软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2012,23(4):846−863 [doi: 10.3724/SP.J.1001.2012.04160]

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

# 一种软件演化过程模型的代数语义*

代 飞[1,2+], 李 彤[1,2], 谢仲文[1], 于 倩[1,2], 卢 萍[1], 郁 涌[1,2], 赵 娜[1,2]

[1](云南大学 软件学院,云南 昆明 650091)

[2](云南省软件工程重点实验室(云南大学),云南 昆明 650091)

## Towards an Algebraic Semantics of Software Evolution Process Models

DAI Fei[1,2+], LI Tong[1,2], XIE Zhong-Wen[1], YU Qian[1,2], LU Ping[1], YU Yong[1,2], ZHAO Na[1,2]

[1](School of Software, Yunnan University, Kunming 650091, China)

[2](Key Laboratory for Software Engineering of Yunnan Province (Yunnan University), Kunming 650091, China)

+ Corresponding author: E-mail: feidai@ynu.edu.cn, http://www.sei.ynu.edu.cn

**Dai F, Li T, Xie ZW, Yu Q, Lu P, Yu Y, Zhao N. Towards an algebraic semantics of software evolution process models. *Journal of Software*, 2012,23(4):846−863.** http://www.jos.org.cn/1000-9825/4160.htm

**Abstract**: As a number of software evolution process models increased, as modeled by EPMM (software evolution process meta model), verifying the correctness of these models becomes the important. This paper extends EPMM with ACP (algebra of communicating processes) and proposes EPMM-A (software evolution meta model-algebra). In order to discuss behavior verification in the unified framework of EPMM-A, a process term is used to define an algebraic semantics of a software evolution process model. Based on equational reasoning of EPMM-A, behavior verification of a software evolution process model emphasizes algebraic reasoning as opposed to model-based reasoning. This method combines the advantages of both Petri nets and ACP, which can effectively support software evolution process formal verification.

**Key words**: software evolution process; process verification; algebraic semantics; Petri net; ACP (algebra of communicating processes)

**摘 要**: 随着大量的软件演化过程模型被软件演化过程元模型建模产生,如何验证过程模型的正确性,是摆在人们面前的一个重要任务.针对软件演化过程元模型,引入进程代数 ACP(algebra of communicating processes)对其扩展,提出软件演化过程元模型代数,使用进程项指定软件演化过程模型的代数语义,在进程代数的统一框架下,基于等式推理验证软件演化过程模型的行为,使行为验证方式从模型推导变为代数推导.这种方法充分结合了 Petri 网和 ACP 的长处,可以有效地支持软件演化过程的形式验证.

**关键词**: 软件演化过程;过程验证;代数语义;Petri 网;ACP(algebra of communicating processes)

**中图法分类号**: TP311 **文献标识码**: A

“变化”是现实世界永恒不变的主题,只有“变化”,事物才能向前发展.软件是对现实世界中问题空间与解空间的具体描述,是客观事物的一种反映,是人类思维的外化产物[1].既然现实世界是不断变化发展的,那么软件也

* 基金项目: 国家自然科学基金(60963007); 云南省软件工程重点实验室开放基金(2010KS01, 2011SE04)
收稿时间: 2010-07-08; 修改时间: 2011-03-29; 定稿时间: 2011-11-30

必须随之而不断地变化发展.近年来,软件演化已成为软件生存周期中最重要的形态之一,成为今天软件工程研究的热点领域.如何有效支持软件系统的演化,是软件工程面临的一项重要挑战.一方面,支持软件演化的软件过程(简称软件演化过程)可以建立软件演化的整体任务框架,被视为是提高软件演化质量和效率的一种重要技术,受到学术界的关注和重视;另一方面,对建模产生的软件演化过程模型,如何验证过程模型的正确性,是摆在我们面前的一个重要任务.

在软件过程领域,建模和验证是两大关键活动.软件过程建模是对软件过程进行抽象和表示的活动,方式可以是非形式化、半形式化或形式化的,其目的是建立过程模型.通过对过程模型的认识和分析,增强对过程本身的理解,进而更好地支持软件开发.软件过程验证是检验软件过程模型与过程需求是否一致的活动,方式也可以是非形式化、半形式化或形式化的,其目的是验证过程模型的正确性、提高过程模型的可靠性,进而提高软件质量.与非形式化的建模和验证相比,形式化的建模和验证更加准确,有利于计算机实现,是开发半自动化或自动化验证工具的基础.对于软件过程建模,文献[2]作了很好的综述和总结.与软件过程建模相比,有关软件过程验证的研究相对较少.

文献[3]提出了软件演化过程元模型 EPMM(software evolution process meta model),用以建模软件演化过程,支持软件演化.EPMM 的提出,有效地解决了软件演化过程的形式建模问题.但是,随着软件演化过程模型被 EPMM 建模产生,软件演化过程模型的形式验证问题却尚未得到解决.

对于软件演化过程模型的验证,本文认为验证的属性可以分为 3 种:结构验证、性质验证和行为验证.结构验证是指检验过程模型的结构是否满足某些语法要求,属于静态验证;性质验证是指检验过程模型在执行过程中是否满足性质,属于动态验证;行为验证是指检验过程模型的行为是否与过程规约(见定义 13)规定的行为一致,是过程模型验证的最高层次.本文关注从代数语义的角度,支持软件演化过程模型的行为验证.

为了支持软件演化过程模型的行为形式验证,本文扩展了 EPMM,引入 ACP(algebra of communicating processes)[4,5]风格的进程代数,提出软件演化元模型代数 EPMM-A(software evolution process meta model-algebra,见第 3 节),既作为软件演化过程模型的代数语义域,又用来定义软件演化过程模型的过程规约;基于 EPMM-A,使用进程项(见定义 11)指定软件演化过程模型的代数语义,将基于 Petri 网的软件演化过程模型形式转换为进程项,进而在 EPMM-A 的统一框架下研究软件演化过程模型的行为验证,使行为验证方式从模型推导(见定义 20)转变为代数推导(见定义 21).其主要工作如下:

第一,从元语言的角度扩展 EPMM,引入 ACP 风格的进程代数,提出了 EPMM-A,既作为软件演化过程模型的代数语义域,又用来定义过程模型的过程规约;

第二,从进程理论的角度,将软件演化过程模型和进程项均看作描述软件演化过程行为的进程,并提出行为空间和行为图;

第三,从行为等价的角度,定义进程间的互模拟关系,作为行为验证的一个标准;

第四,从形式语义的角度,基于 EPMM-A,使用进程项指定软件演化过程模型的代数语义,并从互模拟关系的角度证明软件演化过程模型与其代数表示具有相同的行为空间,说明代数语义的正确性.

