UML Statecharts 的切片模型检验方法

董 威,王 戟,齐治昌
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 统一建模语言 UML 已被广泛应用于软件设计和开发中,而验证 UML 模型是否满足关键的性质需求成为一个重要问题.由于空间爆炸和语义的复杂性,对 Statecharts 进行模型检验受到软件规模和设计精化程度的制约.本文在用扩展层次自动机(EHA)结构化的表示 UML Statecharts 后,通过分析 EHA 中存在的层次、并发和事件同步等特征定义了一组依赖关系.对于由状态和迁移组成的切片准则,给出对 EHA 进行切片的算法.该算法能保证切片后的 EHA 与原来的 Statecharts 对性质具有相同的可满足性,且删除了与被验证性质无关的层次和并发状态,缓解了空间爆炸问题.

关键词: UML Statecharts; 模型检验; 切片
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2082-07

Slicing UML Statecharts for Model Checking

DONG Wei, WANG Ji, QI Zhi-chang
(*School of Computer Science, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: Unified Modeling Language (UML) has been widely used in software development. Verifying whether a UML model meets the required properties has become a challenge. Model checking is an important technology of automatic verification to ensure the correctness of designed models. Because of space explosion and complicated semantics, model checking Statecharts are restricted by the software scale and the refinement degree of design. In this paper, UML Statecharts are structurally expressed by Extended Hierarchical Automaton (EHA). A set of dependence relations is specified after analyzing the characteristics such as hierarchy, concurrency and synchronization in EHA. The algorithm of slicing EHA based on the slicing criterion which is composed by states and transitions is presented. The sliced EHA and the original Statecharts are equivalent to the property. The algorithm removes the hierarchies and concurrent states which are irrelevant with the property, and reduce the state space efficiently.

Key words: UML Statecharts; model checking; slice

1 引言

统一建模语言 UML(Unified Modeling Language)[1]是一种描述能力强大的可视化建模语言,它提供多种图元从不同角度和应用层次刻画系统特性以及复杂的运行环境,为软件开发人员提供了从系统需求分析、设计到实现的有力支持.UML Statecharts 是 Harel's Statecharts[2]的一种变体,刻画了对象在其生命周期中的行为和状态变迁,在软件分析和设计建模过程中占有重要的地位,而对其进行正确性验证以判断设计规范是否满足目标需求也成为一个关键问题.

模型检验(model checking)是一种重要的自动验证技术,主要通过显式状态搜索或隐式不动点计算来验证有穷状态并发或实时系统的模态/命题性质.近来已出现了一些对 Statecharts 进行模型检验的研究[3~6].由于 UML Statecharts 本身具有并发和层次等特征,所以在模型检验时也面临空间爆炸问题.这些问题在大规模软件系统的设计规范经过递增和精化到达详细设计阶段时尤其突出(如飞行控制系统[7]).程序切片是从程序中根据切片准则提取与特定计算相关语句的方法.近来,已有一些使用程序切片思想缩减模型检验所需状态空间的研究.文献[8]提出如何使用控制依赖和数据依赖对顺序程序切片进行模型检验,但该方法不能用于并发程序.文献[9]研究了如何对 Promela 代码进行静态切片以用于模型检验、模拟和协议理解,但没给出形式化的切片方法.文献[10]研究怎样对多线程 JAVA 程序切片以进行模型检验,其主要关注的是使用锁的线程同步语句引入的依赖关系,而不考虑信道、非确定选择和并发分支和接合.文献[7]对 RSML 规范语言中的状态图用切片思想进行缩减,目标是为了更好的分析和理解设计规范.该方法假定不存在变量,动作只是产生事件,而且只定性的描述了控制依赖和数据依赖关系,没有给出切片算法.

收稿日期:2002-06-06;修回日期:2002-10-19
基金项目:国家自然科学基金(No.69973051,No.90104007);国家 863 项目 (No.2001AA113202);霍英东青年教师基金(No.71064)

本文把切片思想与 UML Statecharts 的模型检验相结合.在用扩展层次自动机(Extended Hierarchical Automaton, EHA)结构化的表示 UML Statecharts 后,根据 EHA 中存在的层次、并发和同步等特征定义了一组依赖关系.根据这些依赖关系,针对由状态和迁移组成的切片准则给出了切片算法.当按一定规则从被验证的线性时态逻辑(Linear Temporal Logic, LTL)性质中提取切片准则时,切片前后的 EHA 对于要验证的性质是等价的.切片后的模型抛弃了与被验证性质无关的层次和并发状态,缩减了状态空间的规模和生成相应 LTS(Labeled Transition System)的开销.该方法还有助于更好的分析和理解设计模型.切片方法应用在生成系统全局状态图之前,因此引入的额外开销并不大,而且它独立于模型检验算法,可以和 on-the-fly 或 partial-order 等其他模型检验优化技术结合使用.

本文第 2 节描述如何用 EHA 结构化的表示 UML Statecharts;第 3 节定义了 EHA 中存在的一组依赖关系;对 EHA 进行切片的算法在第 4 节给出;文章最后给出结论.

2 用 EHA 表示 UML Statecharts

UML Statecharts 的语法和语义在文献[11]中都是以精确的自然语言描述.文献[5,12]给出了形式化定义.图 1 是一个 TV 控制器的例子,其中 *TV_SYS* 和 *ON* 是 *AND*-状态,*TV*、*USER*、*OFF*、*IMAGE* 和 *SOUND* 是 *OR*-状态,其余为 *BASIC*-状态.*t*1 和 *t*2 是层间迁移.

