软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(7):1772−1788 [doi: 10.13328/j.cnki.jos.004841]

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

一种演化超图文法到状态转移系统的映射方法*

徐洪珍[1,2], 曾国荪[2], 王晓燕[1]

[1](东华理工大学 计算机科学与技术系,江西 南昌 330013)

[2](同济大学 计算机科学与技术系,上海 201804)

通讯作者: 徐洪珍, E-mail: xuhz@ecit.cn, http://www.ecit.edu.cn/

摘 要: 运用模型检测技术验证动态演化的正确性,是近年来软件体系结构动态演化研究领域面临的一个挑战.然而,当前的方法很少考虑软件体系结构动态演化时的相关条件.针对该问题,提出用条件状态转移系统表示软件体系结构动态演化的状态模型,将软件体系结构超图映射为状态,演化规则运用映射为条件状态转移关系,给出软件体系结构动态演化的条件超图文法到条件状态转移系统的映射方法以及相应的实现算法,实现了软件体系结构动态演化的条件状态转移系统的构建,并证明了在该映射方法下,软件体系结构动态演化条件超图文法与条件状态转移系统的互模拟等价.最后通过案例分析,运用该方法以及模型检测技术,验证了软件体系结构动态演化的相关性质,从而验证了该方法的有效性.

关键词: 软件演化;体系结构;模型检测;条件超图文法;条件状态转移

中图法分类号: TP311

中文引用格式: 徐洪珍,曾国荪,王晓燕.一种演化超图文法到状态转移系统的映射方法.软件学报,2016,27(7):1772−1788. http://www.jos.org.cn/1000-9825/4841.htm

英文引用格式: Xu HZ, Zeng GS, Wang XY. Method of mapping evolution hypergraph grammars to state transition systems. Ruan Jian Xue Bao/Journal of Software, 2016,27(7):1772−1788 (in Chinese). http:// www.jos.org.cn/1000-9825/4841.htm

Method of Mapping Evolution Hypergraph Grammars to State Transition Systems

XU Hong-Zhen[1,2], ZENG Guo-Sun[2], WANG Xiao-Yan[1]

[1](Department of Computer Science and Technology, East China University of Technology, Nanchang 330013, China)

[2](Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

Abstract: How to verify the correctness of dynamic evolution process using the model checking technique is a challenge in the dynamic software architecture evolution research field at present. In fact, the existing approaches in this direction rarely consider relevant conditions of dynamic software architecture evolution. To solve the problem, this paper proposes a state model of dynamic software architecture evolution using the conditional state transition system. This approach maps software architecture hypergraphs to states, and the application of evolution rules to the conditional state transition relation. It also provides the method for mapping conditional hypergraph grammars of dynamic software architecture evolution to conditional state transition systems and corresponding realization algorithms, as well as for implementing the construction of the conditional state transition system of dynamic software architecture

* 基金项目: 国家自然科学基金(61272107, 61262001); 国家高技术研究发展计划(863)(2009AA012201); 国家教育部网络时代的科技论文快速共享专项(20110740001); 江西省青年科学家培养对象计划(20142BCB23017); 江西省自然科学基金(20114BAB201043); 江西省科技支撑计划(20112BBE50048); 江西省发明专利产业化技术示范项目(20143BBM26115)

Foundation item: National Natural Science Foundation of China (61272107, 61262001); National High Technology Research and Development Program of China (863) (2009AA012201); Special Plan of the Rapid Sharing of Technology Papers in the Network Era of Higher Education of China (20110740001); Program of Training Young Scientists of Jiangxi Province (20142BCB23017); Natural Science Foundation of Jiangxi Province of China (20114BAB201043); Science & Technology Support Program of Jiangxi Province of China (20112BBE50048); Invention Patent Industrialization Technology Demonstration Project of Jiangxi Province of China (20143BBM26115)

收稿时间: 2014-08-18; 修改时间: 2014-11-24; 采用时间: 2015-04-09

evolution. Furthermore, the bisimulation equivalence between the conditional hypergraph grammar of software architecture dynamic evolution and the conditional state transition system under the mapping method is proved. Finally, the paper presents a case study in applying the proposed method and model checking to verify corresponding properties of dynamic software architecture evolution, demonstrating the effectiveness of the proposed method.

Key words: software evolution; architecture; model checking; conditional hypergraph grammar; conditional state transition

随着计算机技术和网络技术的不断发展,Internet 已成为当今主流的软件运行环境.在 Internet 开放环境下,软件的用户需求、计算环境等不断发生改变.当面对这些变化的需求和环境时,软件往往需要不断动态演化才能增强生命力,才能适者生存.支持动态演化的软件能在运行时改变系统的实现,包括完善系统功能、改变体系结构等,而无需重启或重编译系统[1].软件演化已成为软件生命周期中的重要组成部分[2],而软件动态演化由于具有持续可用等优点,则逐渐成为软件工程领域研究的热点.

软件体系结构 SA(software architecture)描述了软件系统的结构组成、组成元素之间的交互、连接及约束等[3].由于在软件设计中的核心地位,SA 很自然地成为研究软件动态演化的重要组成部分.SA 动态演化是指 SA 的组成元素、拓扑结构、交互关系等在系统运行时被改变或调整的过程,这种行为也通常称为 SA 运行时重配置(reconfiguration)[4].很长一段时间以来,如何建模动态演化,是 SA 演化研究领域的主要焦点[5].然而,如何验证动态演化的正确性,是近年来 SA 演化领域面临的更大挑战[6].因为即使运用形式化建模技术,也不能完全保证 SA 动态演化的正确性.为了保证演化的正确性,SA 动态演化必须满足一定的性质,且必须验证这些性质得到满足.模型检测(model checking)作为近 20 年来最为流行的形式化验证技术之一[7],已被逐渐运用于验证 SA 动态演化的正确性.然而,当前这方面的研究很少考虑 SA 动态演化时的相关条件.

针对这一问题,本文在前期工作[8−10]的基础上,提出用条件状态转移系统表示 SA 动态演化的状态模型,用抽象状态机 ASM(abstract state machine)[11]作为 SA 动态演化条件超图文法和条件状态转移系统的统一语义表示,将动态演化过程中的 SA 超图映射为状态,SA 演化规则运用映射为条件状态转移关系,给出 SA 动态演化的条件超图文法到条件状态转移系统的映射方法和实现算法,并证明了在该映射方法下,SA 动态演化的条件超图文法与条件状态转移系统的互模拟等价.最后,通过案例分析,运用本文的方法和模型检测技术验证了案例系统 SA 动态演化的相关性质,从而验证了本文方法的有效性.

1 相关工作

目前,SA 动态演化的研究工作主要集中在 SA 动态演化的建模及分析方面.例如,Oquendo[12]和梅宏[13]等人设计或使用不同的 ADL(architecture description language,体系结构描述语言),建立 SA 对象的增加、删除、替换等操作,描述、分析 SA 动态演化;Kacem 等人[14]通过扩展 UML(unified modeling language,统一建模语言),建立动态元类描述 SA 动态演化;Bruni[15]、马晓星[16]等人利用图文法,针对具体的案例系统设计了一些 SA 重配置规则,从不同方面建模案例系统的 SA 动态演化;常志明等人[17]利用 Bigraph 理论对自适应 SA 进行形式化规约,并分析了 SA 自适应动态演化的一致性和完整性;Dormoy 等人[18]提出用时态逻辑描述基于构件的软件系统的 SA 动态重配置,并使用 FPath 等工具进行了相关分析;Fiadeiro 等人[19]提出一种面向服务的体系结构 SOA(service-oriented architecture)的动态演化描述模型,该模型支持定义 ADL 来描述 SOA 的动态演化.但这些方法都未涉及 SA 动态演化的验证.

近年来,模型检测技术逐渐被应用于验证 SA 及其演化是否满足所期望的性质.Pelliccione 等人[20]提出了一个形式化框架 CHARMY,支持迭代建模和验证 SA,但未考虑 SA 的演化性;Zhang 等人[6]提出了一种分类和比较方法,对当前应用模型检测验证 SA 的技术进行了综述,指出模型检测 SA 的动态演化特性是今后的研究方向;Lanoix 等人[21]提出用 B 方法描述 SA 的动态演化,用有界模型检测工具 ProB 验证了 SA 动态演化的相关性质,但未考虑 SA 动态演化时需满足的相关条件;Eckardt 等人[22]提出用实时状态图(real-time state chart)和时间图变换(timed graph transformation)建模 SA 动态演化的行为,用模型检测器 UPPAAL 验证了 SA 动态演化的安全性等,也没有考虑 SA 动态演化时需满足的相关条件.尽管当前在运用模型检测验证 SA 的相关性质方面取得了较

多的成果,但这些成果很少考虑 SA 动态演化的特性,更没有考虑 SA 动态演化时的相关条件.如何验证 SA 动态演化的相关性质,是模型检测 SA 领域目前需要解决的关键问题之一[6].