上述工作将 Petri 网和 ACP 有机地结合在一起,使用两种形式化方法分别刻画软件演化过程的不同方面:一方面,使用 Petri 网直观描述软件演化过程的结构,基于 Petri 网的相关技术,可以对软件演化过程模型的结构进行验证;另一方面,使用 EPMM-A 定义软件演化过程的过程规约,基于等式推理,可以对软件演化过程模型的行为进行符号演算,使验证方式从模型推导变为代数推导.

本文第 1 节回顾相关研究的进展情况.第 2 节介绍软件演化过程元模型.第 3 节提出软件演化过程元模型代数.第 4 节定义软件演化过程的行为空间.第 5 节定义软件演化过程模型的代数语义.第 6 节总结全文.

## 1 相关工作

软件过程验证包括对软件过程模型的结构检查、语法检查、性质检查和语义检查等.在现有的软件过程验证研究中,过程模型被验证的属性主要可以分为两种:(1) 结构验证;(2) 性质验证.

### 1.1 结构验证

在结构验证方面,相关文献大多关注过程模型自身的正确性,从静态角度检验过程模型是否满足建模者所期望的结构和语法限制.

Hsueh 等人提出了一个基于 UML 的方法来定义、验证和确认软件过程,在其建模环境下,组织构建一个软件过程模型将遵循 6 个步骤:过程模型定义、过程模型验证、模型转换、模型确认、过程教育(process education)、过程执行.在过程模型验证时,Hsueh 使用 OCL(object constraint language)来定义过程模型必须满足的限制和规则,其中,规则分为两种:过程相关的规则和 CMMI 相关的规则[6].

费立蜀等人基于工作流中的控制流模型提出了一个过程定义模型,这个模型结合了控制流模型、数据模型和组织模型来支持软件过程的定义;将软件过程验证分为两种情况:结构合理性验证和系统合理性验证;结构合理性又称为静态合理性,是指过程定义的静态结构没有结构冲突;验证时只对过程定义的控制流进行判断即可,不考虑数据和组织结构[7].

胡旷等人提出了一种以活动为中心的软件过程元模型,并以 XML 对其进行描述.他们认为,要建立正确的过程模型,不仅要保证过程模型语法的正确性,还应考虑过程模型语义的正确性;从控制流和数据流的角度定义了结构语义约束和数据语义约束对软件过程模型进行验证[8].

柳军飞等人提出了一种基于时序逻辑的形式化过程建模语言 XYZ/PME,该语言是时序逻辑语言 XYZ/E 的子语言,它支持以角色为中心的逐步求精的过程建模方法,可在统一的形式框架内表示不同抽象级的过程模型,并在统一的逻辑框架内验证过程模型的一致性[9].

Yoon 等人基于活动产品图(activity artifact graph)提出了一种系统的方法来定义和裁剪标准软件过程.根据项目需求对标准软件过程进行裁剪后,可以通过验证确定过程的相似度以及裁剪是否合理.在验证软件过程的过程中,Yoon 从静态验证的角度,提出了语法正确性检查、类型一致性检查和标准的合理性评价等手段[10].

Lee 等人基于 UML 的元模型提出了一种定义软件过程中任务分配策略的方法,将模型分为 3 类:元模型、概念模型和实例模型,并提供了一种方法可以检查用 UML 表示的过程中的不一致(inconsistency),并可以在必要的时候消除这种不一致[11].

其他过程语言,如 IDEF0[12],ProcessWeaver[13]和 Statemate[14]只允许使用静态类型验证.

### 1.2 性质验证

在性质验证方面,相关文献大多关注过程模型在执行过程中的某些性质,从动态角度检验过程模型是否满足建模者所期望的动态性质.

Cobleigh 等人使用 Little-JIL 形式定义软件过程和过程属性,在过程验证中,使用 FLAVERS(flow analysis for verification of systems)检验过程模型的行为是否满足过程相关的特定性质[15].本质上,这种验证方式是基于有限状态自动机的安全性质验证.但是,Little-JIL 语法支持的递归和异常处理机制会给过程模型验证带来很多问题.

Min 等人基于高级 Petri 网提出了一个以过程为中心的软件工程环境 SoftPM.SoftPM 支持 3 种不同抽象层次的过程描述,分别是认知模型、MAN 网和 Pr/T 网.SoftPM 除了使用 Petri 网相关的技术对软件过程模型进行死锁(deadlocks)、陷进(trap)和活性(liveness)等性质的验证以外,还采用其他技术对软件过程模型的动态性质进行分析和验证[16].

Yang 等人基于 Pi 演算提出了一种代数方法来检测软件过程模型的不一致性(inconsistencies),并帮助过程设计者有效定位和解决不一致性;为了减少 Pi 演算的复杂性,还设计了一种图像化的建模语言,并将软件过程模型的不一致性分为领域级不一致性(domain-level inconsistencies)和环境级不一致性(environmental inconsistencies)[17].

Wallace 使用 Alloy 建模组织的过程.Alloy 是由麻省理工大学开发的一种文本语言,在软件过程模型验证的过程中,Wallace 使用 SAT 验证过程模型是否满足特定性质,若不满足,则给出反例[18].

文献[19]使用软件过程建模语言 FUNSOFT 建模软件过程,并提出了一种基于计算机辅助验证的方法来验证过程模型的属性.与基于文档的分析方法相比[20],该方法不关注软件过程的结果,而是对软件过程模型具有的属性进行分析和验证.

Lerner 使用 LTST(模型检测器)对由软件过程建模语言 Little-JIL 描述的软件过程模型进行性质验证,整个过程需要从 Little-JIL 描述的软件过程模型中构建出用于模型检测的模型 FSP[21].模型检测的性质只限于死锁、活动和饥饿等,不能验证过程相关的性质.