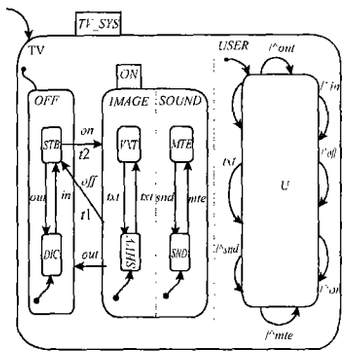

图 1 TV 控制器的 UML Statecharts

根据语义解释执行 UML Statecharts 时需要一个动作语言,UML 标准中未对其进行定义.我们定义了一个基本的动作语言,能够对整型和布尔变量进行赋值和算术运算;可以定义 = =、<、>、< > 等条件表达式,并能进行逻辑运算;可以形式^ *obj*.*e* 向对象 *obj* 发送事件 *e*(^ 表示发送给任何可以接受 *e* 的对象).

为了便于建立 UML Statecharts 到 LTS 之间的映射,文献[13]把 EHA 作为 UML Statecharts 的中间模型.文献[6]扩充了 EHA 的定义以包含 UML Statecharts 的一些主要特点和基本功能,并对操作语义也进行了相应的改进.后面可以看到这种结构化的图形语言也使得依赖关系和切片算法定义起来更加方便自然.EHA 由简单的顺序自动机组成,状态通过精化函数映射到一组对其进行细化的(并发)顺序自动机.

定义 1 顺序自动机:顺序自动机 A 是一个 4 元组 $(\sigma_A, s_A^0, \lambda_A, \delta_A)$,$\sigma_A$ 是有穷的状态集合,s_A^0 是初始状态,λ_A 是有穷的迁移标记集合,$\delta_A \subseteq \sigma_A \times \lambda_A \times \sigma_A$ 是迁移关系.

定义 2 扩展层次自动机(EHA):EHA H 是一个 5 元组 (F, E, ρ, A_0, V),F 是一个有穷的顺序自动机集合,$\forall A_1, A_2 \in F, \sigma_{A_1} \cap \sigma_{A_2} = \emptyset$;$E$ 是有穷的事件集合;V 是变量集合;$\rho: \bigcup_{A \in F} \sigma_A \to 2^F$ 是精化函数,给出 F 的一个树型结构满足:(1)存在一个唯一的根自动机 $A_0 \in F$,使得不存在 $s \in \bigcup_{A \in F} \sigma_A$、$A_0 \in \rho(s)$;(2)每个非根自动机恰有一个父状态:对于 $\forall A \in F \setminus \{A_0\}$,$\exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} \sigma_{A'}$,$A \in \rho(s)$;(3)不存在环路:$\forall S \subseteq \bigcup_{A \in F} \sigma_A$,$\exists s \in S$,$S \cap (\bigcup_{A \in \rho(s)} \sigma_A) = \emptyset$.

把 UML Statecharts 转换为 EHA 的过程中,层间迁移提升到它离开和进入的最高层状态之间,从而变为非层间迁移.为了不改变语义,需要对迁移标记进行扩充,添加受限源状态 *sr* 和确定目标状态 *td* 这两个集合.*sr* 将迁移的使能限制在源状态下的一个格局中,*td* 决定了迁移使系统进入目标状态时有哪些子状态也同时进入.

图 2 是图 1 相对应的 EHA,其中 $F = \{TV\_SYS, TV, USER, POWER, IMAGE, SOUND\}$,$\rho(S) = \{TV, USER\}$,$\rho(OFF) = \{POWER\}$,$\rho(ON) = \{IMAGE, SOUND\}$.迁移 $t1$ 的 *td* 为 $\{STB\}$,$t2$ 的 *sr* 也为 $\{STB\}$.图中粗框的状态为其顺序自动机的初始状态.

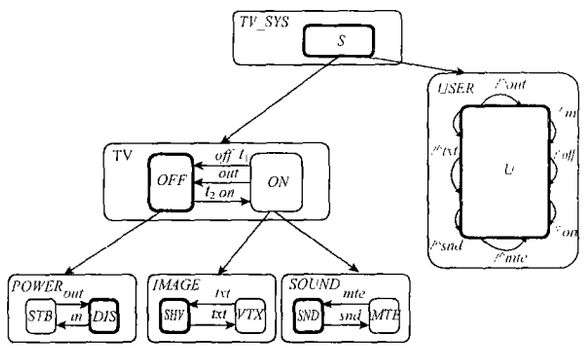

图 2 TV 控制器 Statecharts 对应的 EHA

层次自动机的全局状态由格局来表示,它由包含的顺序自动机的某些局部状态组成.

定义 3 格局:H 的一个格局是一个集合 $Conf \subseteq S(A_0)$ 使得:1) $\exists_1 s \in \sigma_{A_0}$ 满足 $s \in Conf$;2) $\forall s, A$,若 $s \in Conf$ 且 $A \in \rho(s)$,则 $\exists_1 s' \in A, s' \in Conf$.

EHA 的操作语义可以用一个 LTS 定义,它是通过迁移相关联的一组状态.在 Statecharts 中,操作语义中的状态被称为状况(status),每个状况由格局和当前的环境组成[6].

定义 4 EHA 的操作语义:EHA H 的操作语义是一个 LTS $Ts = (S, s_0, L, \to)$,其中 S 是状况集合,s_0 是初始状况,$L: S \to 2^{AP}$ 为每个状况标注一组原子命题,$\to \subseteq S \times S$ 是迁移关系.

Ts 中的一个迁移是 Statecharts 中通过选择最大无冲突迁移集所完成的一个 RTC 步[11].关系 $\to$ 通过一个演绎系统进行定义,它由三条操作语义规则给出:前进规则(Progress Rule),组合规则(Composition Rule)和暂停规则(Stuttering Rule).该语义基于开放模型(包括了封闭模型)给出,详细内容可参见文献[6].在对 EHA 进行切片时,我们局限在封闭模型中考虑,因为环境的行为和依赖关系是无法预料的.