尽管当前也存在少量基于图变换系统的模型检测方法或工具,但这些方法或工具处理的对象都是一般的图,且不能充分表示图变换的条件.例如,GROOVE(graphs for object-oriented verification)是一种运用模型检测技术验证面向对象系统的图形化工具[23],该工具运用状态转移模型描述对象系统的行为,但它只能处理简单的图,即,具有简单边的图,且只能处理一些简单的条件,即简单的负右应用条件[24].

与以上方法不同,本文所述的 SA 动态演化条件超图文法到条件状态转移系统的映射方法所建立的条件状态转移系统,则可以充分表示所需验证的 SA 的结构信息、交互信息以及 SA 动态演化的前置条件和后置条件.

2 基本概念

与前期工作[8−10]一样,本文采用超图表示 SA、条件超图文法建模 SA 动态演化过程.采用条件超图文法建模 SA 动态演化,不仅可以直观地表示 SA 的结构、交互及其拓扑,而且可以有效地表示 SA 动态演化的前断言和后断言,还能采用基于文法的形式化方法分析 SA 动态演化的相关性质[9].关于运用条件超图文法建模 SA 动态演化的详细过程可参考我们的前期工作[9].这里,为了描述清楚,只给出相关定义.

定义 2.1(超图)[9]. 设 $L=(L_V,L_E)$是一对符号集,其中,L_V为节点符号集,L_E为超边符号集.L 上的一个超图 H 定义为一个六元组 $H=(V,E,s,t,vl,el)$,其中,

(1) V 是节点集合;

(2) E 是超边集合;

(3) $s,t:E\to V^*$分别表示超边到入节点和出节点的映射,其中,*表示每条超边可以连接多个入节点和出节点;

(4) $vl:V\to L_V,el:E\to L_E$ 分别是节点和超边上的标记函数,用于表示节点和超边的相关属性.

与前期工作[8−10]一样,本文用方角矩形表示构件,圆角矩形表示连接件,用构件/连接件名以及它们之间的交互关系标记超边,用通信端口名标记节点,SA 对应的超图称为 SA 超图.例如,图 1 所示即为用超图表示的一个 SA 实例[9],其中包含 3 个构件 $client_1$,$broker_1$ 和 $server_1$、2 个连接件 $client\text{-}connector_1$ 和 $server\text{-}connector_1$、4 个通信端口 Pc_1,Pcb_1,Pbs_1 和 Ps_1;CR,CA,SR,SA 分别代表客户请求(client request)、客户响应(client answer)、服务器请求(server request)和服务器响应(server answer)等交互关系.

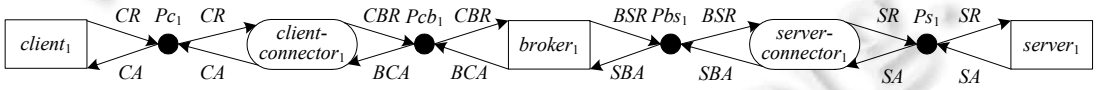


Fig.1 Hypergraph representation of the client/server system architecture

图 1 Client/Server 系统体系结构的超图表示

定义 2.2(带应用条件的 SA 演化规则)[9]. 设有一 SA,其对应的超图为 H,给定一个超图产生式规则 $p:L\to R$,即,超图 L 到超图 R 的一个部分超图态射[9],$c=(c_L,c_R)$为 p 的一个应用条件.若运用 p 可由 H 变换得到 H',且满足应用条件 c,H'为另一个 SA 超图,则称 p 为一个 SA 演化规则,其中,L 称为 p 的左手边(left-hand side),R 称为 p 的右手边(right-hand side),c_L 为 p 的一个左应用条件,c_R 为 p 的一个右应用条件.

运用带应用条件的演化规则建模单步 SA 动态演化过程可粗略描述如下:假设有一 SA,其超图表示为 H,再给定一 SA 演化规则 $p:L\to R$,$c=(c_L,c_R)$为 p 的一个应用条件,若 SA 超图 H 满足 c_L,且运用 p 动态演化后的 SA 超图满足 c_R,则寻找 H 的一个子图 L,用超图 R 对其进行替换,并保留 H 中其他部分不变,得到另一个超图 H',即为演化后的 SA 超图.

定义 2.3(SA 动态演化条件超图文法)[9]. 一个 SA 动态演化条件超图文法定义为一个四元组 $G=(\mathcal{R},P,H_0,AC)$,其中,

(1) $\mathscr{H}$为动态演化过程中的有限 SA 超图集;

(2) P 为有限的 SA 演化规则集;

(3) H_0 为初始 SA 超图;

(4) AC 为 P 上的应用条件集.

为了验证基于条件超图文法的 SA 动态演化满足相关的性质,可采用模型检测技术进行形式化验证.模型检测是一种验证有限状态系统的形式化技术[7],其基本思想[7,25,26]是:用状态转移系统(state transition system)M表示系统模型,用模态逻辑或时态逻辑公式 F 描述系统性质,这样,"系统是否具有所期望的性质"就转化为数学问题"状态转移系统 M 是否满足公式 F",用公式表示即为 $M\models F$.本文扩展了传统的状态转移系统[7,26],增加了状态转移的条件,提出用条件状态转移系统表示 SA 动态演化的状态模型,为此建立如下定义.

定义 2.4(条件状态转移系统). 一个条件状态转移系统定义为一个六元组 $M=(S,s_0,C,R,AP,L)$,其中,

(1) $S=\{s_0,s_1,\ldots,s_n\}$为有限的状态集;

(2) s_0 为初始状态;

(3) C 为有限的条件集;

(4) $R\subseteq S\times C\times S$ 为状态转移关系集,其中,R 中的单个状态转移关系记为 $s\xrightarrow{c}s'$,表示在系统状态 s 上,如果满足条件 c,则存在从状态 s 到状态 s'的一个转移关系.这里,$s,s'\in S,c\in C$;

(5) AP 为给定的原子命题集;

(6) $L:S\to 2^{AP}$ 为标记函数,用于给每个状态标记该状态所满足的原子命题集.

为了运用条件状态转移系统表示 SA 动态演化,本文对系统每个状态中的变量用变量数组 $u_0,u_1,\ldots,u_m$ 存放,则每个状态 $s_i=(x_{0i},x_{1i},\ldots,x_{mi})$表示为 $s_i=(u_0[i],u_1[i],\ldots,u_m[i])$,其中,$u_j[i]=x_{ji},i=0,\ldots,n,j=0,\ldots,m$.

状态转移关系 $s_i\xrightarrow{c}s_j$ 表示为:如果在状态 s_i 上条件 c 为真,即 $s_i\models c$,则:

$$(u_0[i],u_1[i],\ldots,u_m[i])\to(u_0[j],u_1[j],\ldots,u_m[j]).$$

为了描述方便,状态转移关系 $s_i\xrightarrow{c}s_j$ 也记为 $c\wedge(u_0[i],u_1[i],\ldots,u_m[i])\to(u_0[j],u_1[j],\ldots,u_m[j])$.原子命题则通常采用如下形式:$u_j[i]=x_{ji}$,这样,在状态 $s_i=(x_{0i},x_{1i},\ldots,x_{mi})$上,原子命题 $u_j[i]=x_{ji}$ 为真.

定义 2.5(抽象状态机)[11]. 一个抽象状态机 ASM(abstract state machine)定义为一个 3 元组:

$$\mathscr{M}=(\mathscr{A},\mathscr{I}_0,\mathscr{R}).$$

其中,

(1) $\mathscr{A}$为有限的抽象状态集;

(2) $\mathscr{I}_0$为初始抽象状态;

(3) $\mathscr{R}$为一组数量有限的转移规则.

$\mathscr{A}$的所有词汇表称为 $\mathscr{A}$的超宇宙(superuniverse),$\mathscr{A}$中的状态通常用谓词命题表示,ASM 的转移规则通过 *Update* 函数实现状态的转移,其基本形式如下:

if *Condition* then *Update*,

其中,*Update* 为一赋值函数:$f(t_1,t_2,\ldots,t_n):=t$.ASM 转移规则的语义是:如果在当前状态满足条件 *Condition*,则对当前状态的各个谓词变量 $f(t_1,t_2,\ldots,t_n)$重新赋值为 t,t 为下一状态中谓词变量的值,也即,状态机从当前状态转移到下一状态.

3 SA 动态演化条件超图文法与条件状态转移系统的映射

为了实现与特定的模型检测工具无关,以在较高抽象层实现 SA 动态演化条件超图文法与条件状态转移系统的映射,本文将源模型(SA 动态演化条件超图文法)和目标模型(条件状态转移系统)都用抽象状态机 ASM[11]进行表示.这样,ASM 为 SA 动态演化的条件超图文法和条件状态转移系统提供了统一的语义表示,并且可以证明:在该映射方法下,SA 动态演化的条件超图文法和条件状态转移系统互模拟等价.

3.1 SA动态演化条件超图文法的ASM表示

3.1.1 SA 超图的 ASM 状态表示

本文将 SA 动态演化条件超图文法中的每个 SA 超图表示为 ASM 中的一个状态,记为 $\mathcal{A}_{gg}$.$\mathcal{A}_{gg}$ 的超宇宙(superuniverse)记为$|\mathcal{A}_{gg}|$,由 SA 超图中的构件、连接件、通信端口和交互关系等的标识符组成.具体表示方法如下.