### 1.3 总 结

总的来说,对于软件过程验证,相关文献主要集中在结构和性质层面,方法以语法检验和模型检测为主.但是,结构验证只能从静态角度对软件过程模型的结构和语法进行检验,无法确保动态性质的满足;性质验证大多只考虑行为某一方面的特性,无法确保过程模型的行为与过程规约一致.本文从代数语义的角度,支持软件演化过程模型的行为验证,目前还鲜有文献关注.

## 2 软件演化过程元模型

本文作者之一基于基本 Petri 网,扩展面向对象技术和 Hoare 逻辑,提出了 EPMM(软件演化过程元模型)[3],是本文过程模型验证的前提和基础.本质上,EPMM 是一个形式化的元模型,自顶向下分为 4 层:全局层、过程层、活动层和任务层.各层的形式定义如下:

**定义 1(任务)**[3]. 任务是一个四元组 $t=\langle\{Q_1\},\{Q_2\},M_t,M_0\rangle$,其中,

(1) $Q_1$ 和 $Q_2$ 是一阶谓词公式;$\{Q_1\}$称为前断言,用于定义任务 $t$ 执行前的状态;$\{Q_2\}$称为后断言,用于定义任务 $t$ 执行后的状态;

(2) $A(F)=\langle\{Q_1\},\{Q_2\}\rangle$是一个 2-断言,用于定义任务 $t$ 的功能;

(3) $M_t$ 是接收消息的集合,当任务 $t$ 获得 $M_i$ 中的一个或多个消息时,任务 $t$ 被执行;

(4) $M_0$ 是发送消息的集合,$\forall m\in M_0, m=(r,b)$,表示当任务 $t$ 被执行时,$t$ 发送一个消息 $m$ 到 $r$,$b$ 称为消息体,包含许多参数.

**定义 2(活动)**[3]. 活动是一个四元组 $a=\langle I,O,L,B\rangle$,其中,

(1) $I$,$O$ 和 $L$ 分别称为活动的输入数据结构、输出数据结构和局部数据结构;

(2) $B$ 称为活动体,既可以是一个软件过程 $p$,也可以是任务 $t_{main},t_1,t_2,\ldots,t_n$ 的集合,这些任务或软件过程在数据结构 $I$,$O$ 和 $L$ 上执行.任务 $t_{main}$ 是在活动中首先获得消息 *execution* 并加以执行的特殊任务;

(3) 活动的定义是一个类的定义,称为活动类.当活动被执行时,活动对象就被创建.

**定义 3(软件过程系统)**[3]. 软件过程系统是一个四元组 $\Omega=(C,A;F,M)$,其中,

(1) $\langle C,A,F\rangle$是一个没有孤立元素的网,即 $A\cup C\neq\varnothing$;

(2) $C$ 是一个条件的有限集,$\forall c\in C$ 称为一个条件;

(3) $A$ 是一个活动的有限集,$\forall a\in A$ 称为一个活动;元素 $a$ 的发生称为 $a$ 被执行或者 $a$ 点火;

(4) $M\subseteq 2^C$ 称为 $\Sigma$ 的实例类,其中 $2^C$ 表示 $C$ 的幂集;

(5) $\forall a\in A,\exists m\in M$,以致 $a$ 在 $m$ 可以发生.

**定义 4(软件过程)**[3]. 令 $\Omega=(C,A;F,M)$是一个软件过程系统,设 $M_0\in M(M_0\subseteq C)$是 $\Omega$ 的一个实例,且 $p=(C,A;F,M_0)$.$M_0$ 称为 $p$ 的初始情态,元素 $d\in M_0$ 称为一个托肯,$p$ 称为一个软件过程.

**定义 5(全局模型)**[3]. 全局模型是一个二元组 $g=\langle P,E\rangle$,其中,

(1) $P$ 是一个软件过程的集合;

(2) $E\subseteq P\times P$ 是一个偏序的二元关系,称为 $P$ 的嵌入关系;$E=\{(p,p')|p,p'\in P\wedge p'$嵌入到 $p$ 中$\}$,$p'$称为 $p$ 的子过程.

本质上,由 EPMM 建模产生的软件演化过程模型是一个扩展的基本 Petri 网,条件集 $C$ 中的元素只有“有托肯”和“无托肯”两种状态.根据点火规则,软件演化过程模型可以被执行.

## 3 软件演化过程元模型代数

### 3.1 软件演化过程元模型代数

从形式语义的角度来看,为了使用进程项来指定软件演化过程模型的代数语义,本节首先扩展 EPMM,引入 ACP 风格的进程代数,提出了 EPMM-A(软件演化过程元模型代数),既作为软件演化过程模型的代数语义域,又用来定义过程模型的过程规约.本质上,EPMM-A 是进程代数 ACP 的扩展.

形式语义学中,通常把程序设计语言称为对象语言,把用以解释或说明程序设计语言的语言称为元语言.同样地,本文把 EPMM 看作对象语言,把用以解释软件演化过程模型的 EPMM-A 看作元语言.EPMM-A 包含 3 部分内容:第 1 部分是类子和常量,第 2 部分是操作符,第 3 部分是公理,其定义见表 1.

