事件发生后,灯亮(On)并发出开始计时(set)事件,触发 Timer 的 t_3 转换,Timer 转入计时(Cnt)状态并发出重置(reset)事件给环境(物理时钟).灯亮 5 分钟后,环境发出 timeout 事件触发 Timer 的 t_4 转换,Timer 转入 Idle 状态并发出关闭(sw_off)事件,该事件触发 Light 的 t_2 的执行,灯光关闭(Off).这是 SC_ALC 一次定时控制的过程.我们将计时作为环境的工作,reset 事件通知环境开始计时,时间到时,环境用事件 timeout 通知系统.



t_1: pressed/set,t_2: sw_off
t_3: set/reset, t_4: timeout/sw_off

Fig.5 Automatic light control system SC_ALC

图 5 灯光自动开关控制系统 SC_ALC

系统的抽象描述:SC_Light:S={Off,On},s_0=Off,ES={pressed,set,sw_off},T={t_1,t_2};SC_Timer:S={Idel,Cnt},s_0=Idel,ES={set,reset,timeout,sw_off},T={t_3,t_4};SC_ALC=and(SC_Light,SC_Timer),即,SC_ALC 是 SC_Light 和 SC_Timer 的并发组合.

时序语义:

bs_TLF(SC_Light)≡LB_1=Off

$\wedge\Box(LB_1=\text{Off}\wedge occ(\text{pressed})\wedge\$OLB_1=\text{On}\wedge\$Ose_1=\{\text{set}\}\vee LB_1=\text{Off}\wedge\neg occ(\text{pressed})\wedge\$OLB_1=\text{Off}\wedge\$Ose_1=\varnothing$

$\vee LB_1=\text{On}\wedge occ(\text{sw\_off})\wedge\$OLB_1=\text{Off}\wedge\$Ose_1=\varnothing\vee LB_1=\text{On}\wedge\neg occ(\text{sw\_off})\wedge\$OLB_1=\text{On}\wedge\$Ose_1=\varnothing)$

bs_TLF(SC_Timer)≡LB_2=Idel

$\wedge\Box(LB_2=\text{Idel}\wedge occ(\text{set})\wedge\$OLB_2=\text{Cnt}\wedge\$Ose_2=\{\text{reset}\}\vee LB_2=\text{Idel}\wedge\neg occ(\text{set})\wedge\$OLB_2=\text{Idel}\wedge\$Ose_2=\varnothing$

$\vee LB_2=\text{Cnt}\wedge occ(\text{timeout})\wedge\$OLB_2=\text{Idel}\wedge\$Ose_2=\{\text{sw\_off}\}\vee LB_2=\text{Cnt}\wedge\neg occ(\text{timeout})\wedge\$OLB_2=\text{Cnt}\wedge\$Ose_2=\varnothing)$

bs_TLF(SC_ALC)≡bs_TLF(SC_Light)∧bs_TLF(SC_Timer)

要证明系统满足性质:当系统处于缺省状态时,一旦开关按下,灯亮并且开始计时.只需证如下蕴涵式成立

$bs\_TLF(\text{SC\_ALC})\Rightarrow(\Box(LB_1=\text{Off}\wedge LB_2=\text{Idel}\wedge occ(\text{pressed})\Rightarrow\$O\$O(LB_1=\text{On}\wedge LB_2=\text{Cnt}\wedge occ(\text{reset}))))$.

4 讨 论

Statecharts 语义定义中的一个关键选择是,一步动作的效果作用在当前步[7,8]还是下一步[9,10].本文采用后者.这一选择实际上严格规定了事件间的因果关系(causality).并且,由于转换上的触发事件和生成事件不在同一步起作用,关于$\neg e/e$ 类标号产生的不一致性问题也不存在.我们的并发组合语义是逻辑合取,所以,满足条件并且不在同一异或状态中的转换总是都被执行(语义贪心性质).例如,在图 1 的 *SCR* 中,系统在状态 G 和 E 时事件 e_4 发生,则 t_8 和 t_6 同时执行.

每一对(u,\$Ou)表示的是一个系统步(step),每两步之间是否有时间间隔取决于系统执行环境与时间模型.时间模型有两种:一种是同步时间模型(synchronous time model),在每一时间单位内只允许走一步,外部事件在每一时刻都可能触发系统;一种是异步时间模型(asynchronous time model),在一个时刻可能走多步,这些步是由外部事件的一次触发引起的转换链(外部触发的转换生成的事件又触发另一转换,直至系统不再生成新的事件为止),这样的一个反应链称为一个超步(super-step).在一个超步结束前,新的外部事件对系统不起作用.两种时间模型可在环境约束上表示,设 $newevents\subseteq 2^{ES}$ 是一个不断变化的事件集常量.