1) SA 超图中的每类构件/连接件用一元谓词 C_j 表示,该类构件/连接件的实例用一元谓词命题 $C_j(n_i)_{\mathcal{A}_{gg}}$ =true 或 $C_j(n_i)_{\mathcal{A}_{gg}}$ =false 表示.其中, $C_j(n_i)_{\mathcal{A}_{gg}}$ =true 表示在当前状态 $\mathcal{A}_{gg}$(即,该状态对应的 SA 超图)中,第 n_i 个 C_j 类型的构件/连接件实例存在; $C_j(n_i)_{\mathcal{A}_{gg}}$ =false 表示在状态 $\mathcal{A}_{gg}$ 中,第 n_i 个 C_j 类型的构件/连接件实例不存在.为了描述方便,本文将该构件/连接件实例也记为 n_i;
2) 构件与连接件间的通信端口类型用二元谓词 P 表示,每个通信端口实例用二元谓词命题 $P(n_i,n_j)_{\mathcal{A}_{gg}}$ = true 或 $P(n_i,n_j)_{\mathcal{A}_{gg}}$ =false 表示.其中, $P(n_i,n_j)_{\mathcal{A}_{gg}}$ =true 表示在状态 $\mathcal{A}_{gg}$ 中,构件/连接件实例 n_i 和 n_j 之间的通信端口存在; $P(n_i,n_j)_{\mathcal{A}_{gg}}$ =false 表示在状态 $\mathcal{A}_{gg}$ 中,构件/连接件实例 n_i 和 n_j 之间的通信端口不存在;
3) 构件与连接件间的交互关系类型用二元谓词 R_j 表示,每个交互关系实例用二元谓词公式 $R_j(n_i,n_j)_{\mathcal{A}_{gg}}$ =true 或 $R_j(n_i,n_j)_{\mathcal{A}_{gg}}$ =false 表示,其含义与情形 2)中的解释类似.为了简化生成的状态空间,本文规定构件与连接件间的交互关系只有两类:R 表示请求关系,A 表示应答关系.

因此,一个用 ASM 状态表示的 SA 超图实例实际上是所有上述定义的谓词命题的逻辑组合,记为 $[\![s]\!]_{\mathcal{A}_{gg}}$.为了方便起见,本文将 $P(n_i)_{\mathcal{A}_{gg}}$ =true, $P(n_i)_{\mathcal{A}_{gg}}$ =false 分别简写为 $[\![p(n_i)]\!]_{\mathcal{A}_{gg}}$,$[\![\neg p(n_i)]\!]_{\mathcal{A}_{gg}}$,其中,$p(n_i)$表示状态 $\mathcal{A}_{gg}$ 中的任意谓词命题.例如,给定一个 SA,其超图如图 2 所示,其中,构件 c_1 和 d_1 的类型分别为 C 和 D,连接件 cr_1 的类型为 Cr,p_1,p_2 分别表示 c_1 和 cr_1、cr_1 和 d_1 之间的通信端口,R_1,R_2,A_1,A_2 分别表示构件和连接件之间的请求、应答交互关系,则该 SA 超图对应的 ASM 状态 $\mathcal{A}_{cs}$ 定义为

$$[\![C(c_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![D(d_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![Cr(cr_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![P(c_1,cr_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![P(cr_1,d_1)]\!]_{\mathcal{A}_{cs}} \wedge$$
$$[\![R(c_1,cr_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![A(cr_1,c_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![R(cr_1,d_1)]\!]_{\mathcal{A}_{cs}} \wedge [\![A(d_1,cr_1)]\!]_{\mathcal{A}_{cs}}.$$

$\mathcal{A}_{cs}$ 的超宇宙定义为$|\mathcal{A}_{cs}|$={C,D,Cr,P,R,A,c_1,cr_1,d_1,p_1,p_2,R_1,R_2,A_1,A_2}.为了简便,下标 $\mathcal{A}_{cs}$ 在后文中将省略.

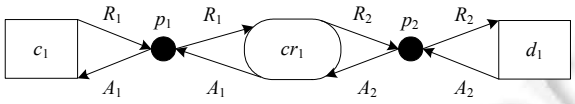


Fig.2 SA hypergraph example

图 2 SA 超图举例

3.1.2 SA 演化规则的 ASM 规则表示

ASM 通过转移规则(transition rule)描述系统的动态演化行为[11].为了用 ASM 转移规则表示 SA 演化规则,则必须在 ASM 转移规则中实现:(1) SA 演化规则左手边的匹配;(2) SA 演化规则应用条件的处理;(3) SA 元素的删除;(4) SA 元素的增加;(5) 悬挂通信端口和悬挂交互关系的处理.这里仅以带左应用条件的 SA 演化规则为例讨论其 ASM 规则表示,SA 演化规则的右应用条件在实际运用中通常可转换为其左应用条件[9],故这里不再特别加以讨论.

设 p:$L\rightarrow R$ 是一个 SA 演化规则,ac=(pro,neg)为 p 的左应用条件,其中,pro 为 p 的正左应用条件,neg 为 p 的负左应用条件.p 的左手边 L 用 ASM 表示为$[\![s_{lhs}]\!]$=$p_{lhs}(V_{lhs})$,其中,V_{lhs} 表示 L 中 SA 元素对应的谓词变量集,$p_{lhs}(V_{lhs})$表示 L 对应的谓词命题公式.p 的右手边 R 用 ASM 表示为$[\![s_{rhs}]\!]$=$p_{rhs}(V_{rhs})$,其中,V_{rhs} 表示 R 中 SA 元素对应的谓词变量集,$p_{rhs}(V_{rhs})$表示 R 对应的谓词命题公式.正左应用条件 pro 用 ASM 表示为$[\![s_{pro}]\!]$=$p_{pro}(V_{pro})$,其中,V_{pro} 表示 pro 中相关 SA 元素对应的谓词变量集,$p_{pro}(V_{pro})$表示 pro 对应的谓词命题公式.负左应用条件 neg

用 ASM 表示为$[\![s_{neg}]\!]=p_{neg}(V_{neg})$,其中,$V_{neg}$ 表示 *neg* 中相关 SA 元素对应的谓词变量集,$p_{neg}(V_{neg})$表示 *neg* 对应的谓词命题公式,则演化规则 p 的左手边匹配用 ASM 表示为 $p_{lhs}(V_{lhs})$,表示存在与左手边 L 完全一样的谓词命题公式(即,存在 SA 子超图 L).p 的应用条件用 ASM 表示为 $p_{pro}(V_{pro})\wedge\neg p_{neg}(V_{neg})$,表示既满足 p 的正左应用条件谓词命题公式 $p_{pro}(V_{pro})$,且不满足 p 的负左应用条件谓词命题公式 $p_{neg}(V_{neg})$.按照带应用条件的 SA 演化规则实施步骤,本文设计如下算法以实现 SA 演化规则的 ASM 表示.

算法 1. *ASMRepresentionofSARule*(V_{lhs},$p_{lhs}(V_{lhs})$,V_{rhs},$p_{rhs}(V_{rhs})$,$p_{pro}(V_{pro})$,$p_{neg}(V_{neg})$).

输入:

- p 的左手边变量集合 $V_{lhs}=C_{lhs}\cup P_{lhs}\cup R_{lhs}\cup A_{lhs}$;
- p 左手边的构件及连接件变量集合 C_{lhs};
- p 左手边的通信端口变量集合 P_{lhs};
- p 左手边的请求交互变量集合 R_{lhs};
- p 左手边的应答交互变量集合 A_{lhs};
- p 左手边对应的谓词命题公式 $p_{lhs}(V_{lhs})$;
- p 的右手边变量集合 $V_{rhs}=C_{rhs}\cup P_{rhs}\cup R_{rhs}\cup A_{rhs}$;
- p 右手边的构件及连接件变量集合 C_{rhs};
- p 右手边的通信端口变量集合 P_{rhs};
- p 右手边的请求交互变量集合 R_{rhs};
- p 右手边的应答交互变量集合 A_{rhs};
- p 右手边对应的谓词命题公式 $p_{rhs}(V_{rhs})$;
- p 的正左应用条件对应的谓词命题公式 $p_{pro}(V_{pro})$;
- p 的负左应用条件对应的谓词命题公式 $p_{neg}(V_{neg})$;