**Table 1** EPMM-A

**表 1** EPMM-A

| | | | | |
|---|---|---|---|---|
| *Sorts*: | | | | |
| $T,A,P;A\subseteq P$ | | | | |
| *Constants*: | | | | |
| $\delta\in P$ | *inaction*(*deadlock*); $t_{main}\in T$ | | the main task | |
| *Functions*: | | | | |
| $\_+\_:P\times P\to P$ | alternative composition or sum | | | |
| $\_\bullet\_:P\times P\to P$ | sequential composition or product | | | |
| $\_\|\|\_:P\times P\to P$ | parallel composition or merge | | | |
| $\_\lfloor\_:P\times P\to P$ | left merge | | | |
| $\_\|\_:T\times T\to A$ | synchronisation merge | | | |
| $\_^*:P\to P$ | resursion merge | | | |
| $\lambda\_(\_):M\times P\to A$ | causualstate operator | | | |
| *Axioms*: | | | | |
| $x,y,z:P;\ a,b,c:A;\ t_1,t_2,t_3:T;$ | | | | |
| $x+y=y+x$ | A1 | | $t_1\mid t_2=t_2\mid t_1$ | S1 |
| $(x+y)+z=x+(y+z)$ | A2 | | $(t_1\mid t_2)\mid t_3=t_1\mid(t_2\mid t_3)$ | S2 |
| $x+x=x$ | A3 | | $x^*y=x\bullet(x^*y)+y$ | BKS1 |
| $(x+y)\bullet z=x\bullet z+y\bullet z$ | A4 | | $x^*(y\cdot z)=(x^*y)\bullet z$ | BKS2 |
| $(x\bullet y)\bullet z=x\bullet(y\bullet z)$ | A5 | | $x^*(y\bullet((x+y)^*z)+z)=(x+y)^*z$ | BKS3 |
| $x+\delta=x$ | A6 | | | |
| $\delta\bullet x=\delta$ | A7 | | $x\|\|y=x\lfloor y+y\lfloor x$ | M1 |
| $x\|\|y=y\|\|x$ | C1 | | $a\lfloor x=a\cdot x$ | M2 |
| $(x\|\|y)\|\|z=x\|\|(y\|\|z)$ | C2 | | $(a\bullet x)\lfloor y=a\bullet(x\|\|y)$ | M3 |
| $(x\lfloor y)\lfloor z=x\lfloor(y\|\|z)$ | C3 | | $(x+y)\lfloor z=x\lfloor z+y\lfloor z$ | M4 |
| $\lambda_M(\delta)=\delta$ | | CSO1 | $\lambda_M(x+y)=\lambda_M(x)+\lambda_M(y)$ | CSO4 |
| $\dot{a}\subseteq M\wedge\dot{a}\cap M=\varnothing\Rightarrow\lambda_M(a)=a$ | | CSO2 | $\lambda_M(a\bullet x)=\lambda_M(a)\bullet\lambda_{(M-\dot{a})\cup a^{\cdot}}(x)$ | CSO5 |
| $\dot{a}!\subseteq M\vee\dot{a}\cap M\neq\varnothing\Rightarrow\lambda_M(a)=\delta$ | | CSO3 | | |

对于 EPMM-A 的非形式化描述如下所示:

首先,介绍 EPMM-A 中的类子和常量.变量 $T$ 表示任务集,变量 $A$ 表示活动集,变量 $P$ 表示进程集,分别对应于 EPMM 中的任务层、活动层和过程层,且活动集 $A$ 包含在进程集 $P$ 内.进一步地,进程集 $P$ 用来解释 EPMM 中的软件过程,活动集 $A$ 用来解释 EPMM 中的活动,任务集 $T$ 用来解释 EPMM 中的任务,即 $P$,$A$,$T$ 是对象语言 EPMM 的论域.$x$,$y$ 和 $z\in P$ 表示进程变量;$a$,$b$ 和 $c\in A$ 表示活动常量;$t_1$,$t_2$ 和 $t_3\in T$ 表示任务变量;常量 $\delta$ 表示死进程,是指不再执行任何活动的过程;常量 $t_{main}$ 表示活动内部首先执行的任务.

其次,介绍 EPMM-A 中的操作符(算子)和公理.首先介绍操作符:•(顺序复合操作符)、+(选择复合操作符)和||(并发复合操作符).设 $x$ 和 $y$ 是进程,则:(1) 顺序复合进程 $x$•$y$ 表示成功执行 $x$ 后再执行 $y$,通常 $x$•$y$ 简记为 $xy$;

(2) 选择复合进程 $x+y$ 表示要么执行 $x$ 要么执行 $y$;(3) 并发复合进程 $x||y$ 表示 $x$ 和 $y$ 的并发执行.对于+,选择复合进程 $x+y$ 的执行,需要与外部环境交互,从而做出选择.此外,+满足交换律和结合律,见公理 A1 和公理 A2.公理 A3 说明,进程 $x$ 选择复合进程 $x$ 等价于进程 $x$ 本身.对于+,顺序复合满足右分配率,见公理 A4.公理 A5 说明,顺序复合满足结合律.公理 A6 说明,进程 $x$ 选择复合 $\delta$ 等于进程 $x$ 自身.公理 A7 说明,$\delta$ 顺序复合进程 $x$ 等于 $\delta$.

在 EPMM-A 中,活动由任务通过|(同步复合操作符)复合而成.|是文献[22]中通信合并操作符的简化形式,公理 S1 和公理 S2 分别对应于文献[22]中的公理 C1 和公理 C2,更名的原因在于 EPMM-A 中不考虑显式通信.注意,在本文的过程规约中,活动 $a$ 具有如下形式:$t_1|t_2|t_3|\ldots|t_n,n>0,i\in N,t_i\in T$,其中,$N$ 表示自然数集,$T$ 表示任务集合.

在并发理论中,并发性(concurrent)是指一个系统内部事件之间没有因果关系.因果关系并不等同于事件先后关系,有因果关系必有先后关系,反之则未必.基于对并发性的不同理解,历史上出现过真并发语义和交织语义.真并发语义认为[23]:并发性是指系统中没有统一的时钟,内部发生的两个事件之间不存在因果关系.交织语义[24]认为:并发性是指系统内部事件执行的先后顺序的不确定性.也就是说,如果系统的行为是每个进程行为的所有可能的交错复合,那么,这些进程中全体事件的集合一定可以按执行时间的先后排成一个全序.在进程代数中,人们拥护交织语义,故而存在一个经典的公理:$a||b=a\bullet b+b\bullet a$,意思是:从观察者的角度,把系统看作是黑盒子,与之交互时,并发复合活动 $a$ 和 $b$ 的行为执行效果等价于先执行 $a$ 再执行 $b$,或者先执行 $b$ 再执行 $a$ 的行为执行效果.本文中 EPMM-A 支持交织语义.这样做的好处是:在刻画行为语义够用的前提下,使用交织语义,数学处理比较简单.其缺点是:失去了软件演化过程模型本身的真并发性,无法体现不确定性选择活动和真并发活动之间的区别.在支持交织语义的前提下,本文可以使用 $\llcorner$ (左并发复合操作符)将进程间的并发关系进行简化,见公理 M1.注意,在公理 $x\|y=x\llcorner y+y\llcorner x$ 中,并没有考虑进程 $x$ 与 $y$ 之间的显式通信,原因是,在 Petri 网中,一旦 $x$ 和 $y$ 是真并发的,则两者之间必定没有显式通信.$\llcorner$ 与||大致相同,唯一的差别在于 $\llcorner$ 要求先执行左边进程的活动,然后再并发复合左边的进程和右边的进程,见公理 M2.公理 M3 说明,活动 $a$ 左并发进程 $x$ 的执行效果等价于先执行活动 $a$ 再执行进程 $x$.对+而言,$\llcorner$ 具有右分配率,见公理 M4.公理 C1 和公理 C2 说明,||分别满足交换率和结合律.公理 C3 说明,$\llcorner$ 具有结合律.