同步时间模型约束为 $SynENV\equiv_{def} env=\varnothing\wedge\Box(\$Oenv=newevents)$.

在同步时间模型下,将 Statechart SC 与环境并发而成的闭系统时序语义为 $bt\_TLF(SC)\wedge SynENV$.

异步时间模型约束为 $AsynENV\equiv_{def} se=\varnothing\wedge env=\varnothing\wedge\Box(se=\varnothing\wedge env=newevents\vee se\neq\varnothing\wedge env=\varnothing)$.

在异步时间模型下,每步都要观察系统生成的事件集是否为空,只有当系统生成事件集为空时,环境事件才可能不为空.将 Statechart SC 与环境并发而成的闭系统时序语义为 $bt\_TLF(SC)\wedge AsynENV$.

一般认为,Statecharts 满足完美同步假设(perfect synchrony hypothesis)——对于外部触发事件,系统即时(与外部触发同一时刻)地给予反应.这实际上是假设系统的执行速度无限快,使多个步的执行可在一个时刻完成.如果对环境采用异步时间模型约束,我们的语义满足这一假设,而采用同步时间模型约束则不然.

不同层次状态之间的转换(interlevel transitions)破坏了组合性,所以我们考虑的 Statecharts 子集中不包含 interlevel 转换.本文对 Statecharts 的实时性只考虑了:转换条件一旦满足并且无冲突就必须马上执行.若要更多地考虑对系统时间方面的描述,就要在环境约束中显式地定义全局时钟变量.在我们的语义定义中,事件都是全局事件,但从可组合的角度考虑,能够定义局部事件比较理想.这就要增加一个隐藏/封闭(hide-closure)操作,将事件设置为局部事件.关于时钟变量和局部事件,我们将在下一步工作中加以考虑.

5 相关工作比较

关于 Statecharts 的形式化语义研究有很多[7−22],并已衍生出多种变型[23].UML 状态机也是它的一种变型.UML 标准确立后,关于 UML 状态机语义的讨论也多了起来[24−27].文献[7]是最早的关于 Statecharts 形式化语义的研究,提供了一个形式化语法和操作语义.文献[8,14,15]讨论了 Statecharts 所应有的特性以及如何定义语义可使之具备这些特性,其中文献[14]定义了一个指称语义,文献[8]定义了一个操作语义和一个声明语义(与指称语义的区别是不具有组合性),并证明二者是一致的.文献[11]是最早的组合语义,它提出了可用于推导 Statecharts 安全性和活性的组合公理,并且定义了它的一个指称语义,以证明该组合公理的可靠性和完备性.文献[12,16−19,24−26]等基于标号转移系统或 Kripke 结构,定义结构化操作语义,文献[27,28]又进一步将操作语义用于 Statecharts 的模型检测.用其他形式化语言定义 Statecharts 语义的有:文献[20]用 Z 语言;文献[13,21]用进程代数;文献[22]用直觉逻辑等.

我们的工作主要受到文献[11,29]的启发.文献[11]为 Statecharts 定义了一个指称语义,并定义了一个一阶逻辑语言作为 Statecharts 的性质描述语言,建立了一个 Statecharts 的组合公理系统.在文献[11]的指称语义中,基本元素是带入边和出边的,也称为基本 Statecharts 的一种中间实体,它可能是状态,也可能是完整的 Statechart,或是组合了一半的仍未形成一个完整 Statechart 的中间对象,概念上比较复杂,组合操作也比较多.我们的基本元素是不含层次和并发的完整的 Statecharts,概念上比较简单,组合操作也只有并发和嵌入两个.而且,用时序逻辑公式表示反应系统性质较之一阶逻辑更加直观和简洁.文献[11]的步语义是超步语义,每个转换生成的事件在同一(超)步起作用.我们的步语义与文献[9,10]相同,每个转换生成的事件在下一步起作用.可以说,文献[11]的语义可组合性是语句一级的,自底向上的;我们的语义组合是模块(或系统)一级的,组合顺序不影响最终系统的语义.从软件工程角度来看,从模块级定义组合语义更合适,因为软件开发过程中涉及到的分解与组合的单位一般是模块而不是语句.