3 EHA 中的依赖关系

在对顺序程序进行切片的方法中,只需考虑控制依赖和数据依赖[14].在对并发程序进行切片的方法[15]中,由于包含多个控制流和数据流,所以增加了同步依赖、通信依赖、选择依赖等关系.而层次自动机更复杂一些,它不仅有并发、通信等机制,还具有层次结构,迁移和状态的语法成分比普通程序语言的一条语句更为复杂,并且执行语义需要选择最大无冲突迁移集等等.这些使得要以每个层次自动机中的状态和迁移为单元进行切片变得异常困难,而且可能造成切片过程需要较大的时空开销.我们考虑以状态和迁移为单元定义依赖关系,而以顺序自动机为单位进行切片,删除 EHA 中与要验证性质无关的顺序自动机.

根据后文需要首先定义如下一些函数和记号.假定 $H=(F,E,\rho,A_0,V)$是一个 EHA. $Src,Tar:\bigcup_{A\in F}\delta_A\to\bigcup_{A\in F}\sigma_A$,对于 $t=(s,l,s')\in\delta_A(A\in F)$, $Src(t)=s$, $Tar(t)=s'$.每个状态中的变量可以分为三类:引用变量、更新变量和内部变量.状态 s 的引用变量用 $s.UV$ 表示,它包括所有在 s 之下(包括 s 嵌套子自动机中的状态和迁移)的动作中被引用、并可以在该状态所代表的子层次自动机之外定义和引用的变量. s 的更新变量用 $s.DV$ 表示,它是在 s 之下的动作中被赋值、并在该状态所代表的子层次自动机之外可以定义和引用的变量集合. $s.UV$ 和 $s.DV$ 分别可以看作模块 s 的输入变量和输出返回变量,相对 s 来说都是全局变量.内部变量是只在 s 之下定义和使用的局部变量,不为 s 之外元素所见,但它可能是 s 的子自动机中某些状态的引用或更新变量.类似的, $t.UV$ 表示在迁移 t 的动作中引用的变量, $t.DV$ 表示在迁移 t 的动作中被赋值的变量.此外,用 $t.CV$ 表示在迁移的使能条件中引用的变量.需要注意的是,一个变量可能即是某个状态的引用变量,也是其更新变量.

每个状态中的事件也可分为三类. $s.TE$ 包括所有在状态 s 之外产生、且被作为 s 之下某个迁移激发事件的事件. $s.GE$ 包括所有在 s 下在某个动作产生、且被作为 s 之外某些迁移激发事件的事件集合.此外还有 s 的内部事件,它由 s 之下的动作产生,并只使能 s 之下的迁移,对外是不可见的.类似的, $t.TE$ 表示由迁移 t 的激发事件组成的单事件集合, $t.GE$ 表示 t 的动作中产生的事件集合.

非基本状态间的迁移可以看作从一个顺序自动机出口状态到另一个顺序自动机入口状态的迁移.我们假定一个迁移的激发事件只能由与该迁移所在顺序自动机并发的自动机中的动作产生,这符合实际情况,因为事件是用来同步控制的.

定义 5　路径(Path):顺序自动机 $A\in F$ 中的一条路径是一个状态和迁移序列 $(s_1,t_1,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1},s_k)$, $s_i\in\sigma_A$, $t_i\in\delta_A$,且 $Src(t_i)=s_i$, $Tar(t_i)=s_{i+1}$, $1\leqslant i<k$.

定义 6　顺序数据依赖(Sequence Data-dependence,$\to_{sdd}$):如果 $A\in F$ 且 $u,v\in\sigma_A$, $r,t\in\delta_A$,

(1) $u\to_{sdd}v$ 当且仅当在 A 中存在路径 $P:(s_1=v,t_1,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1},s_k=u)$,满足 $(v.DV\cap u.UV)-((\bigcup_{1<i<k}s_i.DV)\cup(\bigcup_{1\leqslant i<k}t_i.DV))\neq\varnothing$.

(2) $r\to_{sdd}v$ 当且仅当在 A 中存在路径 $P:(s_1=v,t_1,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1}=r,s_k)$,满足 $(v.DV\cap r.UV)-((\bigcup_{1<i<k}s_i.DV)\cup(\bigcup_{1<i<k-1}t_i.DV))\neq\varnothing$.

(3) $u\to_{sdd}t$ 当且仅当在 A 中存在路径 $P:(s_1,t_1=t,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1},s_k=u)$,满足 $(t.DV\cap u.UV)-((\bigcup_{1<i<k}s_i.DV)\cup(\bigcup_{1<i<k-1}t_i.DV))\neq\varnothing$.

(4) $r\to_{sdd}t$ 当且仅当在 A 中存在路径 $P:(s_1,t_1=t,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1}=r,s_k)$,满足 $(t.DV\cap r.UV)-((\bigcup_{1<i<k}s_i.DV)\cup(\bigcup_{1<i<k-1}t_i.DV))\neq\varnothing$.

非形式化的说,若一个元素 x(这里元素可以是状态或迁移,下文相同)中计算得到的某个变量值对在元素 y 中计算的某个变量值可能有直接影响,且它们在同一个顺序自动机中,那么 y 顺序数据依赖 x.图 3 是一个 EHA 示例,其中状态 s_3 顺序数据依赖 s_1,因为 d 的取值直接受 b 的影响.从 s_2 到 s_1 的迁移顺序数据依赖 s_3,因为输出 d 在 s_3 中计算得到.