此外,在并发理论中,线性时间和分时时间也是人们常常争论的问题之一.一方面,拥护线性时间的人们认为,等式 $a\bullet(b+c)=a\bullet b+a\bullet c$ 是有效的.其理由是:从观察者的角度,先看到 $a$ 再看到 $b$ 或者 $c$ 的行为执行效果等价于先看到 $a$ 再看到 $b$,或者先看到 $a$ 再看到 $c$ 的行为执行效果;另一方面,拥护分时时间的人们认为,等式 $a\bullet(b+c)=a\bullet b+a\bullet c$ 是无效的,而等式 $(a+b)\bullet c=a\bullet c+b\bullet c$ 是有效的.其理由是:在并发系统中,死锁、中断机制等性质与进程间做出选择的时间有关.本文中 EPMM-A 支持分时时间,故而,等式 $(a+b)\bullet c=a\bullet c+b\bullet c$ 作为公理 A4 存在.$\lambda$(因果状态操作符)借鉴了文献[25]所定义的状态操作符,用来定义软件演化过程模型在情态 $M$ 处发生点火操作时所执行的活动.它具有两个参数:下标表示软件演化过程模型的情态,参数表示描述软件演化过程模型的进程项.公理 CSO1 说明,死进程 $\delta$ 在任何情态下都无法发生点火操作.若活动 $a$ 在情态 $M$ 是使能的,则 $\lambda_M(a)=a$,见公理 CSO2;若活动 $a$ 在情态 $M$ 不是使能的,则 $\lambda_M(a)=\delta$,见公理 CSO3.公理 CSO4 说明,进程 $x+y$ 在情态 $M$ 的点火情况等价于进程 $x$ 在情态 $M$ 的点火情况或进程 $y$ 在情态 $M$ 的点火情况.公理 CSO5 说明,进程 $a\bullet x$ 在情态 $M$ 的点火情况等价于活动 $a$ 在情态 $M$ 的点火情况,再顺序复合进程 $x$ 在情态 $(M-\dot{}a)\cup a\dot{}$ 的点火情况.

对于*(递归复合),用来描述进程的递归操作.文献[26]提出了*的公理 BKS1,BKS2 和 BKS3,本文将它们引入到 EPMM-A 中.

### 3.2 基本概念

在 EPMM-A 的上下文中,定义了基调、开项、闭项、进程项和软件演化元模型代数等基本概念.

**定义 6(基调)**. EPMM-A 上的基调 $\Sigma$ 由类子:过程集 $P$、活动集 $A$、任务集 $T$ 和一系列操作符+,•,||,$\llcorner$,|,*,$\lambda$_() 组成,每个操作符 $f$ 的操作数记为 $ar(f)$,其中,+代表选择复合操作符,•代表顺序复合操作符,||代表并发复合操作符,$\llcorner$ 代表左并发复合操作符,|代表同步复合,*代表递归复合操作符,$\lambda$_()代表因果状态操作符.

通常认为,含有 0 个操作数的操作符称为常量,含有 1 个操作数的操作符称为一元操作符,含有 2 个操作数

的操作符称为二元操作符.

**定义 7(开项集)**. 令$\Sigma$是 EPMM-A 上的基调,基调$\Sigma$上的开项集(open terms)$\prod(\Sigma)$是满足以下条件的最小集合:

(1) 每个常量都是开项集中的元素;

(2) $V\subseteq\prod(\Sigma)$;

(3) 如果$f\in\Sigma$,且$v_1,v_2,\ldots,v_{ar(f)}\in\prod(\Sigma)$,那么$f(v_1,v_2,\ldots,v_{ar(f)})\in\prod(\Sigma)$.

**定义 8(开项)**. 令$\Sigma$是 EPMM-A 上的基调,$\prod(\Sigma)$是基调$\Sigma$上的开项集,若$u\in\prod(\Sigma)$,则称$u$为开项(open term).

开项的定义也可采用结构归纳定义,表示成以下形式:

$$u::=c|x|u_1+u_2|u_1\bullet u_2|u_1\|u_2|u_1\llcorner u_2|u_1*u_2|\lambda_M(u_1),$$

其中,::=表示归纳定义,$c$表示常量,$x$表示变量,$u_1$和$u_2$表示开项.

**定义 9(闭项集)**. 令$\Sigma$是 EPMM-A 上的基调,$\prod(\Sigma)$是基调$\Sigma$上的开项集,若$\forall u\in\prod(\Sigma)$,且$u$不包含变量,则称$\prod(\Sigma)$是闭项集(closed terms),记为$T(\Sigma)$.

**定义 10(闭项)**. 令$\Sigma$是 EPMM-A 上的基调,$T(\Sigma)$是基调$\Sigma$上的闭项集,若$w\in T(\Sigma)$,则称$w$为闭项(closed term).

开项和闭项的区别在于:闭项是不包含变量的开项.

**定义 11(软件演化过程元模型进程项)**. 在EPMM-A中,软件演化过程元模型进程项是指由活动集$A$中的元素通过+,•,||,$\llcorner$,*,$\lambda$_()复合而成的闭项,简称为进程项(process term).