文献[29]在文献[11]的语义基础上建立了 Statecharts 与时序逻辑语言 FNLOG 的语义联系,集成二者以期较全面地表示实时反应系统的规范.Statecharts 作为行为规范语言,FNLOG 作为功能规范语言,建立 Statecharts 到 FNLOG 的语义等价转换.一个 Statechart SC 有一个语义等价的 FNLOG 功能规范 F,通过证明 F 是否满足某个性质 p,就证明了 SC 是否满足某个性质 p.与文献[29]类似,我们用一个 XYZ/E 公式作为 Statechart 的等价表示.FNLOG 只能表示静态的功能规范,无法表达系统行为;而 XYZ/E 对 LTL 作了扩充,不仅能够表示系统性质,还能表示状态转换,从而也就具有了描述系统行为的能力.所以,在我们的语义定义中,XYZ/E 不仅能够表示 Statecharts 所应满足的性质,还能精确地表示 Statecharts 的行为.

6 结 语

本文在基本迁移系统上解释 Statecharts 的语义,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 Statechart 的时序语义.建立了图形建模语言 Statecharts 与线性时序逻辑语言 XYZ/E 的语义联系,为对 Statecharts 进行严格形式化推导提供了基础.以不含层次和并发的基本 Statecharts 为基本语义元素,再递归地定义两个组合操作(并发和嵌入)的语义,得到完整的 Statecharts 语义.这样的语义在模块(或系统)级可组合,组合顺

序不影响最终系统的语义.求精的语义保持问题,可直接从语义定义得到保证.由于 XYZ/E 既可表示系统的性质又可表示系统的行为,就使得考察 Statecharts 的行为是否满足某些性质的问题变成考察逻辑蕴涵式是否为真的问题.我们下一步的工作是建立起以逐步求精为指导思想,以 Statecharts 和 UML 活动图等图形语言为前端建模语言,以 XYZ/E 为严格的形式化语义基础[5]的体系结构描述框架.

致谢 中国科学院软件研究所的李广元、张文辉研究员对本文工作提出了很多有益的建议,本文的评审专家也为本文的完善提出了许多宝贵意见,在此表示感谢!

References:

[1] Harel D. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 1987,8(3):231−274.

[2] Manna Z, Pnueli A. The Temporal Logic of Reactive and Concurrent System: Specification. New York: Springer-Verlag, 1992.

[3] Tang ZS, *et al*. Temporal Logic Programming and Software Engineering. Beijing: Science Press, 2002 (in Chinese).

[4] Kesten Y, Manna Z, Pnueli A. Temporal verification of simulation and refinement. In: de Bakker JW, de Roever WP, Rozenberg G, eds. Proc. of the Decade of Concurrency. LNCS 803, Berlin: Springer-Verlag, 1994. 273−346.

[5] Zhu XY, Tang ZS. A temporal logic-based software architecture description language XYZ/ADL. Journal of Software, 2003,14(4): 713−720 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/713.htm

[6] Zhu XY, Tang ZS. A temporal logic semantics for UML activity diagrams. Journal of Computer Research and Development, 2005,42(9):1478−1484 (in Chinese with English abstract).

[7] Harel D, Pnueli A, Schmidt JP, Sherman R. On the formal semantics of Statecharts. In: Pnueli A, ed. Proc. of the 2nd IEEE Symp. on Logic in Computer Science. Washington: Computer Society Press of the IEEE, 1987. 56−64.

[8] Pnueli A, Shalev M. What is in a step: On the semantics of Statecharts. In: Ito T, Mayer AR, eds. Proc. of the Symp. on Theoretical Aspects of Computer Science. LNCS 526, Berlin: Springer-Verlag, 1991. 244−264.

[9] Harel D, Naamad A. The STATEMATE semantics of Statecharts. ACM Trans. on Software Engineering and Methodology, 1996, 5(4):293−333.

[10] Leveson NG, Heimdahl MPE, Hildreth H, Reese JD. Requirements specification for process-control systems. IEEE Trans. on Software Engneering, 1994,20(9):684−707.

[11] Hooman JJM, Ramesh S, de Roever WP. A compositional axiomatization of Statecharts. Theoretical Computer Science, 1992, 101(2):289−335.

[12] Uselton A, Smolka SA. A compositional semantics for Statecharts using labeled transition systems. In: Jonsson B, Parrow J, eds. Proc. of the 5th Int'l Conf. on Concurrency Theory. LNCS 836, Berlin: Springer-Verlag, 1994. 2−17.

[13] Uselton A, Smolka SA. A process algebraic semantics for Statecharts via state refinement. In: Olderog ER, ed. Proc. of the IFIP Working Conf. on Programming Concepts, Methods and Calculi. North-Holland: Elsevier Science Publishers, 1994.

