从 UML 状态图到 PVS 规范的自动转换、验证

赖明志，尤晋元
（上海交通大学计算机科学与工程系，上海 200030）

摘 要： 将 UML(统一建模语言)图形转换成形式化规范是一种精确化 UML 语义、扩大形式化软件方法适用范围的有效途径.PVS 是一种通用高阶逻辑形式化规范语言，具有很强的描述能力以及丰富的定理证明、模型验证工具支持.本文论证了使用 PVS 来对 UML 进行形式化的优势，并且给出了 UML 的状态图到 PVS 规范的转换模型与规则.

关键词： UML 状态图；PVS；层次自动机模型；模型验证
中图分类号： TP301 **文献标识码：** A **文章编号：** 0372-2112 (2002) 12A-2122-04

Automatic Transform UML Statechart into PVS

LAI Ming-zhi, YOU Jin-yuan
(*Depart of Computer Science & Engineering Shanghai Jiao Tong University, Shanghai* 200030, *China*)

Abstract: Formalizing UML diagrams is an effective way to get a precise UML semantics and extend the usage of formal methods. PVS is a higher-order logic based general formal specification langugae with strong expressiveness and foraml analysis tools support. By compring with other formal methods, the advantages of PVS are given in formalizing UML. A hierarchical automata model and algorithms are provided for transforming UML Statechart to PVS specification.

Key words: statechart; PVS; hierarchical automata; model checking

1 引言

UML[1]是一种通用的面向对象软件建模语言，它统一了多种面向对象建模语言的优点，具有很强的描述能力以及良好的可扩展机制.然而，UML 缺少精确的语义，使得 UML 模型不能用于进一步的分析和验证.

形式化方法使用具有精确数学语义基础的形式化规范语言对系统的需求分析、设计进行描述，它具有精确定义的语义模型、自动化的验证工具的支持，可以对软件规范进行严格的分析和验证.然而，形式化方法也存在许多不足之处，如设计者需要有很好的数学基础，难以用于设计大型的软件，直接使用形式化规范对系统建模难度很大.

现有的许多研究[2~4]都试图结合面向对象和形式化方法的优点，通过将非形式化的图形转换成具有精确语义定义的形式化规范，为许多嵌入式实时、分布式，以及高可靠性系统的分析设计提供一种精确、高效、可用性强的软件方法.PVS[5]是一种通用的高阶逻辑规范语言，它集成了语法语义检查、模型验证、定理证明等强大的工具支持，在许多软硬件系统、算法、协议的分析、设计、验证中得到使用.

本文提出了一种将 UML 状态图转换成 PVS 规范的方法.软件设计人员可以首先使用 UML 图形进行建模，然后将它自动转换成为 PVS 规范，再进行模型验证或者定理证明等分析.这种方法的优点包括：避免了直接使用形式化方法对系统建模，降低了形式化方法的使用难度，提高了形式化方法的可扩展性；同时，它也为 UML 语言的精确定义提供了参考.

本文安排如下：第二部分讨论使用 PVS 作为 UML 目标形式化规范语言的优势以及相关工作对比；第三部分描述了层次自动机模型；第四部分是转化算法的设计；最后得出结论.

2 PVS 的优势以及相关工作

PVS 是由 SRI 开发的一种基于定理证明的通用形式化方法，它包括 PVS 规范语言以及分析验证工具.PVS 规范语言是基于谓词子类型的高阶逻辑语言，是一种强类型语言.

本文选择 PVS 作为形式化 UML 图形的目标规范语言，它具有以下优势：

(1)通用的形式化规范语言，具有很强的表达能力；

(2)使用强类型表示，内建的基于定理证明有效性检查，保证了语法的一致性以及规范的简洁性；

(3)支持基于 CTL 的模型验证(Model Checking)；

(4)使用自动化和交互方式相结合的定理证明机制.

在本文的工作中，主要集中于实现对 UML 状态机的转化和验证.PVS 的验证工具以定理证明为基础，同时集成了模型验证.PVS 的这种定理证明与模型验证集成机制也是本文选用 PVS 的原因.

收稿日期：2002-06-10；修回日期：2002-10-10
基金项目：国家自然科学基金(No.69973032)

一些其它的将非形式化图形转换成形式化规范的研究中,分别选用了不同的目标形式化语言.

研究[2]使用符号模型验证语言 SMV 形式化,来进行需求分析的状态图方法 RSML.RSML 也是一种来源于 Harel 状态图的方法,它使用层次机模型来实现设计的精化过程.和 UML 状态图相比,它的描述能力比较有限.SMV 是一种专用的符号模型验证工具,只能用于模型验证.