定义 7　并发数据依赖(Parallel Data-dependence,$\to_{pdd}$):如果 $A,B\in F$ 且 $u\in\sigma_A$, $r\in\delta_A$, $v\in\sigma_B$, $t\in\delta_B$,若存在 $C\in F$ 和 $s\in\sigma_C$ 满足 $A,B\in\rho(s)$, $u\to_{pdd}v$(或 $u\to_{pdd}t$,或 $r\to_{pdd}v$,或 $r\to_{pdd}t$)当且仅当 $u.UV\cap v.DV\neq\varnothing$(或 $u.UV\cap t.DV\neq\varnothing$,或 $r.UV\cap v.DV\neq\varnothing$,或 $r.UV\cap t.DV\neq\varnothing$).

非形式化的说,如果在元素 x 中计算得到的某个变量值对在元素 y 中计算的某个变量值有直接影响,且 x 和 y 分别位于的两个并发的自动机有相同的父状态,那么 y 并发数据依赖 x.在图 3 中, $s7$ 并发数据依赖 $s8$,因为 a 的值直接由 $s8$ 中计算的 d 确定.

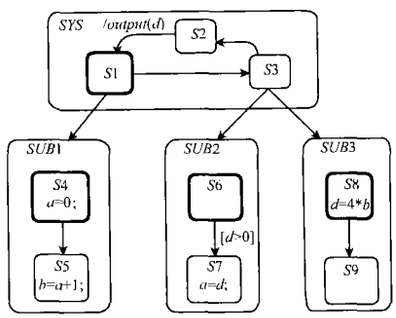

图 3　一个 EHA 例子

定义 8　同步依赖(Synchronization-dependent,$\to_{sd}$):如果 $A,B\in F$ 且 $u\in\sigma_A$, $r\in\delta_A$, $v\in\sigma_B$, $t\in\delta_B$,若存在 $C\in F$ 和 $s\in\sigma_C$ 满足 $A,B\in\rho(s)$,则 $u\to_{sd}v$(或 $u\to_{sd}t$,或 $r\to_{sd}v$,或 $r\to_{sd}t$)当且仅当 $u.TE\cap v.GE\neq\varnothing$(或 $u.TE\cap t.GE\neq\varnothing$,或 $r.TE\cap v.GE\neq\varnothing$,或 $r.TE\cap t.GE\neq\varnothing$).

非形式化的说,如果元素 x 中某个迁移激发事件由元素 y 中动作产生,且 x 和 y 分别位于的两个并发的自动机有相同的父状态,那么 x 同步依赖 y.在图 2 中,自动机 TV 中从 off 到 on 的迁移同步依赖自动机 $USER$ 中产生事件 on 的迁移.状态 OFF 同步依赖产生事件 out 和 in 的两个迁移.

定义 9　迁移控制依赖(Transition Control-dependence,$\to_{tcd}$):如果 $A\in F$ 且 $r\in\delta_A$,

(1)若 $v\in\sigma_A$,则 $r\to_{tcd}v$ 当且仅当在 A 中存在路径 $P:(s_1$

$=v,t_1,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1}=r,s_k)$,满足$(v.DV\cap r.CV)-((\cup_{1<i<k}s_i.DV)\cup(\cup_{1<i<k-1}t_i.DV))\neq\emptyset$.

(2)若 $t\in\delta_A$,则 $r\rightarrow_{tcd}t$ 当且仅当在 A 中存在路径 $P:(s_1,t_1=t,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1}=r,s_k)$,满足$(t.DV\cap r.CV)-((\cup_{1<i<k}s_i.DV)\cup(\cup_{1<i<k-1}t_i.DV))\neq\emptyset$.

3)如果 $B\in F$ 且存 $C\in F$ 和 $s\in\sigma_C$ 满足 $A,B\in\rho(s)$、$v\in\sigma_B$,$t\in\delta_B$,则 $r\rightarrow_{tcd}v$(或 $r\rightarrow_{tcd}t$)当且仅当 $t.CV\cap v.DV\neq\emptyset$(或 $r.CV\cap t.DV\neq\emptyset$).

非形式化的说,如果在元素 x 中计算得到的某个变量值对迁移 r 的使能条件真假有直接影响,那么 r 迁移控制依赖 x.在图 3 中,从 $s6$ 到 $s7$ 的迁移迁移控制依赖 $s8$.

定义 10 精化数据依赖(Refinement Data-dependence,$\rightarrow_{rdd}$):如果 $A\in F$ 且 $u\in\sigma_A,B\in\rho(u),v\in\sigma_B,t\in\delta_B$,

(1)$u\rightarrow_{rdd}v$ 当且仅当存在从 v 到 B 中的某个出口结点 e 的路径 $P:(s_1=v,t_1,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1},s_k=e)$满足$(v.DV\cap u.DV)-((\cup_{1<i\leqslant k}s_i.DV)\cup(\cup_{1\leqslant i<k}t_i.DV))\neq\emptyset$,或 $u.GE\cap v.GE\neq\emptyset$;

(2)$u\rightarrow_{rdd}t$ 当且仅当存在从 $s_1=Src(t)$ 到 B 中的某个出口结点 e 的路径 $P:(s_1,t_1=t,s_2),(s_2,t_2,s_3),\cdots,(s_{k-1},t_{k-1},s_k=e)$满足$(t.DV\cap u.DV)-((\cup_{1<i\leqslant k}s_i.DV)\cup(\cup_{1<i<k}t_i.DV))\neq\emptyset$,或 $u.GE\cap t.GE\neq\emptyset$.

非形式化的说,如果元素 x 在状态 u 的直接子自动机中,并且在 x 中计算得到的某个变量值就是该变量最终被 u 返回的值,或 x 中产生事件用来和 u 并发的某个状态同步,那么 u 精化数据依赖 x.在图 3 中,$s3$ 精化数据依赖于 $s8$,因为 $s3$ 的输出变量 d 在 $s8$ 中计算得到.