[14] Huizing C, Gerth R, de Roever WP. Modeling Statecharts behavior in a fully abstract way. In: Dauchet M, Nivat M, eds. Proc. of the 13th Colloquium on Trees in Algebra and Programming. LNCS 299, Berlin: Springer-Verlag, 1988. 271−294.

[15] Huizing C, de Roever WP. Introduction to design choices in the semantics of Statecharts. Information Processing Letters, 1991,(37): 205−213.

[16] Maggiolo-Schettini A, Peron A, Tini S. Equivalences of Statecharts. In: Sassone M, ed. Proc. of the Concurrency Theory: the 7th Int'l Conf. LNCS 1119, Berlin: Springer-Verlag, 1996. 687−702.

[17] Mikk E, Lakhnech Y, Siegel M. Hierarchical automata as model for Statecharts. In: Shyamasundar R, Euda K, eds. Proc. of the 3rd Asian Computing Science Conf. LNCS 1345, Berlin: Springer-Verlag, 1997. 181−196.

[18] von der Beeck M. A concise compositional Statecharts semantics definition. In: Bolognesi T, Latella D, eds. Proc. of the Formal Techniques for Distributed System Development, FORTE/PSTV 2000. Boston: Kluwer Academic Publishers, 2000.

[19] Lüttgen G, von der Beeck M, Cleaveland R. A compositional approach to Statecharts semantics. In: Rosenblum DS, ed. Proc. of the 8th Foundations of Software Engineering (FSE-8). New York: ACM Press, 2000. 120−129.

[20] Mikk E, Lakhnech Y, Petersohn C, Siegel M. On formal semantics of Statecharts as supported by STATEMATE. In: Duke DJ, Evans AS, eds. Proc. of the 2nd BCS-FACS Northern Formal Methods Workshop. Berlin: Springer-Verlag, 1997.

[21] Lüttgen G, vonder Beeck M, Cleaveland R. Statecharts via process algebra. In: Baeten JCM, Mauw S, eds. Proc. of the Concurrency Theory, 10th Int'l Conf. LNCS 1664, Berlin: Springer-Verlag, 1999. 399−414.

[22] Lüttgen G, Mender M. The intuitionism behind Statecharts steps. ACM Trans. on Computational Logic, 2002,3(1):1−38.

[23] von der Beeck M. A comparision of Statechart variants. In: de Roever WP, Langmaack H, Vytopil J, eds. Proc. of the Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS 863, Berlin: Springer-Verlag, 1994. 128−148.

[24] Latella D, Majzik I, Massink M. Towards a formal operational semantics of UML Statechart diagrams. In: Ciancarini P, Fantechi A, Gorrieri R, eds. Proc. of the 3rd Int'l Conf. on Formal Methods for Open Object-Oriented Distributed Systems. Boston: Kluwer Academic Publishers, 1999. 15−18.

[25] von der Beeck M. Formalization of UML-Statecharts. In: Gogolla M, Kobryn C, eds. Proc. of the Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th Int'l Conf. LNCS 2185, Berlin: Springer-Verlag, 2001. 406−421.

[26] Jiang H, Lin D, Xie XR. The formal semantics of UML state machine. Journal of Software, 2002,13(12):2244−2250 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/2244.pdf

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

[28] Mikk E, Lakhnech Y, Siegel M. Towards efficient modelchecking Statecharts: A Statecharts to promela complier. In: Langerak R, ed. Proc. of the 3rd Int'l SPIN Workshop. LNCS 1516, Berlin: Springer-Verlag, 1997.

[29] Sowmya A, Ramesh S. Extending Statecharts with temporal logic. IEEE Trans. on Software Engineering, 1998,24(3):216−231.

附中文参考文献:

[3] 唐稚松,等.时序逻辑程序设计与软件工程.北京:科学出版社,2002.

[5] 朱雪阳,唐稚松.基于时序逻辑的软件体系结构描述语言 XYZ/ADL.软件学报,2003,14(4):713−720. http://www.jos.org.cn/1000-9825/14/713.htm

[6] 朱雪阳,唐稚松. UML 活动图的时序逻辑语义.计算机研究与发展,2005,42(9):1478−1484.

[26] 蒋慧,林东,谢希仁.UML 状态机的形式语义.软件学报,2002,13(12):2244−2250. http://www.jos.org.cn/1000-9825/13/2244.pdf

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

朱雪阳(1971－),女,福建莆田人,博士,主要研究领域为软件建模语言,软件体系结构与形式化方法.

唐稚松(1925－),男,研究员,博士生导师,中国科学院院士,主要研究领域为计算机科学理论,软件工程.