输出:ASM 转移规则 p.

```
1   Rule p=
2     Choose V_lh∪V_rhs with p_lhs(V_lhs)∧p_pro(V_pro)∧¬p_neg(V_neg) do
3       for all v∈V_lhs with v∉V_rhs do   //删除所有在 L 中但不在 R 中的 SA 元素
4         if type(v)=C then   //删除相应的构件/连接件,C 代表构件/连接件类型
5           C(v):=false
6           for all w with P(v,w)_lhs=true or P(w,v)_lhs=true do   //删除悬挂的通信端口
7             if P(v,w)_lhs=true then
8               P(v,w):=false
9             else
10              P(w,v):=false
11            end if
12          end for
13          for all w with R(v,w)_lhs=true or R(w,v)_lhs=true do   //删除悬挂的请求交互关系
14            if R(v,w)_lhs=true then
15              R(v,w):=false
16            else
17              R(w,v):=false
18            end if
19          end for
20          for all w with A(v,w)_lhs=true or A(w,v)_lhs=true do   //删除悬挂的应答交互关系
```

```
21              if A(v,w)_lhs=true then
22                A(v,w):=false
23              else
24                A(w,v):=false
25              end if
26            end for
27          end if
28        end for
29        for all v∈V_rhs with v∉V_lhs do   //增加所有在 R 中但不在 L 中的 SA 元素
30          if type(v)=C then   //增加相应的构件/连接件
31            C(v):=true
32          else if type(v)=P then   //增加相应的通信端口
33            for all w with P(v,w)_rhs=true or P(w,v)_rhs=true do
34              if P(v,w)_rhs=true then
35                P(v,w):=true
36              else
37                P(w,v):=true
38              end if
39            end for
40          else if type(v)=R then   //增加相应的请求交互关系
41            for all w with R(v,w)_rhs=true or R(w,v)_rhs=true do
42              if R(v,w)_rhs=true then
43                R(v,w):=true
44              else
45                R(w,v):=true
46              end if
47            end for
48          else if type(v)=A then   //增加相应的应答交互关系
49            for all w with A(v,w)_rhs=true or A(w,v)_rhs=true do
50              if A(v,w)_rhs=true then
51                A(v,w):=true
52              else
53                A(w,v):=true
54              end if
55            end for
56          end if
57        end for
58    end choose
```

其中,$p_{lhs}(V_{lhs})\wedge p_{pro}(V_{pro})\wedge\neg p_{neg}(V_{neg})$表示存在与 SA 演化规则 p 左手边 L 相匹配的子图,且满足 p 的左应用条件,Choose $V_{lhs}\cup V_{rhs}$ with $p_{lhs}(V_{lhs})\wedge p_{pro}(V_{pro})\wedge\neg p_{neg}(V_{neg})$ do 结构表示非确定性选择满足条件 $p_{lhs}(V_{lhs})\wedge p_{pro}(V_{pro})\wedge\neg p_{neg}(V_{neg})$的子图中的变量集 $V_{lhs}\cup V_{rhs}$.这也意味着,如果在目标 SA 超图中存在与演化规则左手边 L 的多个匹配,则一个 SA 演化规则可能对应多个 ASM 转移规则.其中每一次匹配运用,都对应于一个 ASM 转移规则.

这里为了方便起见,将构件/连接件谓词 C_j 统一表示为 C.

该算法的时间复杂性主要在于三重循环.设 m_l 为 L 中相关 SA 元素的总个数,即,所有构件、连接件、通信端口和交互关系的总个数,m_r 为 R 中相关 SA 元素的总个数,n_l 为 L 中通信端口的个数,n_r 为 R 中通信端口的个数,k_l 为 L 中请求交互关系的个数,k_r 为 R 中请求交互关系的个数,s_l 为 L 中应答交互关系的个数,s_r 为 R 中应答交互关系的个数,则算法 1 的时间复杂性为 $T_1=(m_l+m_r)\times\max(m_l\times\max(n_l,k_l,s_l),m_r\times\max(n_r,k_r,s_r))$.

例如,设图 3 所示为一带负左应用条件的增加服务器演化规则 $p:L\rightarrow R$[9],表示增加服务器时系统 SA 中不能已经存在两个服务器,则按照算法 1,其对应的 ASM 转移规则为:

Rule *addS*=

Choose $C(cs_1),C(sc_1),C(s_1),P(cs_1,sc_1),P(sc_1,s_1),R(cs_1,sc_1),R(sc_1,s_1),A(sc_1,cs_1),A(s_1,sc_1)$ with

$(C(cs_1)=\text{true}\wedge C(sc_1)=\text{true}\wedge P(cs_1,sc_1)=\text{true}\wedge R(cs_1,sc_1)=\text{true}\wedge A(sc_1,cs_1))=\text{true})\wedge\neg(S(s_1)=\text{true}\wedge S(s_2)=\text{true})$ do

$C(s_1)$=true

$P(sc_1,s_1)$=true

$R(sc_1,s_1)$=true

$A(s_1,sc_1)$=true

end choose

其中,在 *Choose* 中,"$(C(cs_1)=\text{true}\wedge C(sc_1)=\text{true}\wedge P(cs_1,sc_1)=\text{true}\wedge R(cs_1,sc_1)=\text{true}\wedge A(sc_1,cs_1))=\text{true})$"表示存在与 p 的左手边 L 相匹配的子图,"$\neg(S(s_1)=\text{true}\wedge S(s_2)=\text{true})$"表示不满足 p 的负左应用条件.

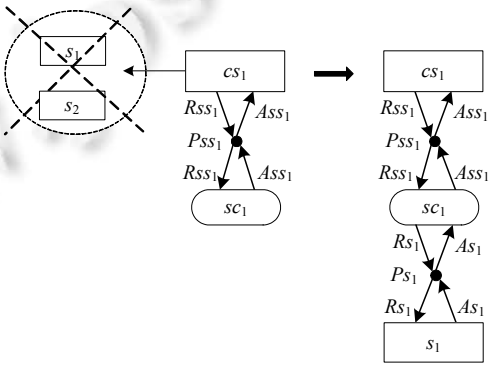


Fig.3 Adding server evolution rule with a negative left application condition

图 3 带负左应用条件的增加服务器演化规则

3.2 条件状态转移系统的ASM表示

由于条件状态转移系统是 ASM 的一种特例[11],所以用 ASM 表示条件状态转移系统非常直接.

1) 条件状态转移系统中的状态在 ASM 中用状态 $\mathcal{A}_{sts}$ 表示,其初始状态记为 $\mathcal{I}_{sts}$.在实际编程实现中,条件状态转移系统的状态变量$\{s_0,s_1,\ldots,s_n\}$在 ASM 中用状态变量数组 $S[n]$表示,其中,$S[i]$对应状态 s_i,每个状态 s_i 中的变量用变量数组 $v_0,v_1,\ldots,v_m$ 存放,则每个状态 $s_i=(x_{0i},x_{1i},\ldots,x_{mi})$可表示为 $s_i=(v_0[i],v_1[i],\ldots,v_m[i])$,其中,$v_j[i]=x_{ji}$,$i=0,\ldots,n$,$j=0,\ldots,m$.这里,$m$ 表示每个状态中最多可能的变量个数.

2) 条件状态转移系统中的状态转移关系在 ASM 中表示为:在满足相应条件后,对状态变量数组中相应的元素重新进行赋值.即,条件状态转移关系 $s_i \xrightarrow{c} s_j$ 用 ASM 表示为:

Rule t_{sts}=

if c then $S[j](v_0):=x_{0j}$, $S[j](v_1):=x_{1j},\ldots,S[j](v_{mj}):=x_{mj}$ end if

这里,状态 $s_i=(x_{0i},x_{1i},\ldots,x_{mi})$,状态 $s_j=(x_{0j},x_{1j},\ldots,x_{mj})$.

3.3 SA动态演化条件超图文法到条件状态转移系统的映射

通过前面两节,已将 SA 动态演化的条件超图文法和条件状态转移系统都用 ASM 进行了统一的语义表示,本节将给出 SA 动态演化条件超图文法映射为条件状态转移系统的方法.

3.3.1 SA 超图到状态的映射

因为 SA 超图和条件状态转移系统中的状态都用 ASM 的状态变量数组进行了表示,故,可以很容易地实现 SA 超图到条件状态转移系统中状态的映射:

1) 将 SA 超图中的每类构件或连接件映射为一个一维布尔变量数组 C_j;

2) 将 SA 超图中的通信端口、请求关系和应答关系分别映射为二维布尔变量数组 P,R 和 A.

这样,SA 超图到条件状态转移系统中状态的映射就相应于给上述变量数组赋予一定的值,而初始 SA 超图则是给上述变量数组赋予一定的初始值.

为了便于程序实现,这里规定:在 SA 动态演化过程中,每类构件/连接件、通信端口、请求关系和应答关系的数量是有限的,即,上述每类变量数组均可事先定义一个上界.

相应地,本文建立一个从 ASM 表示的 SA 超图 $\mathcal{A}_{SA}$ 到 ASM 表示的条件状态转移系统状态 $\mathcal{A}_{STS}$ 的映射 F: $\mathcal{A}_{SA}\rightarrow\mathcal{A}_{STS}$,使得:

1) 对每个布尔变量数组名 T,$F(T_{SA})=T_{STS}$;

2) 对每个布尔变量数组元素$[\![T(n_i)]\!]$,$F([\![T(n_i)=b]\!]_{SA})=[\![T(n_i)=b]\!]_{STS}$,其中,$b$ 等于 true 或 false.