定义 11 精化控制依赖(Refinement Control-dependence,$\rightarrow_{rcd}$):如果 $A,B\in F$ 且 $u\in\sigma_A,v\in\delta_B$ 且 v 是 B 的初始状态,那么 $v\rightarrow_{rcd}u$ 当且仅当 $B\in\rho(u)$.

非形式化的说,如果状态 v 是状态 u 的一个直接子顺序自动机的初始状态,那么 v 精化控制依赖 u.在图 2 中,自动机 *TV* 中的状态 *OFF* 精化控制依赖状态 *S*,*POWER* 中的状态 *DIS* 精化控制依赖状态 *OFF*.只判断初始状态是因为同一顺序自动机的其他状态和迁移很容易从初始状态得到.

可以对 H 定义一个关系$\rightarrow d$,它是上述几种依赖关系的并集(即$\rightarrow_{sdd}$、$\rightarrow_{pdd}$、$\rightarrow_{sd}$、$\rightarrow_{tcd}$、$\rightarrow_{rdd}$和$\rightarrow_{rcd}$的并集).从依赖关系的定义可以看到,若一个状态或迁移所在的顺序自动机在 EHA 的第 n 层,那么它所依赖的元素位于的顺序自动机在 EHA 的第$(n-1)$到$(n+1)$层之间这个较小的局部范围中.

4 对 EHA 进行切片

Weiser 对顺序程序给出一个近似最少语句切片迭代算法[14],文献[15]根据并发程序的依赖关系图寻找从切片准则点可达的语句组成程序切片.为了模型检验而对 EHA 进行切片时,切片准则和算法都需要较大改动.

算法的基本思想是从切片准则开始通过依赖关系迭代找到对切片准则中的状态和迁移有影响的那些状态和迁移.最后的切片是以顺序自动机为单位的,当某个顺序自动机的一个状态或迁移被包含在切片中时,则该顺序自动机中的所有状态和迁移都应包含进来.但当其中某个状态在算法结束后没有被任何元素依赖时,该状态的子自动机和动作可以不被包含在切片中;而当某个迁移在算法结束后没有被任何元素依赖时,其动作也可以被删除.

4.1 EHA 切片的计算

由于在要验证的性质中可能出现多个状态名称和变量,因此我们采用关注多个状态和迁移的切片准则.假定 $H=(F,E,\rho,A_0,V)$是一个 EHA.

定义 12 切片准则:H 的切片准则 C 是一个二元偶$\langle\{s_1,\cdots,s_k\},\{t_1,\cdots,t_n\}\rangle$,$s_i\in S(A_0),t_i\in T(A_0)$,且 $s_i\neq s_j(1\leqslant i,j\leqslant k,i\neq j),t_i\neq t_j(1\leqslant i,j\leqslant n,i\neq j)$.

算法 1 根据 EHA 中的依赖关系迭代生成关于切片准则 C 的切片.集合 *RS* 和 *RT* 分别是要最终保留在切片中的状态和迁移;*IS* 和 *IT* 分别是需要在下一次迭代中寻找依赖关系的状态和迁移;*NS* 和 *NT* 分别是每次迭代找到的、*IS* 和 *IT* 所依赖的状态和迁移;*ES* 和 *ET* 则分别是 *NS* 和 *NT* 中新找到的元素所在的顺序自动机中其他未包含进来的状态和迁移.算法中用到两个布尔函数:$Refine_R$ 确定最终在切片中保留下来的每个元素是否保留其精化信息,比如每个状态的出口、入口动作和子自动机,或者每个迁移的动作;布尔函数 $Refine_N$ 是一个中间辅助函数.

在 Step1 的初始化中,*RS* 和 *RT* 只包含切片准则中的状态和迁移,并且 $Refine_R$ 的值为真表明对它们进行精化的语法成分要保留.而此时 *ES* 和 *ET* 中的元素是那些与 *RS* 和 *RT* 中元素在一个顺序自动机中、但不包含在切片准则中的状态和迁移,它们目前不需要精化.Step2 找到 *IS* 和 *IT* 中元素所依赖的状态和迁移.如果一个状态 s 或迁移 t 不需要精化,那么对于 s 只在它是初始状态的情况下寻找它所精化控制依赖的状态,而对于 t 只寻找它所迁移控制依赖或同步依赖的元素.在 Step3 中,把 *NS* 和 *NT* 中元素的 $Refine_N$ 置为 True;找到与 *RS* 和 *RT* 中新元素在相同顺序自动机中、且未被包含进来的状态和迁移,并把其 $Refine_N$ 函数值置为 False,因为它们目前不需要被精化.*IS* 和 *IT* 在 Step4 中进行更新,使其包含那些被找到但不属于 *RS* 和 *RT* 的元素、属于 *RS* 和 *RT* 但现在需要精化而之前不需要精化的元素、以及不需要精化的初始状态.在 Step5 中根据 $Refine_N$ 对 $Refine_R$ 进行刷新,*RS* 和 *RT* 在 Step6 中更新.算法在 *IS* 和 *IT* 中都不再有元素时结束,否则返回到 Step2.

算法结束后,*RS* 和 *RT* 中的状态和迁移所在的顺序自动机就组成了 H 关于 C 的切片.对于 $s\in RS$,若 $Refine_R(s)=$ False,则切片中不需要考虑 s 的精化自动机,且删除 s 的出口、入口动作;对于 $t\in RT$,若 $Refine_R(t)=$ False,则删除 t 的动作.

若某个顺序自动机中的一个状态或迁移在切片中,那么该顺序自动机中的所有状态和迁移都在切片中,而没有任何状态和迁移在切片中的顺序自动机将被删除.这使得可以顺序自动机为单元进行切片.可以看到对于切片准则 C,H 的切片由 H 中所有可能影响 C 中状态和迁移执行的顺序自动机

组成.该算法找到的切片并不一定是最小的 EHA 切片.可以看到当一个元素在被标记为 *True* 后就不会再次搜索它所依赖的元素,所以算法的开销和 EHA 的状态和迁移数目之和成线性关系;并因 EHA 的第 n 层顺序自动机中状态或迁移所依赖的元素位于第$(n-1)$到$(n+1)$层,所以切片算法的效率比较高.