**定义 12(软件演化元模型代数)**. 由所有软件演化过程元模型进程项组成的集合称为软件演化过程元模型代数.

### 3.3 软件演化元模型代数的模型

形式语义学建立在元语言的基础上.因此,只有元语言具有严密的数学理论基础,才能严格定义对象语言的语义、验证语义的合理性.故而,基于 EPMM-A 使用进程项定义软件演化过程模型的代数语义之前,必须首先讨论它的语义.

EPMM-A 的形式语义是指应在一个数学域上给出 EPMM-A 的语义解释,并且使得它的公理在这种解释下是有效的(valid).在进程代数中,变迁规则定义了进程项的操作语义,由进程项的操作语义组成的行为空间(见定义 15),是进程代数的一个语义解释模型.同样地,对于 EPMM-A,由变迁规则所定义的进程项的操作语义,即变迁(见定义 14)所组成的行为空间,也是EPMM-P的一个语义解释模型.因此,定义语言解释模型的关键问题是:给出 EPMM-A 的变迁规则,并说明 EPMM-A 的论域和解释关系(对象语言和元语言之间的映射关系).

EPMM-A 的变迁规则见表 2.它分别定义了顺序复合操作符•、选择复合操作符+、并发复合操作符||、左并发复合操作符$\llcorner$、递归操作符*和状态操作符$\lambda$_()的变迁规则.在变迁规则中,$T$是任务集,$A$是活动集,$P$是进程集.在变迁规则中:(1) $a$是活动常量,$p,p',q,q'$是过程变量;(2) $p\xrightarrow{a}p'$是变迁,表示进程$p$执行活动$a$后演化为进程$p'$;(3) 活动集$A$中的活动由任务集$T$中的任务同步复合而成.

论域是解释 EPMM-A 的符号和语法对象的载体;解释关系是一种映射关系,将对象语言中的符号和语法对象映射为论域中的元素.在行为空间中,解释 EPMM-A 的符号和语法对象的论域有:$T$,$A$和行为空间,其中,$T$表示任务集,$A$表示活动集.对 EPMM-A 而言,进程集$P$中的元素解释为行为空间;活动集$A$解释为活动集$A$本身;任务集$T$解释为任务集$T$本身,即活动集和任务集意思不变;公理系统推导出的等式关系解释为进程间的互模拟关系.

需要注意的是,使用元语言对对象语言的语法元素进行解释时,通常情况下,两种语言的论域不同,因此,语言的身份也就很好区分.但是在少数情况下,对象语言和元语言可以共享同一论域.典型的例子是《新华字典》,字典中的条目是汉语,是被解释的对象,属于对象语言的范畴;而用以解释条目的文字也是汉语,属于元语言的范畴,用于说明条目的含义.本文中 EPMM-A 和行为空间共享论域$T$和$A$.这种现象就是在同一论域和解释中,一种语言身份的双重性.

**Table 2** Transition rules of EPMM-A

**表 2** EPMM-A 的变迁规则

$T;\ a:A;\ \ p,p',q,q'\in P$

$$a\xrightarrow{a}\surd \qquad \frac{p\xrightarrow{a}p'}{p\bullet q\xrightarrow{a}p'\bullet q} \qquad \frac{p\xrightarrow{a}\surd}{p\bullet q\xrightarrow{a}q}$$

$$\frac{p\xrightarrow{a}p'}{p+q\xrightarrow{a}p'} \qquad \frac{q\xrightarrow{a}q'}{p+q\xrightarrow{a}q'} \qquad \frac{p\xrightarrow{a}\surd}{p+q\xrightarrow{a}\surd} \qquad \frac{q\xrightarrow{a}\surd}{p+q\xrightarrow{a}\surd}$$

$$\frac{p\xrightarrow{a}p'}{p\parallel q\xrightarrow{a}p'\parallel q} \qquad \frac{q\xrightarrow{a}q'}{p\parallel q\xrightarrow{a}p\parallel q'} \qquad \frac{p\xrightarrow{a}\surd}{p\parallel q\xrightarrow{a}q} \qquad \frac{q\xrightarrow{a}\surd}{p\parallel q\xrightarrow{a}p}$$

$$\frac{p\xrightarrow{a}p'}{p\between q\xrightarrow{a}p'\parallel q} \qquad \frac{p\xrightarrow{a}\surd}{p\between q\xrightarrow{a}q}$$

$$\frac{p\xrightarrow{a}p'}{p*q\xrightarrow{a}p'\bullet(p*q)} \qquad \frac{q\xrightarrow{a}q'}{p*q\xrightarrow{a}q'} \qquad \frac{p\xrightarrow{a}\surd}{p*q\xrightarrow{a}p*q} \qquad \frac{q\xrightarrow{a}\surd}{p*q\xrightarrow{a}\surd}$$

$$\frac{p\xrightarrow{a}p'}{\lambda_{M\cup\cdot a}(p)\xrightarrow{a}\lambda_{M\cup a}(p')} \qquad \frac{p\xrightarrow{a}\surd}{\lambda_{M\cup\cdot a}(p)\xrightarrow{a}\surd}$$

## 4 软件演化过程的行为空间

从进程理论的角度对软件演化过程的行为进行研究,提出了过程规约、行为空间和行为图,并定义了一个行为比较的标准:互模拟关系,具体思路如下:

首先,从过程需求的角度,明确提出过程规约,用以描述软件演化过程的行为特征.

其次,为了使用数学规则对进程进行符号演算,需要将进程描述为数学域中的元素,称此域为进程域.文献[27]认为,进程域有 6 种:图域(graph domain)、网域(net domain)、事件结构域(event structure domain)、显示域(explicit domain)、投影限制域(projective limit domain)和项域(term domain).在本文中,进程的数学域有两种:网域和项域.具体而言,在EPMM的上下文中,进程是软件演化过程模型,其数学域是网域;在EPMM-A的上下文中,进程是进程项,其数学域是项域,如图 1 所示.好处在于:一方面,可以使用 Petri 网直观地描述软件演化过程的结构;另一方面,可以使用 EPMM-A 的等式推理验证软件演化过程的行为.

然后,使用 Plotkin 式的结构化操作方法[28]分别定义软件演化过程模型和进程项的行为空间.

最后,从进程语义的角度提出互模拟关系,刻画进程间的行为等价性,作为行为验证的一个标准.