例如图 2 所示的 SA 超图,其对应的 PROMELA 伪代码如下:

```
bool C[c1]=true;
bool Cr[cr1]=true;
bool D[d1]=true;
bool P[c1][cr1]=true;
bool P[cr1][d1]=true;
bool R[c1][cr1]=true;
…
```

这里以模型检测工具 SPIN[27]为例描述条件状态转移系统的状态编码,并且为了便于理解,对实际的 PROMELA 编码进行了一些改动.例如用一维数组元素 $C[c_1]$表示构件 c_1、二维数组元素 $P[c_1][cr_1]$表示通信端口 $P(c_1,cr_1)$、C 代表构件类型、P 代表通信端口类型,等等.

3.3.2 SA 演化规则运用到条件状态转移关系的映射

SA 演化规则运用到条件状态转移关系的映射主要需处理以下几个方面的问题:

1) 演化规则左手边图的匹配;

2) 演化规则应用条件的处理;

3) 演化规则运用的处理(即,条件状态转移关系的处理).

这些处理与算法 1 类似,然而与 ASM 中转移规则不同的是,条件状态转移系统中的状态转移关系必须是确定性的,即,满足一定的条件后进行相应的状态变量赋值.因此,SA 演化规则运用到条件状态转移关系的映射必须消除算法 1 中的 choose 结构,亦即将其转换为确定性结构.为此,本文设计以下算法实现 SA 演化规则运用到条件状态转移关系的映射.

算法 2. *StateTransitionsofSARules*(V_{lhs},$p_{lhs}(V_{lhs})$,V_{rhs},$p_{rhs}(V_{rhs})$,p_{pro},p_{neg}).

输入:

- p 左手边变量集合 $V_{lhs}=C_{lhs}\cup P_{lhs}\cup R_{lhs}\cup A_{lhs}$;
- p 左手边的构件及连接件变量集合 C_{lhs};
- p 左手边的通信端口变量集合 P_{lhs};
- p 左手边的请求交互变量集合 R_{lhs};

- p 左手边的应答交互变量集合 A_{lhs};
- p 左手边对应的谓词命题公式 $p_{hs}(V_{lhs})$;
- p 右手边变量集合 $V_{rhs}=C_{rhs}\cup P_{rhs}\cup R_{rhs}\cup A_{rhs}$;
- p 右手边的构件及连接件变量集合 C_{rhs};
- p 右手边的通信端口变量集合 P_{rhs};
- p 右手边的请求交互变量集合 R_{rhs};
- p 右手边的应答交互变量集合 A_{rhs};
- p 右手边对应的谓词命题公式 $p_{rhs}(V_{rhs})$;
- p 的正左应用条件对应的谓词命题公式 p_{pro};
- p 的负左应用条件对应的谓词命题公式 p_{neg};

输出:相应的条件状态转移关系集合 T.

1 Let $V_{lhs}:=\{v_0,v_1,\ldots,v_n\}$, $V_{rhs}:=\{w_0,w_1,\ldots,w_m\}$, *assignment*:=true, $T:=\varnothing$
2 for all $(x_0,x_1,\ldots,x_n)\in dom(v_0)\times dom(v_1)\times\ldots\times dom(v_n)$ do //选择左手边可能的匹配
3 $v_0:=x_0,v_1:=x_1,\ldots,v_n:=x_n$ //SA 元素变量赋值
4 $g_{lhs}:=p_{lhs}(x_0,x_1,\ldots,x_n)$ //相应的谓词命题公式
5 if $g_{lhs}\equiv p_{lhs}$ then //与左手边图完全匹配
6 for all updates $p_k(y_k):=b_k$ in p do //p 是相应的 SA 演化规则
7 $assignment:=assignment\wedge(p_k(y_k):=b_k)$
8 end for
9 $t:=(g_{lhs}\wedge p_{pro}\wedge\neg p_{neg})\rightarrow assignment$ //生成条件状态转移关系
10 $T:=T\cup\{t\}$
11 end if
12 end for
13 return T

其中,演化规则 p 左手边中所有 SA 元素对应的变量集合设为 $V_{lhs}=\{v_0,v_1,\ldots,v_n\}$,其对应的 SA 元素值为$(x_0,x_1,\ldots,x_n)$,相应的谓词命题公式为 $p_{lhs}(x_0,x_1,\ldots,x_n)$,$T$ 为条件状态转移关系集,初始设为空集$\varnothing$,$dom(v_i)$表示变量 v_i 对应的域(即,该类 SA 元素所有可能的取值),$g_{lhs}\equiv p_{lhs}$ 表示与演化规则 p 的左手边完全匹配,赋值语句中第 6 行~第 8 行对应于算法 1 中相应的赋值语句中的第 3 行~第 57 行.这里为了简便起见,仅用 $p_k(y_k):=b_k$ 表示所有相关的赋值,b_k 表示 true 或 false.

该算法的时间复杂性主要在于外层循环“for all $(x_0,x_1,\ldots,x_n)\in dom(v_0)\times dom(v_1)\times\ldots\times dom(v_n)$ do”和内层循环“for all updates $p_k(y_k):=b_k$ in p do”.设 f_i 为 SA 元素 v_i 所有可能的取值个数,$i=1,2,\ldots,n$,T_1 为算法 1 的时间复杂性,则算法 2 的时间复杂性为 $T_2=f_1\times f_2\times\ldots\times f_n\times T_1$.

3.4 SA动态演化的条件状态转移系统构建

根据前面的映射方法,可由 SA 动态演化的条件超图文法得到对应的条件状态转移系统,本文称其为 SA 动态演化的条件状态转移系统.

定义 3.1(SA 动态演化条件状态转移系统). 给定一个 SA 动态演化条件超图文法,该 SA 动态演化的条件状态转移系统定义为一个六元组 $M=(S,s_0,C,R,AP,L)$,其中,(1) $S=\{s_0,s_1,\ldots,s_n\}$为有限的状态集,对应于 SA 动态演化条件超图文法中的 SA 超图集;(2) s_0 为初始状态,对应于初始 SA 超图;(3) C 为条件状态转移关系的条件集,对应于 SA 动态演化条件超图文法中的应用条件集;(4) $R\subseteq S\times C\times S$ 为条件状态转移关系集,其中每个条件状态转移关系对应于 SA 动态演化条件超图文法中的一次演化规则运用;(5) AP 为给定的原子命题集;(6) $L:S\rightarrow 2^{AP}$ 为标记函数,用于给每个状态标记该状态所满足的原子命题集.

SA 动态演化条件超图文法与条件状态转移系统的对应关系可描述如下:给定一个 SA 动态演化条件超图

文法 $G=\{\{H_0,H_1,\ldots,H_n\},P,H_0,AC\}$,则令 $s_i=H_i,i=1,2,\ldots,n,s_0=H_0,C=AC,R=\{s_i \xrightarrow{c} s_j \mid \exists H_i=s_i \wedge \exists H_j=s_j \wedge \exists p\in P \wedge \exists c\in AC \wedge H_i \xrightarrow[p,c]{} H_j\}$,其中,$c$ 为 SA 演化规则 p 的应用条件; $H_i \xrightarrow[p,c]{} H_j$ 表示满足应用条件 c 时,运用 p 进行的 SA 动态演化.再给定原子命题集 AP 和标记函数 L,则得到对应的 SA 动态演化条件状态转移系统 $M=(S,s_0,C,R,AP,L)$.这里,AP 通常如下给定:根据实际情况,对每个状态中的变量数组元素赋值为 true 或 false.

4 映射下的互模拟等价

本节将证明:在 ASM 表示的抽象层上,SA 动态演化的条件超图文法(其 ASM 表示为 ASM_{SA})与按照第 3 节方法生成的条件状态转移系统(其 ASM 表示为 ASM_{STS})互模拟等价(bisimulation equivalence)[7,26].为了证明 ASM_{SA} 和 ASM_{STS} 互模拟等价,则必须证明:

1) ASM_{SA} 的初始状态 $\mathcal{I}_{SA}$ 和 ASM_{STS} 的初始状态 $\mathcal{I}_{STS}$ 互模拟等价,即,$\mathcal{I}_{SA}$ 和 $\mathcal{I}_{STS}$ 之间存在一一对应关系;
2) ASM_{SA} 的演化规则运用与 ASM_{STS} 的条件状态转移关系互模拟等价,这里又可分为两部分:
   (i) 对 ASM_{SA} 中的每次演化规则运用,都存在 ASM_{STS} 中的一个条件状态转移关系与之对应;
   (ii) 对 ASM_{STS} 中的每个条件状态转移关系,都对应于 ASM_{SA} 中的一次演化规则运用.

本文采用文献[11]的精化模式(refinement scheme),将 ASM_{SA} 看成是一个抽象状态机,ASM_{STS} 看成是 ASM_{SA} 的精化状态机.为此定义一个精化函数 F,使得按照第 3.3 节的映射方法,ASM_{SA} 中每个状态 $\mathcal{A}_{SA}$ 都对应于 ASM_{STS} 中的一个状态 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$,且对 ASM_{SA} 中每个演化规则 p 的运用,对应于 ASM_{STS} 中的一个状态转移关系 $T=F(p)$,则上述互模拟等价中的第 2)点转变为要证明:

(i) 通过映射 F,每个抽象的计算都被一个具体的计算实现.即:ASM_{SA} 中的每次演化规则运用,通过映射 F 的作用,都对应于 ASM_{STS} 中的一个条件状态转移关系;
(ii) 通过映射 F,每个具体的计算都实现了一个抽象的计算.即:ASM_{STS} 中的每个条件状态转移关系,都是通过映射 F 的作用,对应于 ASM_{SA} 中的一次演化规则运用.