算法 1　EHA 的切片算法

RS, IS, ES, NS: set of states; RT, IT, ET, NT: set of transitions;

step1:初始化.对于切片准则$\langle\{s_1,\cdots,s_k\},\{t_1,\cdots,t_n\}\rangle$,令 $RS=\{s_1,\cdots,s_k\}$, $RT=\{t_1,\cdots,t_n\}$,定义 $Refine_R(s)=True(s\in RS)$, $Refine_R(t)=True(t\in RT)$;

$$ES=\{s\mid\exists A\in F, s\in\sigma_A\wedge s\notin RS\wedge((\exists u\in RS, u\in\sigma_A)\vee(\exists t\in RT, t\in\delta_A))\}$$

$$ET=\{r\mid\exists A\in F, r\in\delta_A\wedge r\notin RT\wedge((\exists u\in RS, u\in\sigma_A)\vee(\exists t\in RT, t\in\delta_A))\}$$

令 $Refine_R(s)=False(s\in ES)$, $Refine_R(t)=False(t\in ET)$; $RS=RS\cup ES$, $RT=RT\cup ET$; $IS=RS$, $IT=RT$.

step2:根据下面公式得到 *IS* 和 *IT* 中元素依赖的状态和迁移:

$$NS=\{s\mid u\rightarrow_{d}s, u\in IS\wedge Refine_R(u)\}\cup\{s\mid u\rightarrow_{rcd}s, u\in IS\wedge\neg Refine_R(u)\}\cup\{s\mid t\rightarrow_{d}s, t\in IT\wedge Refine_R(t)\}\cup\{s\mid t\rightarrow_{tcd}s\vee t\rightarrow_{sd}s, t\in IT\wedge\neg Refine_R(t)\}$$

$$NT=\{r\mid u\rightarrow_{d'}, u\in IS\wedge Refine_R(u)\}\cup\{r\mid t\rightarrow d', t\in IT\wedge Refine_R(t)\}\cup\{r\mid t\rightarrow d'\vee t\rightarrow sd', t\in IT\wedge\neg Refine_R(t)\}$$

step3:令 $Refine_N(s)=True(s\in NS)$, $Refine_N(t)=True(t\in NT)$;

$$ES=\{s\mid\exists A\in F, s\in\sigma_A\wedge s\notin(RS\cup NS)\wedge((\exists u\in NS, u\in\sigma_A)\vee(\exists t\in NT, t\in\delta_A))\}$$

$$ET=\{r\mid\exists A\in F, r\in\delta_A\wedge r\notin(RT\cup NT)\wedge((\exists u\in NS, u\in\sigma_A)\vee(\exists t\in NT, t\in\delta_A))\}$$

令 $Refine_N(s)=False(s\in ES)$, $Refine_N(t)=False(t\in ET)$.

step4:重新构造 *IS* 和 *IT*:

$$IS=(NS-RS)\cup\{s\mid s\in NS\wedge s\in RS\wedge\neg Refine_R(s)\wedge Refine_N(s)\}\cup\{s\mid\exists A\in F, s\in ES\wedge s=s_A^0\}$$

$$IT=(NT-RT)\cup\{t\mid t\in NT\wedge t\in RT\wedge\neg Refine_R(t)\wedge Refine_N(t)\}\cup ET$$

step5:更新 $Refine_R$:

对每个 $s\in(NS\cup ES)$, $Refine_R(s)=Refine_N(s)$;

对每个 $t\in(NT\cup ET)$, $Refine_R(t)=Refine_N(t)$.

step6:重新构造 *RS* 和 *RT*:

$$RS=RS\cup NS\cup ES;\quad RT=RT\cup NT\cup ET.$$

step7:若 $IS=\emptyset$ 且 $IT=\emptyset$,则算法结束;否则返回 step2.

对图 2 中的 EHA 切片后的 EHA 如图 4 所示.可以看到顺序自动机 *IMAGE* 和 *SOUND* 被删除了,这是因为 $Refine_R(ON)$ = False.切片后 EHA 的状况数为 24,比起切片前的 48 个状况减少了一半.

在根据语义生成切片后的 EHA 所对应的 LTS 时存在一个问题:迁移的限制源状态和确定目标状态所在的顺序自动机可能被删掉.该问题可以通过以下方法解决:把迁移的 *td* 中不存在的状态删除,若之后 $td=\emptyset$,则按映射关系 ρ 回溯,找到第一个仍保留的状态添加到 *td* 中.对 *sr* 也用同样的方法处理.

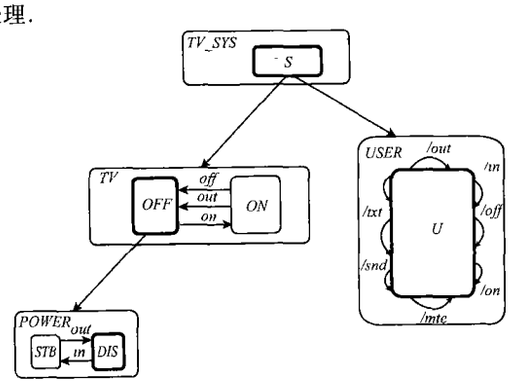

图 4　切片后的 TV EHA

在图 4 中可以看到,*USER* 中产生的事件 *txt*、*snd* 和 *mte* 都不能激发切片中的某个迁移,对切片准则没有任何影响.但它们可能会导致系统状态空间增长许多.它们被保留下来是因为状态 *S* 精化数据依赖它们.*S* 是从状态 *U* 通过精化控制依赖找到的,其实在没有任何元素以精化控制依赖之外的关系依赖 *S* 时,可以不用接着找 *S* 精化数据依赖的元素,因为 *USER* 中对切片标准有影响的元素可以通过直接对 TV 中的元素找并发数据依赖、同步依赖或迁移控制依赖得到.因此我们可以对切片算法补充下面一个优化措施.