研究[3]使用一种面向对象扩展的代数语言 O-Slang 来对 OMT(UML 的前身之一)图形进行转化.O-Slang 语言在逻辑上属于基于一阶逻辑的代数规范,主要集中于描述系统的结构属性,在形式化推理方面能力较弱.

研究[4]使用 Promela 实现对 UML 状态图的形式化转换.该工作首先建立了一个自动机模型,并且使用进程演算给该模型赋予操作语义,然后将该模型转换成规范语言 Promela,该规范语言可使用 SPIN 实现模型验证.SPIN 也是一个专用的模型验证工具.

3　层次自动机模型设计

3.1　UML 状态机

UML 状态图包含了很复杂的语法定义,具有很强的描述能力.为了简化模型,本文不考虑带参数的事件,忽略 Guard,并且认为动作的执行效果只产生输出事件.UML 状态图定义了复杂的 pseudo-state 用于多个状态的同步关系以及 History 等概念,限于篇幅本文没有考虑这些概念.状态机模型中的进入和离开状态时的动作也未做考虑.图 1 是一个 UML 状态图实例.

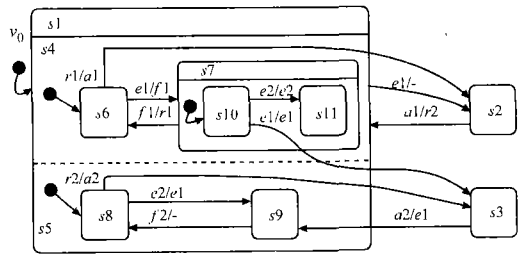


图 1　UML 状态图示例

3.2　层次自动机模型

首先提出层次自动机对简化 UML 状态图进行建模,定义其语义,以实现 UML 状态图到 PVS 规范的转换.

3.2.1　层次自动机定义

定义 1　基本自动机　表示成 3 元组$(\sigma_A, S_A^0, \delta_A)$,其中 σ_A 为有限状态集合;S_A^0 为初始状态,δ_A 为自动机内部状态迁移的集合.

状态迁移比较复杂,定义成一个 7 元组:($srcS$, sr, ev, g, a, tr, $tarS$),其中,$srcS$、$tsrS$ 代表源、目标状态,sr、tr 表示源、目标状态限制,ev 表示激发事件,g 表示所要满足的逻辑条件.

针对图 1 中的跨层状态迁移,如:$s10 \rightarrow s3$,将其定义成同层的状态迁移:$s1 \rightarrow s3$,同时引入源状态限制$\{s4, s7, s10\}$.与此类似在高层向低层状态的迁移中引入目标状态限制.

定义 2　层次自动机　由一个 4 元组表示:$\{F, E, \rho, \Gamma\}$,其中 F 为基本自动机的有限集合,E 为有限事件的集合,精化函数 $\rho: \bigcup_{A \in F} \sigma_A \rightarrow 2^F$ 构成了一个树形结构,满足条件:①存在唯一的根基本自动机,它不能作为函数的值域;②每个非根基本自动机都有且仅有一个父自动机;③函数定义不存在循环.

定义 3　在一个层次状自动机$\{F, E, \rho, \Gamma\}$当中,对于基本自动机 $A \in F$,存在下列函数:

(1) $\xi A = \{A\} \cup (\bigcup_{A' \in (\bigcup_{s \in \sigma A}(\rho_A s))} \xi A')$:自动机 A 以及由它所精化的全部自动机的集合;

(2) $\psi A = \bigcup_{A' \in \xi A} \sigma A'$:自动机 A 以及它所精化的自动机内部的所有状态的集合;

(3) $\zeta A = \bigcup_{A' \in \xi A} \delta A'$:内部所有的状态转移的集合.

定义 4　状态序列　给定:$s, s' \in \psi H$,$s \prec s'$当且仅当$s' \in \psi(\rho(s))$,其中"$\prec$"为二元关系操作符.状态序列是一个偏序关系.

3.2.2　层次自动机语义

定义 5　*Configuration* 的定义　一个层次自动机 H 的 Configuration 可以定义为集合 $C \subseteq \psi H$ 满足条件:①每个 Configuration 中必须包含一个根自动机的状态;(②Configuration 中的任意状态所精化的子自动机的状态集合中必然存在一个状态是 Configuration 的元素;③Configuration 元素的前序状态必然也是 Configuration 的元素;④任何一个 Configuration 都不存在非空真子集.

在状态图的定义中,系统的当前状态包含了当前的 Configuration 以及环境 Σ,环境 Σ 设定为一个事件的集合.事件可以触发自动机的状态迁移,状态迁移使得相应事件被消费,同时其引起的动作可能产生新的事件.