下面对第 1)点、第 2)点分别进行证明.

命题 4.1(初始状态的互模拟等价). ASM_{SA} 的初始状态 $\mathcal{I}_{SA}$ 和 ASM_{STS} 的初始状态 $\mathcal{I}_{STS}$ 是互模拟等价的,即:

$$\mathcal{I}_{STS}=F(\mathcal{I}_{SA}).$$

证明:由第 3.3.1 节显然可知,ASM_{SA} 中的状态与 ASM_{STS} 中的状态具有一一对应的关系.从而,ASM_{SA} 的初始状态 $\mathcal{I}_{SA}$ 和 ASM_{STS} 的初始状态 $\mathcal{I}_{STS}$ 具有一一对应关系,即,$\mathcal{I}_{STS}=F(\mathcal{I}_{SA})$,故本命题成立. □

命题 4.2(SA 演化规则运用与状态转移关系的互模拟等价).

(1) 对 $\forall\mathcal{A}_{SA}\wedge\forall\mathcal{A}_{STS}\wedge\forall\mathcal{B}_{SA}$,满足 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$ 且 $\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$,则存在 $\mathcal{B}_{STS}$,使得 $\mathcal{B}_{STS}=next_{F(p)}(\mathcal{A}_{STS})$,且 $\mathcal{B}_{STS}=F(\mathcal{B}_{SA})$;

(2) 对 $\forall\mathcal{A}_{SA}\wedge\forall\mathcal{A}_{STS}\wedge\forall\mathcal{B}_{STS}$,满足 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$ 且 $\mathcal{B}_{STS}=next_{F(p)}(\mathcal{A}_{STS})$,则存在 $\mathcal{B}_{SA}$,使得 $\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$ 且 $\mathcal{B}_{STS}=F(\mathcal{B}_{SA})$;

其中,$\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$ 表示 ASM_{SA} 中对 $\mathcal{A}_{SA}$ 运用演化规则 p 演化后得到的 SA 超图状态,$\mathcal{B}_{STS}=next_{F(p)}(\mathcal{A}_{STS})$ 表示 ASM_{STS} 中对 $\mathcal{A}_{STS}$ 进行运用 p 对应的状态转移得到的状态,其他表示类似.

证明:

(1) 设 p 是 ASM_{SA} 中的一个 SA 演化规则,T 是由 p 按照算法 2 生成的 ASM_{STS} 中的一序列条件状态转移关系,$v_0=x_0,v_1=x_1,\ldots,v_n=x_n$ 是满足 p 的应用条件,且与 p 左手边匹配的所有变量的一次赋值.那么,显然有 $(x_0,x_1,\ldots,x_n)\in dom(v_0)\times dom(v_1)\times\ldots\times dom(v_n)$,故由算法 2 可生成 $(x_0,x_1,\ldots,x_n)$ 对应的一个条件状态转移关系 $t\in T$,即:对 $\forall\mathcal{A}_{SA}\wedge\forall\mathcal{A}_{STS}\wedge\forall\mathcal{B}_{SA}$,如果满足 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$ 且 $\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$,则存在 $\mathcal{B}_{STS}$,使得 $\mathcal{B}_{STS}=next_{F(p)}(\mathcal{A}_{STS})$.

又由于算法 2 中的赋值语句(第 6 行~第 8 行)完全对应于算法 1 中相应的赋值语句(第 3 行~第 57 行),即:进行状态转移 t 后,状态 $\mathcal{B}_{STS}$ 的相应赋值完全对应于运用演化规则 p 后状态 $\mathcal{B}_{SA}$ 的相应赋值,故有 $\mathcal{B}_{STS}=F(\mathcal{B}_{SA})$.

(2) 设 $t\in T$ 是由算法 2 生成的 ASM_{STS} 中一个使能的条件状态转移关系,使得对 $\forall\mathcal{A}_{SA}\wedge\forall\mathcal{A}_{STS}\wedge\forall\mathcal{B}_{STS}$,满足 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$ 且 $\mathcal{B}_{STS}=next_t\mathcal{A}_{STS}$).因为 t 使能且 $t=(g_{lhs}\wedge p_{pro}\wedge\neg p_{neg})\rightarrow assignment$,则显然:对 ASM_{STS} 中所有出现在

g_{lhs} 中的变量 $x_0,x_1,\ldots,x_n$,它们对应的谓词命题为真,且满足($p_{pro}\wedge\neg p_{neg}$).又因为 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$,且 ASM_{SA} 中的状态与 ASM_{STS} 中的状态具有一一对应关系,所以对 ASM_{SA} 中的相应变量值 $x_0,x_1,\ldots,x_n$,它们对应的谓词命题也为真.

因为 t 是由算法 2 生成的条件状态转移关系,故存在 ASM_{SA} 中的一个 SA 演化规则 p,由 p 可生成 t,即,$t=F(p)$.又因为 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$,所以对 ASM_{SA} 中的相应变量 $x_1,x_2,\ldots,x_n$,其 g_{lhs} 也为真,且满足($p_{pro}\wedge\neg p_{neg}$).因此,SA 演化规则 p 可运用于 ASM_{SA} 中的状态 $\mathcal{A}_{SA}$,即:存在 $\mathcal{B}_{SA}$,使得 $\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$.

又由于算法 2 中的赋值语句(第 6 行~第 8 行)完全对应于算法 1 中相应的赋值语句(第 3 行~第 57 行),故运用演化规则 p 后,状态 $\mathcal{B}_{SA}$ 的相应赋值完全对应于运用条件状态转移关系 t 后状态 $\mathcal{B}_{STS}$ 的相应赋值,故有:

$$\mathcal{B}_{STS}=F(\mathcal{B}_{SA}).$$

故命题成立. □

5 案例分析

为了便于描述,下面使用一个基于 C/S 风格的应用系统[9]作为案例,讨论如何将 SA 动态演化的条件超图文法映射为条件状态转移系统,并验证 SA 动态演化的相关性质.如图 4 所示,该系统包含 3 类构件:客户 c_i、控制服务器 cs_1、服务器 s_i;两类连接件:客户连接件 cc_i、服务器连接件 sc_i;客户 c_i 和客户连接件 cc_i 之间的通信端口为 Pc_i,客户连接件 cc_i 和控制服务器 cs_1 之间的通信端口为 Pcs_i;Rc_i 表示客户 c_i 向客户连接件 cc_i 发出的客户请求,Ac_i 表示客户连接件 cc_i 向客户 c_i 发回的客户应答,等等.另外,本文规定系统 SA 动态演化还必须满足如下条件:(1) 系统演化过程中只能包含一个控制服务器;(2) 系统演化过程中最多只能有 m 个服务器,不妨设 m=5;(3) 系统最多只能接受 n 个客户的连接请求,不妨设 n=100;(4) 为了保证通信时不出现环路,每个构件和连接件只能通过一个通信端口相连.

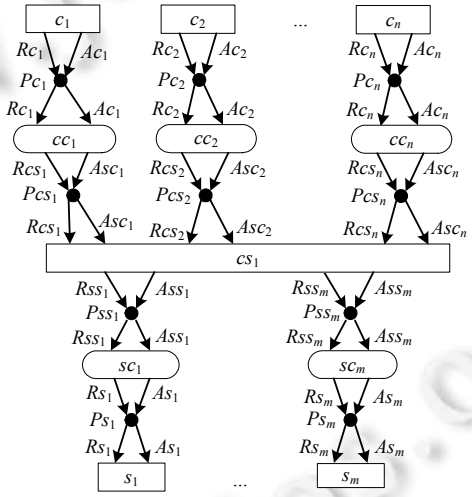


Fig.4 Client/Server system example

图 4 客户/服务器系统例子

5.1 案例系统SA动态演化的条件超图文法

由于本文的目的不是讨论如何建立 SA 演化规则及其应用条件,所以这里仅以增加服务器和增加客户为例定义以下带应用条件的演化规则:带应用条件的增加服务器演化规则(如图 3 所示)和增加客户演化规则(如图 5 所示),其他的 SA 演化规则及其应用条件定义可参考我们的前期工作[9].

假设系统初始 SA 超图为 H_0(空集),则可以定义该系统 SA 动态演化的一个条件超图文法.

$$G=\{\{H_0,H_1,\ldots,H_n\},P,H_0,AC\},$$

其中,$H_1,\ldots,H_n$ 为系统动态演化过程中的有限 SA 超图系列,P 为演化规则集,AC 为演化规则的应用条件集.