优化措施:当在 Step2 中寻找依赖关系时,如果找到的一个新状态只被精化控制依赖,而不被任何元素以其他关系依赖时,那么若它被保留在 *IS* 中进行下一次迭代时就不再寻找它所精化数据依赖的元素.

用此方法可以使得生成事件 *txt*、*snd* 和 *mte* 的三个迁移的 $Refine_R$ 值为 False,从而可以把它们的动作去掉后变成完成迁移[11],然后与原来的完成迁移合并成一个迁移.当考虑优化措施时,TV 系统状况数目减少为 15 个,不到原来的三分之一.

4.2　从 LTL 公式中提取切片准则

我们在此考虑对线性时态逻辑(LTL)公式描述的并发系统性质进行验证.描述 UML Statecharts 性质的 LTL 公式包含三

种形式的原子命题[6]：(1) $x\ rop\ c$，其中 rop 是关系符(例如 $>$、$<$、$\neq$ 等)，当变量 x 和常量 c 满足 rop 时命题为真；(2) $@s_i$，其中 s_i 是状态名，当 UML Statecharts 的当前活跃格局包括 s_i 时命题为真；(3) $\wp e$，在当前状况的事件队列中包含事件 e 时命题为真. 对于 TV 的例子，LTL 公式 $\varphi_1: G(\wp off \rightarrow X(@OFF))$ 表示不论何时当 *off* 事件产生时，系统在下一步中都位于状态 *OFF*. LTL 公式的构造规则和语义可参见文献[16].

给定 UML Statecharts G 和一个 LTL 性质 φ，我们希望能从 φ 中提取合适的切片准则 C_φ，根据 C_φ 对 G 对应的 EHA H 进行切片能得到一个较小的 EHA 且对 φ 仍保持相同的可满足性. 下面先针对 EHA 操作语义对应的 LTS 给出两个定义[17].

定义 13 stuttering 等价路径：φ 是一个性质，AP_φ 是 φ 中出现的原子命题集，定义 $L_\varphi(s) = L(s) \cap AP_\varphi$. 两条无穷路径 $\zeta = s_0, s_1, \cdots$ 和 $\eta = r_0, r_1, \cdots$ 是 φ - stuttering 等价，记为 $\zeta \sim_\varphi \eta$，当且仅当存在两个无穷非负整数序列 $0 = i_0 < i_1 < i_2 < \cdots$ 和 $0 = j_0 < j_1 < j_2 < \cdots$ 满足对每个 $k \geqslant 0$ 有：

$$L_\varphi(s_{i_k}) = L_\varphi(s_{i_k+1}) = \cdots = L_\varphi(s_{i_{k+1}} - 1) = L_\varphi(r_{j_k}) = L_\varphi(r_{j_k+1}) = \cdots = L_\varphi(r_{j_{k+1}} - 1)$$

定义 14 stuttering 等价结构：两个结构 M 和 M' 为 φ-stuttering 等价，记为 $M \sim \varphi M'$，当且仅当

- M 和 M' 有相同的初始状态 s_0；
- 对 M 中每条从 s_0 开始的路径 ζ，在 M' 中存在一条从 s_0 开始路径 ζ' 使得 $\zeta \sim \varphi\zeta$.
- 对 M' 中每条从 s_0 开始的路径 ζ'，在 M 中存在一条从 s_0 开始路径 ζ 使得 $\zeta' \sim \varphi\zeta$.

把不包含 X 算子的 LTL 公式子集记为 LTL_{-X}. 下面引理[17]说明两个 φ - stuttering 等价的结构对一个 LTL_{-X} 公式具有相同的可满足性.

引理 1 φ 是一个 LTL_{-X} 公式，如果两个结构 M 和 M' 是 φ - stuttering 等价的，即 $M \sim \varphi M'$，则 $M, s_0 \models \varphi$ *iff* $M', s_0 \models \varphi$.

对于公式 φ，我们需要从中提取切片准则 C_φ 能够保证切片的 LTS 与原 EHA 的 LTS 之间 φ - stuttering 等价. 对于变量命题 $x\ rop\ c$，只有对 x 的定义能改变其值. 这意味对 φ 中每个 $x\ rop\ c$ 类型的命题，对 x 赋值的状态或迁移应该在切片中保留. 对于命题 $@s_i$，很明显 s_i 应包含在切片中. 对于命题 Λ_e，包含产生事件 e 的动作的状态和迁移应该保留下来. 根据以上讨论，从性质中提取的切片准则定义如下.

定义 15 φ - 准则：给定一个在 EHA H 上的 LTL_{-X} 公式 φ，集合 V 和 E 分别是所有在 φ 中出现的变量及事件，则定义 φ - 准则 $C_\varphi = \langle \{s_1, \cdots, s_k\}, \{t_1, \cdots, t_n\} \rangle$，$s_i \neq s_j (1 \leqslant i, j \leqslant k, i \neq j)$，$t_i \neq t_j (1 \leqslant i, j \leqslant n, i \neq j)$. $\{s_1, \cdots, s_k\}$ 由所有其动作对 V 中变量进行赋值或产生 E 中事件的状态和在 φ 的命题中出现的状态组成，$\{t_1, \cdots, t_n\}$ 由所有其动作对 V 中变量进行赋值或产生 E 中事件的迁移组成.