定义 6　状态迁移的语义定义　自动机 H 的语义模型可以定义成(Φ, Φ^0, T),其中,$\Phi = \bigcup_{C \subseteq \psi H}(C, \Sigma)$表示自动机中的 Configuration 与环境组合的集合,$\Phi^0 = (C^0, \Sigma^0)$表示自动机的初始状态,$T = \Phi \times \Phi \rightarrow$boolean 表示状态之间的关系.

定义 7　状态迁移限制条件 $R(t)$　$\forall t \in \delta_A, R(t) \subseteq \psi\rho(A)$给定一个基本自动机 A 中的状态 S,它的限制条件 R_s:$s \in C, R_s \subseteq C \wedge R_s \subseteq \psi s$.

定义 8　使能状态迁移集合　给定层次自动机 $H = (F, E, \rho, \Gamma)$,在该层次自动机的任意 Configuration 中所有使能的状态迁移:

$$E_H(c, \Sigma) = \{t \in \bigcup_{A \in F} \delta A \mid \{SRC(t)\} \cup (SR(t)) c \wedge Ev(t) \in \Sigma \wedge G(t)$$

定义 9　当前使能状态迁移

$$CE_H(c, \Sigma) = \{t \in E_H(c, \Sigma) \mid \neg(\exists t' \in E_{SRC(t)}(c, \Sigma))\}$$

UML 中内部的状态所触发的迁移具有更高的优先级.

根据以上定义,层次自动机 H 的状态迁移谓词:

$$T_H(s_1, s_2) = T_H((c_1, \Sigma_1), (c_2, \Sigma_2)) = \exists t \in CE_H(c_1, \Sigma_1).\ TGT(t) \cup TR(t)) \subseteq c_2 \wedge (\Sigma_1 \setminus EV(t)) \cup AC(t) = \Sigma_2$$

4　PVS 转换规则

PVS 规范语言内建的递归定义有利于表示状态机模型.

PVS 的强类型使得其规范非常简洁.

这一部分设计将 UML 状态图转换成 PVS 规范语言的具体规则,并且把一个具体的 UML 状态图转换成 Kripke 结构的 PVS 逻辑表达式.

转换规则是基于前一部分提出的层次自动机模型. PVS 转换规则分为两个部分实现:

(1)基本、层次自动机的转换:层次自动机模型是 UML 状态机的语法模型,CTL 模型是 UML 状态机的语义模型.在对 UML状态图的转换过程中,根据层次自动机模型,首先将 UML 抽象语法转换成 PVS 规范,其中 UML 状态图中的语法类型转换成 PVS 规范中的类型定义以及其必须满足的公理;具体状态图的转换变成了简单的按类型定义常量;

(2)可验证的 CTL 模型的构造:在层次自动机类型的基础上,可以构造使用 PVS 规范表示的 CTL 类型,本文中定义了函数 Config、函数常量 CTL_Trans 用于表示 CTL 模型中的元素,具体模型的转化可以通过参数赋值的方式实现.

4.1 自动机模型的转换

自动机模型的转换首先从基本自动机开始,然后在基本自动机的类型定义基础上构造层次自动机.图 2 以及图 3 分别是它们的 PVS 规范.

```
BASIC_AUTO[state:TYPE]:THEORY
BEGIN
SR,TR,Basic-Auto:    TYPE = setof[state]
Transition:TYPE = [ # src:state,sr:SR,ev:Event,g:Guard,ac:
Action,
tr:TR,tgt:state # ]
Transitions(BA):setof[Transition]]
Basic-Auto-Dependency:  AXIOM…
END BASIC_AUTO
```

图 2 基本自动机 PVS 规范

公理 Basic-Auto-Dependency 的定义说明了基本自动机的初始状态、状态迁移与其状态集合之间的类型依赖性.

```
HIERARCHICAL_AUTO[state:TYPE]:THEORY
BEGIN
Hier_Auto:TYPE from Basic_Auto
Sub_Auto(h):  setof[Basic_Auto]
Root_State(h):  setof[state]
Envir(h):  setof[Event]
Refine_func(s):setof[Basic_Auto]
Sub_state(h):  RECURSIVE setof[state] = …
All_Trans(h):RECURSIVE setof[Transition] = …
Hierarchical_Prop:AXIOM…
state_precedent(s,s1):bool = …
trans_confict(t1,t2):bool = …
END HIERARCHICAL_AUTO
```

图 3 层次自动机 PVS 规范

使用上面定义的 PVS 类型,可以依次转换 UML 状态图当中所对应的语法单元,包括基本自动机、状态、状态迁移等.图 4 是从图 1 所表示的 UML 状态图转换成的 PVS 规范:

```
EXAMPLE:THEORY
BEGIN
s1,s2,s3,s4:state
Bs0,Bs1,Bs4,Bs5,Bs7:Basic_Auto
Bs0:setof[state] = {s1,s2,s3}
Refine_func(s1):setof[Basic_Auto] =
  {Bs4,Bs5}
Bs0:Hier_Auto
Sub-Auto(Bs0):setof[Basic_Auto] =
  {Bs0,Bs4,Bs5,Bs7}
END EXAMPLE
```

图 4 示例状态图 PVS 规范

4.2 可验证的 CTL 模型的构造

在完成把 UML 层次自动机模型表示成为 PVS 规范之后,一个具体的 UML 状态图转换成了 PVS 定理中的层次自动机类型的多个常量定义,如图 4 所示.