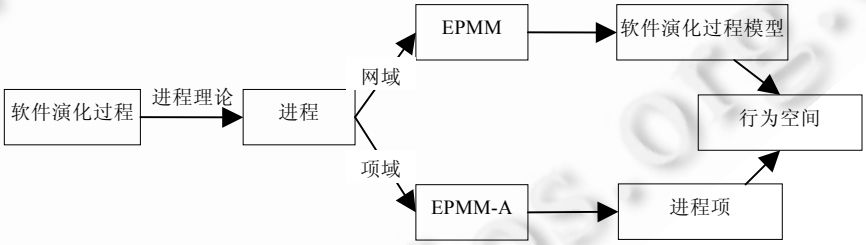



Fig.1 Two mathematical domains of a process

图 1 进程的两种数学域

### 4.1 过程规约

在软件开发过程中,需求描述对于项目的成败至关重要,远远超过了其他因素的影响.同样地,在软件过程工程中,过程需求对过程建模和过程验证也是至关重要的.如果在需求处理过程中有错误未能解决,则后续的过程建模和过程验证都会受到影响:(1) 从过程建模的角度看,过程规约位于软件过程工程的起始阶段,作为建模指南,可以帮助建模者更好地理解所建过程;(2) 从过程验证的角度看,过程规约是建模者和使用者根据过程需求而规定的软件过程的行为特征,作为验证指南,是软件过程验证的必要条件.从这个意义上讲,现阶段的过程验证缺乏行为验证的一个重要原因是:从过程需求的角度,没有明确提出过程规约.

因此,为了保证过程需求的完整性和一致性,更好地支持过程模型的行为验证,十分有必要提出过程规约,见定义 13.方式可以是非形式化的、半形式化的和形式化的.

**定义 13(过程规约)**. 过程规约是建模者和使用者根据过程需求而规定的软件过程的行为特征,可以将完整、一致的过程需求与能够满足过程需求的过程行为以文档的方式明确地固定下来.

从形式验证的角度看,软件演化过程模型的行为验证必然涉及形式的过程模型和形式的过程规约.一方面,由 EPMM 建模产生的软件演化过程模型本身是形式化的,本文不予考虑;另一方面,对于如何形式定义过程规约,通常可以使用时序逻辑或进程代数.若使用时序逻辑描述过程规约,则行为验证的核心问题便是研究基于 Petri 网的模型检测问题.但是,基于 Petri 网的模型检测常常由于发生状态空间爆炸,以至难以判定.因此,本文将使用 EPMM-A 定义过程规约.

例子:假设软件演化过程模型开始执行活动 $a$,随后执行活动 $b$,最后并发执行 $c$ 和 $d$.根据过程需求,基于 EPMM-A,使用进程项,过程规约可以描述为 $a•b•(c\|\|d)$,其中,•表示顺序复合,‖表示并发复合.

### 4.2 行为空间

在进程理论中,系统是指客观世界里的物体(包括具体的和抽象的)依据某种方式与环境进行交互,如机器、软件系统、网络协议等;进程是系统与环境交互的一种抽象,是系统的行为模型;行为又由变迁(操作)构成,变迁(操作)的执行使得系统由一个状态转换为另一个状态,见定义 14.

**定义 14(变迁)**. 变迁是一个三元组 $b=(p,a,p')$,形如 $p \xrightarrow{a} p'$,其中,

(1) $p$ 和 $p'$表示进程,$p,p'\in P$,$P$ 是进程集;

(2) $a$ 表示活动,$a\in A$,$A$ 是活动集.

变迁 $p \xrightarrow{a} p'$ 表示进程 $p$ 执行活动 $a$ 后演化为进程 $p'$;变迁 $p \xrightarrow{a} \surd$ 表示进程 $p$ 执行活动 $a$ 后,能够成功终止;变迁 $p \xrightarrow{a} \delta$ 表示进程 $p$ 执行活动 $a$ 后变为死进程.

注意,这里 $P$ 是一个抽象概念.在 EPMM 的上下文中,进程的数学域是网域,$P$ 指的是软件演化过程模型的集合;在 EPMM-A 中,进程的数学域是项域,$P$ 指的是进程项的集合.

为了以统一的方式比较进程间的行为,下面提出行为空间和行为图,见定义 15 和定义 16.

**定义 15(行为空间)**. 行为空间是一个二元组 $PBS=(P,B)$,其中,

(1) $P$ 是进程集;

(2) $B$ 是变迁集.

**定义 16(行为图)**. 行为图是一个三元组 $BG=(P,E,R)$,其中,

(1) $P$ 是进程集;

(2) $E=\{(p_i,p_j)\mid p_i,p_j\in P,\exists a\in A:(p_i,a,p_j)\in B\}$,其中,$A$ 是活动集,$B$ 是变迁集;

(3) $R:E\to A$,$R(p_i,p_j)=a$ 当且仅当$(p_i,a,p_j)\in B$;

(4) 选 $p_i\in P$ 作为顶点.

称 $P$ 中的元素为节点,$E$ 中的元素描述进程间的执行关系,$R$ 中的元素是进程执行的活动,$p_i$ 为行为图的顶点.

从定义 16 可以看出,本质上,行为图是带顶点的行为空间.与行为空间相比,行为图更加直观,便于比较进程间的行为.

例子:一个系统 PA 的行为空间定义如下:

- $P=\{p_1,p_2,p_3,p_4,\surd\}$;
- $B=\{(p_1,a,p_2),(p_2,b,p_3),(p_3,c,p_4),(p_4,d,\surd)\}$.

若选 $p_1$ 为根节点,则 PA 的行为图定义如下:

- $P=\{p_1,p_2,p_3,p_4,\surd\}$;
- $E=\{(p_1,p_2),(p_2,p_3),(p_3,p_4),(p_4,\surd)\}$;
- $R=\{a,b,c,d\}$.

在行为图的图形表示中,圆圈表示 $P$ 中的元素,有向弧表示 $E$ 中的元素,旁标表示 $R$ 中的元素,则 PA 的行为图如图 2 所示.