下面定理保证切片后的 EHA 与原 EHA 所对应的 LTS 之间 φ - stuttering 等价.

定理 1 对于 UML Statecharts G，H 是 G 所对应的 EHA，LTL_{-X} 公式 φ 描述了 G 的一个性质，C_φ 是 φ - 准则. 若 $SS_H(C_\varphi)$ 是根据 C_φ 得到的 H 的切片，且 M 和 M_s 是对应 H 和 $SS_H(C_\varphi)$ 的 LTS，则 $M \sim \varphi M_s$.

对于 TV 例子，如果要验证性质 $G(\wp out \rightarrow F(@DIS))$，则提取出的切片准则为 $\langle \{DIS\}, \{t\} \rangle$，其中 t 是 *USER* 中产生事件 *out* 的迁移. 根据此准则用算法 1 进行切片后的 EHA 即如图 4 所示. 切片后 EHA 对应的 LTS 的状况数从 48 减少到 24，而当考虑优化措施时，还可以进一步减少为 15 个. 用切片方法验证 UML Statecharts 的具体步骤可总结如图 5 所示.

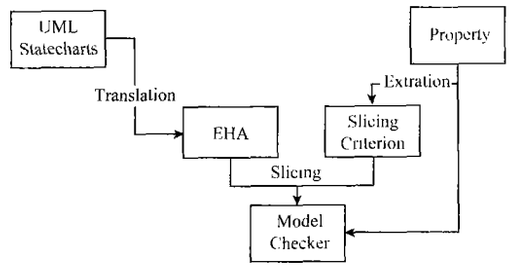

图 5 UML Statecharts 的切片模型检验过程

5 结论

本文针对模型检验用切片方法对 UML Statecharts 的状态空间进行缩减. 首先用经过扩充的 EHA 模型结构化的表示 Statecharts，然后根据 EHA 中存在的依赖关系按照状态和迁移组成的切片准则进行切片. 切片后的模型抛弃了与被验证性质无关的层次和并发状态. 由于一个给定状态或迁移依赖的元素位于 EHA 中一个较小的局部范围内，使得切片效率较高. 下一步工作包括考虑如何以状态和迁移为单元进行高效的切片以得到精细的切片，以及如何对协作图和 Statecharts 建模的多对象复杂系统用切片的思想来进行状态空间缩减等.

参考文献：

[1] UML1.3 documents and Rational Rose98 product[S]. Rational Corporation, 1999.

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

[3] E Mikk, Y Lakhnech, M Siegel, G J Holzmann. Implementing statecharts in PROMELA/SPIN[A]. In Proceedings of Workshop on Industrial-Strength Formal Specification Tecniques(WIFT'98)[C], 1998. 90 - 101.

[4] Johan Lilius, Ivan Porres Paltor. vUML: a tool for verifying UML models [A]. In Proceedings of 14th IEEE International Conference on Automated Software Engineering[C]. IEEE Computer Society, 1999. 255 - 258.

[5] S Gnesi, D Latella. Model checking UML statecharts diagrams using JACK[A]. Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering[C]. 1999. 46 - 55.

[6] W Dong, J Wang, X Qi, Z C Qi. Model checking UML statecharts[A]. Proceedings of the Eighth Asia-Pacific Software Engineering Conference (APSEC 2001)[C]. IEEE Computer Society Press, December 2001. 363 - 370.

[7] M P E Heimdahl, M W Whalen. Reduction and slicing of hierarchical

state machines[A]. Proceedings of the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering[C]. September, 1997. 450 - 467.

[8] M B Dwyer, J Hatcliff. Slicing software for model construction[A]. ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation[C]. 1999. 315 - 353.

[9] Millett L, and Teitelbaum T. Slicing promela and its applications to model checking, simulation, and protocol understanding[A]. Proceedings of the SPIN 98 Workshop in Paris[C]. France, November, 1998. 75 - 83.

[10] John Hatcliff, James Corbett, Matthew Dwyer, Stefan Sokolowski, Hongjun Zheng. A formal study of slicing for multi-threaded programs with JVM concurrency primitives[A]. Proceedings of the International Symposium on Static Analysis (SAS'99)[C]. September, 1999. 1 - 18.

[11] Rational Software, et. al. UML Semantics[S]. OMG document ad/97-08-04, 1997.

[12] Johan Lilius, Ivan Porres Paltor. The Semantics of UML State Machines [R]. TUCS Centre for Computer Science, TUCS Technical Report No 273, 1999.

[13] D Latella, I Majzik, M Massink. Towards a formal operational semantics of UML statechart diagrams[A]. In 3rd International Conference on Formal Methods for Open Object-Oriented Distributed Systems[C]. Boston, Kluwer Academic Publishers, 1999. 15 - 18.

[14] M Weiser. Program slicing[J]. IEEE Trans. Softw. Eng. 1984, SE - 4 (10): 352 - 357.

[15] J Cheng. Slicing concurrent programs-a graph-theoretical approach[A]. Proceedings of First International Workshop on Automated Algorithmic Debugging[C]. LNCS, 749, Springer-Verlag, May, 1993. 223 - 240.

[16] A Puneli. A temporal logic of concurrent programs[J]. Theoretical Computer Science, 1981, 13: 45 - 60.

[17] Edmund M Clarke, Jr Orna Grumberg, Doron A Peled. Model Checking [M]. The MIT Press, Cambridge, Massachusetts, 1999.

作者简介：

董　威　男，1976 年生于陕西咸阳，博士生，主要研究领域为软件自动验证、软件测试.

王　戟　男，1969 年出生于上海，博士，教授，主要研究领域为软件工程.

齐治昌　男，1942 年出生于辽宁锦州，教授，博士生导师，主要研究领域为软件工程.