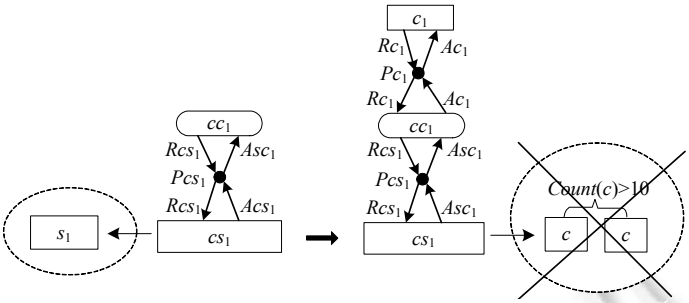


Fig.5 Adding client evolution rule with application conditions

图 5 带应用条件的增加客户演化规则

5.2 案例系统SA动态演化条件超图文法到条件状态转移系统的映射

5.2.1 案例系统 SA 超图到状态的映射

根据第 3 节的方法,将上述 SA 动态演化条件超图文法 G 中的 SA 超图映射为状态.为表示这些状态,在 PROMELA 中定义以下布尔变量或布尔变量数组:

```
bool cs1;          //标识每个状态中的控制服务器 cs1
bool C[100];       //标识每个状态中的各个客户构件
bool Cc[100];
bool S[5];
bool Sc[5];
bool Pc[100];
bool Pcs[100];
bool Pss[5];
…
```

因为在案例系统中,客户构件 c_i 只可能和客户连接件 cc_i 之间存在通信端口,与其他连接件和构件之间不可能存在通信端口,所以这里只用一维数组 Pc[100]表示客户构件和客户连接件之间的通信端口,其他变量数组的定义类似.那么,在初始状态 s_0 中,所有变量或变量数组元素的值均为 false,在其他状态上,所有变量或变量数组元素赋予相应的值.

5.2.2 案例系统 SA 演化规则运用到条件状态转移关系的映射

根据第 3 节所提方法,可将每次 SA 演化规则运用映射为一个条件状态转移关系.例如,设系统当前 SA 超图 H_1 如图 6 所示,如果将图 3 所示的增加服务器演化规则 p 运用于 H_1 进行动态演化,则存在两种可能的匹配,即:既可和连接件 sc_1 匹配,也可和连接件 sc_2 匹配,则运用演化规则 p 后,H_1 可能演化成如图 7 所示的 SA 超图 H_2,也可能演化成如图 8 所示的 SA 超图 H_3.于是,p 的两次匹配运用可映射为两个条件状态转移关系,如图 9 所示,其中,状态 s_1 对应于 SA 超图 H_1,s_2 对应于 H_2,s_3 对应于 H_3.

根据算法 2,演化规则 p 可能的运用映射成条件状态转移关系对应的 PROMELA 伪代码如下:

```
if
::(cs1==true&&Sc[1]==true&&Pss[1]==true&&Rss[1]==true&&Ass[1]==true&&S[1]==false)→S[1]=true;
Ps[1]=true; Rs[1]=true; As[1]=true;
::(cs1==true&&Sc[2]==true&&Pss[2]==true&&Rss[2]==true&&Ass[2]==true&&S[2]==false)→S[2]=true;
Ps[2]=true; Rs[2]=true; As[2]=true;
fi
```

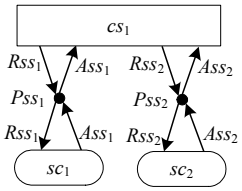


Fig.6 Current SA hypergraph H_1 of the system

图 6 系统当前 SA 超图 H_1

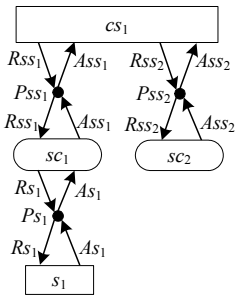


Fig.7 SA hypergraph H_2 after dynamic evolution

图 7 动态演化后的 SA 超图 H_2

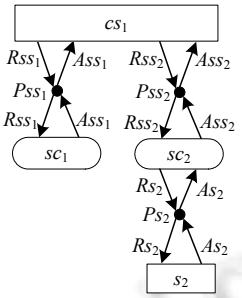


Fig.8 SA hypergraph H_3 after dynamic evolution

图 8 动态演化后的 SA 超图 H_3

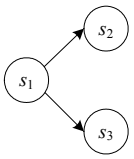


Fig.9 State transition relation of the mapping

图 9 映射成的状态转移关系

5.2.3 案例系统 SA 动态演化的条件状态转移系统

根据前面两小节,可建立该案例系统 SA 动态演化的条件状态转移系统 $M=(S,s_0,R,C,AP,L)$,其中,$S=\{s_0,s_1,\ldots,s_n\}$,$s_0=H_0,s_1=H_1,\ldots,s_n=H_n$,$C=AC$,$R\subseteq S\times C\times S$ 为如图 9 所示的状态转移关系集,AP 为给定原子命题集,$L:S\to 2^{AP}$ 为标记函数.这里,AP 通常采用以下形式:根据实际情况对每个状态中的变量或变量数组元素赋值为 true 或 false.

5.2.4 案例系统 SA 动态演化的活性验证

为了验证本文方法的有效性,这里以案例系统 SA 动态演化的活性为例,运用模型检测进行验证.SA 动态演化性质 p 是一个活性[10],是指存在一个 p 对应的逻辑命题公式 ϕ,使得:

$$\exists r\in P^*,\exists n\in\mathbb{N},H_0\xrightarrow[r]{}H_n\wedge H_n\models\phi,$$

其中,P 为 SA 演化规则集,r 为一个 SA 演化规则序列,H_0 为系统初始 SA 超图,$\mathbb{N}$为自然数集.上式也可用一个计算树逻辑 CTL(computation tree logic)公式表示:$EFH_n,H_n\models\phi$.即:存在一个将来演化到的 SA 实例 H_n,使得 ϕ 满足,其中,E,F 均为时态逻辑算子,E 表示存在一条路径,F 表示将来.例如在上述应用场景中,如果一个客户接入本系统,要求使用系统服务,则最终必须有一个服务器提供服务.

为了保证系统的可用性,必须保证该性质被满足.该性质用逻辑公式可表示为:假设系统当前 SA 超图为 H_0,对任意客户 $c_i\in H_0$,有:

$$\exists r\in P^*,\exists m\in\mathbb{N},H_0\xrightarrow[r]{}H_m\wedge H_m\models(\exists(cc_i\wedge Pc_i\wedge Pcs_i),(Pc_i(c_i,cc_i)\wedge Pcs_i(cc_i,cs_1))\in H_m)\wedge$$
$$(\exists(s_j\wedge sc_j\wedge Ps_j\wedge Pss_j),(Ps_j(sc_j,s_j)\wedge Pss_j(sc_j,cs_j))\in H_m),$$

其中,cc_i 表示一个客户连接件实体,s_j 表示一个服务器实体,Pc_i,Ps_j 为通信端口,P 为上述场景所定义的 SA 演化规则集,r 为一个 SA 演化规则序列,H_m 为系统动态演化过程中的某个 SA 超图实例.

该公式改写为 CTL 公式如下:

$$EF((\exists(cc_i \wedge Pc_i \wedge Pcs_i),(Pc_i(c_i,cc_i) \wedge Pcs_i(cc_i,cs_1)) \in H_m) \wedge$$
$$(\exists(s_j \wedge sc_j \wedge Ps_j \wedge Pss_j),(Ps_j(sc_j,s_j) \wedge Pss_j(sc_j,cs_j)) \in H_m)) = EFf.$$

为了运用模型检测技术验证该性质是SA动态演化活性,本文借助模型检测工具SPIN进行验证.因为SPIN主要检测线性时态逻辑LTL(linear temporal logic)公式,为了运用SPIN实现上述公式的验证,本文对上述公式进行了微调,将任意客户 c_i 加入到案例系统时的SA超图假定为系统的初始SA超图 H_0,则上述公式可用LTL公式表示如下:

$$F((\exists(cc_i \wedge Pc_i \wedge Pcs_i),(Pc_i(c_i,cc_i) \wedge Pcs_i(cc_i,cs_1)) \in H_m) \wedge$$
$$(\exists(s_j \wedge sc_j \wedge Ps_j \wedge Pss_j),(Ps_j(sc_j,s_j) \wedge Pss_j(sc_j,cs_j)) \in H_m)) = Ff.$$

即:在系统当前的SA超图 H_0 上,最终必须有一个服务器 s_j 为 c_i 提供服务.

本文用SPIN对上述公式进行了验证,表1为部分实验结果,其中,实验机器为Dell OptiPlex320,双CPU,主频1.60GHz,内存2GB.例如,当在SA动态演化过程中生成的状态数为40、状态转移数为81时,我们用SPIN验证上述活性公式所需的时间为0.315s;当生成的状态数为100、状态转移数为120时,用SPIN验证所需的时间为1.033s.两种情况下,上述活性的验证结果均为真.实验结果表明,在使用本文方法生成的条件状态系统上,上述活性性质始终成立.