在 PVS 规范当中,PVS 提供了时态逻辑操作符,可以直接构造用于模型验证的语句.这一部分将在前面 PVS 类型定义的基础上使用 PVS 规范构造 CTL 自动机模型:〈S,S^0,R〉,其中,S 代表状态类型,它必须是有限的;S^0 代表状态类型中的一个常量,表示初始状态;R 代表状态之间的关系,表示状态的迁移.状态空间使用 Configuration 的集合来定义.

```
CTL_AUTOMATA[state:TYPE]:THEORY
BEGIN
h:    VAR Hier_Auto
Config:TYPE = [Hier_Auto - > setof[state]]
Config_prop:  AXIOM…
Enabled_Trans(C):setof[Transition] = …
  Current_Trans(C):setof[Transition] = …
  CTL_Trans(C1,C2):bool = …
END CTL_AUTOMATA
```

图 5 CTL 自动机 PVS 规范

在图 5 的 PVS 规范中,函数 Config(h)代表层次自动机 h 的 CTL 状态集合,在 PVS 规范中它表示成一个类型定义,从它的公理限定中可以得出该类型所包含的变量数为有限个,从而保证了后面进行模型验证的可行性.CTL_Trans(C1,C2)代表 CTL 模型中的状态的迁移关系.在对实际 UML 状态图的验证过程中,可以根据需求使用谓词来表达系统应该满足的属性,然后使用 PVS 中定义的时态逻辑操作符定义特性(Property),并且使用 PVS 中的规则进行验证.例如,针对图 4 中所表示的 PVS 规范,可以定义下面的模型验证定理:

```
invariant:THEOREM
  P_init(Config(Bs0))IMPLIES
  AG(CTL_Trans,p_safe)(Config(Bs0)))
```

其中,$Bs0$ 代表一个从 UML 状态图转化过来的层次自动机,

Config(*Bs*0)表示 *CTL* 有限状态机类型,P_inti(Config(*Bs*0))是该状态机的初始状态,AG 是时态逻辑操作符,p_safe 是系统自定义的逻辑属性.使用下面规则可以完成模型验证:

(1)(expand "CTL_Trans");(2)(model_check).

5 结论

本文给出了一种将 UML 状态图转换成 PVS 规范的方法.文中分析了 PVS 在转换 UML 图形上的的优势,通过层次自动机模型定义实现了 PVS 规范的具体转换规则,通过这些规则,UML 状态图可以成功地转换成 PVS 规范进行形式化分析.从第四部分可以看到,这套转换规则非常简洁、有效.

限于篇幅,本文对 UML 状态图模型做了一些简化,当考虑更多的语法单元的转换时,只需在层次状态机的基础上设计一些语法解释层,使用本文中的这些概念来解释那些更复杂的语法单元.另外,对状态图的不变量的转换是对模型进行验证的非常重要的问题,在本文中尚未考虑.

参考文献:

[1] OMG, OMG UML Specification v1.4[Z]. available from: HYPERLINK. http://www.omg.org/technology/documents/modeling/spec/catalog. Sep, 2001.

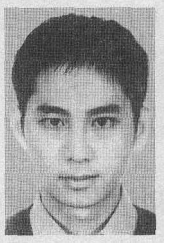
[2] William Chan, Richard J Anderson, et al. Model checking large software specification[J]. *IEEE Trans. on Software Engineering*, 1998, 24(7): 498 - 520.

[3] S A Deloach, T C Hartrum. A theory-based representation for object-oriented domain models[J]. IEEE Trans. on Software Engineering, 2000, 26(6).

[4] Diego Latella, Istvan Majzik, Mieke Massink. *Automatic verification of a* behavioral subset of UML statechart diagrams using the SPIN model-checker[J]. Formal aspects of computing, 1999, 11: 637 - 664.

[5] S Owre, J Rushby, N Shankar, F V Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS[J]. IEEE Trans. *on* Software *Engineering*, 1995, 21(2): 107 - 125.

作者简介:



赖明志　男,1974 年 12 月生于江西省泰和县,博士生,主要研究兴趣为:形式化软件方法、嵌入式实时系统的设计与开发.

尤晋元　男,1939 年 2 月生于江苏省常州市,教授,博士导师,主要研究方向为:系统软件,中间件技术,分布计算,软件工程等.