Table 1 Some experiment results

表1 部分实验结果

状态数	转移数	验证时间(s)	结果
40	81	0.315	True
100	120	1.033	True

6 结束语

SA作为复杂软件系统设计中的核心部分,从较高的抽象层描述了系统的多个方面,从系统全局的组织结构到构件、通信、拓扑等.SA动态演化已经成为软件演化研究的关键问题,然而,如何验证动态演化的正确性是目前SA动态演化研究面临的更大挑战.因为随着外部环境和软件系统本身复杂度的不断增加,概念上的设计错误可能出现在任何高层模型上,即使用形式化的建模技术,也不能完全保证SA动态演化的正确性.为了确保软件系统及其动态演化的可靠性,形式化验证将成为SA动态演化研究中必不可少的组成部分.模型检测作为目前主流的形式化验证技术,已被逐渐应用于验证SA规范是否满足所期望的性质,但目前该方面的研究尚未考虑SA动态演化时的相关条件.模型检测SA动态演化的一个重要组成部分是建立SA动态演化的状态转移系统.本文着重研究如何建立SA动态演化的条件状态转移系统,首先讨论了用ASM对SA动态演化条件超图文法和条件状态转移系统进行统一的语义表示,给出了SA动态演化条件超图文法到条件状态转移系统的映射方法;接着证明了在该映射方法下,SA动态演化条件超图文法和条件状态转移系统的互模拟等价;最后,通过案例分析,讨论了如何运用本文提出的映射方法构建SA动态演化的条件状态转移系统,并运用模型检测技术验证了SA动态演化的活性.

本文提出的SA动态演化条件超图文法到条件状态转移系统的映射方法不仅可以实现与特定的模型检测工具无关,而且还能从理论上证明映射后的SA动态演化条件状态转移系统与映射前的SA动态演化条件超图文法是互模拟等价的.进一步的工作是运用所建立的条件状态转移系统,结合模型检测的其他技术,如符号模型检测等,验证带条件的SA动态演化的其他性质,并对SA动态演化的状态空间进行约简.

References:

[1] Godfrey MW, German DM. The past, present, and future of software evolution. In: Proc. of the 24th IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Press, 2008. 129−138. [doi: 10.1109/FOSM.2008.4659256]

[2] Boehm B. The changing nature of software evolution. IEEE Software, 2010,27(4):26−28. [doi: 10.1109/MS.2010.103]

[3] Shaw M, Clements P. The golden age of software architecture. IEEE Software, 2006,23(2):31−39. [doi: 10.1109/MS.2006.58]

[4] Gomaa H, Hussein M. Software reconfiguration patterns for dynamic evolution of software architectures. In: Proc. of the 4th Working IEEE/IFIP Conf. on Software Architecture (WICSA 2004). Norway: IEEE CS Press, 2004. 79−88. [doi: 10.1109/WICSA.2004.1310692]

[5] Mens T, Magee J, Rumpe B. Evolving software architecture descriptions of critical systems. IEEE Computer, 2010,43(5):42−48. [doi: 10.1109/MC.2010.136]

[6] Zhang P, Muccini H, Li B. A classification and comparison of model checking software architecture techniques. Journal of Systems and Software, 2010,83(5):723−744. [doi: 10.1016/j.jss.2009.11.709]

[7] Clarke E, Grumberg O, Peled D. Model Checking. Cambridge: MIT Press, 2000.

[8] Xu HZ, Zeng GS. Dynamic evolution of software architectures based on hypergraph grammars. Journal of Tongji University (Natural Science), 2011,39(5):745−750 (in Chinese with English abstract). [doi: 10.3969/j.issn.0253-374x.2011.05.021]

[9] Xu HZ, Zeng GS, Chen B. Conditional hypergraph grammars and its analysis of dynamic evolution of software architectures. Ruan Jian Xue Bao/Journal of Software, 2011,22(6):1210−1223 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4017.htm [doi: 10.3724/SP.J.1001.2011.04017]

[10] Xu HZ, Zeng GS. Modeling and verifying composite dynamic evolution of software architectures using hypergraph grammars. Int'l Journal of Software Engineering and Knowledge Engineering, 2013,23(6):775−799. [doi: 10.1142/S0218194013500204]

[11] Börger E. The ASM refinement method. Formal Aspects of Computing, 2003,15(2-3):237−257. [doi: 10.1007/s00165-003-0012-7]

[12] Oquendo F. π-ADL: An architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. ACM Sigsoft Software Engineering Notes, 2004,29(3):1−14. [doi: 10.1145/986710.986728]

[13] Mei H, Chen F, Wang QX, Feng YD. ABC/ADL: An ADL supporting component composition. LNCS, 2002,2495:38−47. [doi: 10.1007/3-540-36103-0_6]

[14] Kacem MH, Kacem AH, Jmaiel M, Drira K. Describing dynamic software architectures using an extended UML model. In: Proc. of the Symp. on Applied Computing. New York: ACM Press, 2006. 1245−1249. [doi: 10.1145/1141277.1141569]

[15] Bruni R, Bucchiarone A, Gnesi S, Melgratti H. Modelling dynamic software architectures using typed graph grammars. Electronic Notes in Theoretical Computer Science, 2008,213(1):39−53. [doi: 10.1016/j.entcs.2008.04.073]

[16] Ma XX, Cao C, Yu P, Zhou Y. A supporting environment based on graph grammar for dynamic software architectures. Ruan Jian Xue Bao/Journal of Software, 2008,19(8):1881−1892 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1881.htm [doi: 10.3724/SP.J.1001.2008.01881]

[17] Chang ZM, Mao XJ, Qi ZC. Applying bigraph theory to self-adaptive software architecture. Chinese Journal of Computers, 2009, 32(1):97−106 (in Chinese with English abstract). [doi: 10.3724/SP.J. 1016.2009.00097]

[18] Dormoy J, Kouchnarenko O, Lanoix A. Using temporal logic for dynamic reconfigurations of components. LNCS, 2012,6921: 200−217. [doi: 10.1007/978-3-642-27269-1_12]

[19] Fiadeiro JL, Lopes A. A model for dynamic reconfiguration in service-oriented architectures. Software & Systems Modeling, 2013, 12(2):349−367. [doi: 10.1007/s10270-012-0236-1]

[20] Pelliccione P, Inverardi P, Muccini H. CHARMY: A framework for designing and verifying architectural specifications. IEEE Trans. on Software Engineering, 2009,35(3):325−346. [doi: 10.1109/TSE.2008.104]

[21] Lanoix A, Dormoy J, Kouchnarenko O. Combining proof and model-checking to validate reconfigurable architectures. Electronic Notes in Theoretical Computer Science, 2011 279(2):43−57. [doi: 10.1016/j.entcs.2011.11.011]

[22] Eckardt T, Heinzemann C, Henkler S, Hirsch M, Priesterjahn C, Schäfer W. Modeling and verifying dynamic communication structures based on graph transformations. Computer Science—Research and Development, 2013,28(1):3−22. [doi: 10.1007/s00450-011-0184-y]

[23] Rensink A. The GROOVE simulator: A tool for state space generation. LNCS, 2004,3062:479−485. [doi: 10.1007/978-3-540-25959-6_40]

[24] Hermann F, Kastenberg H, Modica1 T. Towards translating graph transformation approaches by model transformations. Electronic Communications of the EASST, 2006. http://journal.ub.tu-berlin.de/eceasst/article/view/20

[25] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(S1):1907−1912 (in Chinese with English abstract). http://www.ejournal.org.cn/CN/Y2002/V30/IS1/1907

[26] Baier C, Katoen JP. Principles of Model Checking. Cambridge: MIT Press, 2008.

[27] Holzmann G. The model checker SPIN. IEEE Trans. on Software Engineering, 1997,23(5):279−295. [doi: 10.1109/32.588521]

附中文参考文献:

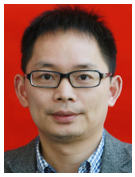
[8] 徐洪珍,曾国荪.基于超图文法的软件体系结构动态演化.同济大学学报(自然科学版),2011,39(5):45−750. [doi: 10.3969/j.issn.0253-374x.2011.05.021]

[9] 徐洪珍,曾国荪,陈波.软件体系结构动态演化的条件超图文法及分析.软件学报,2011,22(6):1210−1223. http://www.jos.org.cn/1000-9825/4017.htm [doi: 10.3724/SP.J.1001.2011.04017]

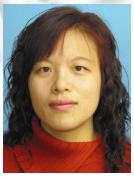
[16] 马晓星,曹春,余萍,周宇.基于图文法的动态软件体系结构支撑环境.软件学报,2008,19(8):1881−1892. http://www.jos.org.cn/1000-9825/19/1881.htm [doi: 10.3724/SP.J.1001.2008.01881]

[17] 常志明,毛新军,齐治昌.Bigraph 理论在自适应软件体系结构上的应用.计算机学报,2009,32(1):97−106. [doi: 10.3724/SP.J.1016.2009.00097]

[25] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(S1):1907−1912. http://www.ejournal.org.cn/CN/Y2002/V30/IS1/1907



徐洪珍(1976－),男,江西临川人,博士,教授,CCF 高级会员,主要研究领域为软件体系结构,软件演化,模型检测.



王晓燕(1980－),女,讲师,主要研究领域为软件体系结构,软件演化.

曾国荪(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为可信软件,并行分布处理.