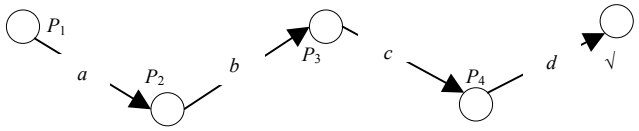



Fig.2 A behavior graph of PA

图 2 PA 的行为图

### 4.3 软件演化过程模型的行为空间

在 EPMM 的上下文中,软件演化过程的行为空间由软件演化过程模型集及点火操作所决定的变迁集构成,分析如下:

令 $p=(N,M_0)$是由 EPMM 建模产生的软件演化过程模型,其中,$N=(C,A;F)$是软件演化过程模型的结构,$M_0$ 是情态.根据 Petri 网的点火规则,活动 $a\in A$ 是使能的当且仅当${}^{\bullet}a\subseteq M_0\wedge a^{\bullet}\cap M_0=\varnothing$.如果活动 $a$ 是使能的,则 $a$ 可以发生点火操作.一旦活动 $a$ 在情态 $M_0$ 处点火,则会产生新的情态 $M_0'$,且 $M_0'=(M_0-{}^{\bullet}a)\cup a^{\bullet}$.也就是说,活动 $a$ 发生点火操作(变迁)后,一个软件演化过程模型 $p=(N,M_0)$就演化为另一个软件演化过程模型 $p'=(N,M_0')$.

基于上述分析可知,对于软件演化过程模型而言,点火规则定义了过程模型间的点火操作.从进程理论的角度看,软件演化过程模型的点火规则就是进程的变迁规则.

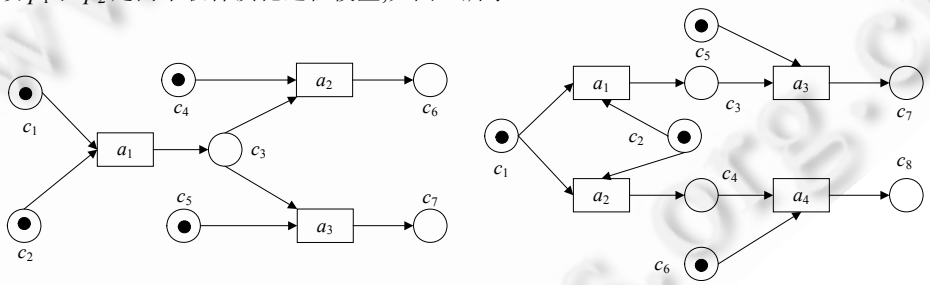
**定义 17(软件演化过程模型的变迁)**. 软件演化过程模型 $p=(N,M_0)$,活动 $a$ 在情态 $M_0$ 处是使能的,则其变迁是一个三元组 $b_m=(p,a,p')$,其中,

(1) $p$ 和 $p'$表示软件演化过程模型,且 $p,p'\in PTN$;$PTN$ 是所有软件演化过程模型的集合;

(2) $a$ 表示活动,且 $a\in A$,$A$ 是所有活动的集合.

变迁$(p,a,p')$表示软件演化过程模型 $p=(N,M_0)$在情态 $M_0$ 处发生点火操作,执行活动 $a$ 后,演化为软件演化过程模型 $p'=(N,M_0')$,其中,$M_0'=(M_0-{}^{\bullet}a)\cup a^{\bullet}$.

例子:设 $p_1$ 和 $p_2$ 是两个软件演化过程模型,如图 3 所示.




软件演化过程模型$p_1$ 软件演化过程模型$p_2$

Fig.3 Software evolution process models $p_1$ and $p_2$

图 3 软件演化过程模型 $p_1$ 和 $p_2$

根据 Petri 网的点火规则,可以得到软件演化过程模型 $p_1$ 和 $p_2$ 的行为空间.

$p_1$ 的 $PBS_1=\{(N_1,M_{10})\xrightarrow{a_1}(N_1,M_{11}),(N_1,M_{11})\xrightarrow{a_2}(N_1,M_{12}),(N_1,M_{11})\xrightarrow{a_3}(N_1,M_{13})\}$,其中,$N_1$ 是软件演化过程模型 $p_1$ 的结构,其定义如下:

- $C=\{c_1,c_2,c_3,c_4,c_5,c_6,c_7\}$;
- $A=\{a_1,a_2,a_3\}$;
- $F=\{(c_1,a_1),(c_2,a_1),(a_1,c_3),(c_3,a_2),(c_4,a_2),(c_3,a_3),(c_5,a_3),(a_2,c_6),(a_3,c_7)\}$;
- $M_{10},M_{11},M_{12}$ 和 $M_{13}$ 是情态,$M_{10}=(c_1,c_2,c_4,c_5)$,$M_{11}=(c_3,c_4,c_5)$,$M_{12}=(c_5,c_6)$,$M_{13}=(c_4,c_7)$.

$p_2$ 的 $PBS_2$={ $(N_2,M_{20})\xrightarrow{a_1}(N_2,M_{21})$, $(N_2,M_{21})\xrightarrow{a_3}(N_2,M_{23})$, $(N_2,M_{20})\xrightarrow{a_2}(N_2,M_{22})$, $(N_2,M_{22})\xrightarrow{a_4}$, $(N_2,M_{24})$},其中,$N_2$ 是软件演化过程模型 $p_2$ 的结构,其定义如下:

- $C=\{c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8\}$;
- $A=\{a_1,a_2,a_3,a_4\}$;
- $F=\{(c_1,a_1),(c_1,a_2),(c_2,a_1),(c_2,a_2),(a_1,c_3),(c_3,a_3),(c_5,a_3),(a_3,c_7),(a_2,c_4),(c_4,a_4),(c_6,a_4),(a_4,c_8)\}$;
- $M_{20},M_{21},M_{22},M_{23}$ 和 $M_{24}$ 是情态,$M_{20}=(c_1,c_2,c_5,c_6),M_{21}=(c_3,c_5,c_6),M_{22}=(c_4,c_5,c_6),M_{23}=(c_6,c_7),M_{24}=(c_5,c_8)$.

从 $p_1$ 的行为空间中选择 $N_1$ 当根节点,从 $p_2$ 的行为空间中选择 $N_2$ 当根节点,软件演化过程模型 $p_1$ 和 $p_2$ 的行为图如图 4 所示.

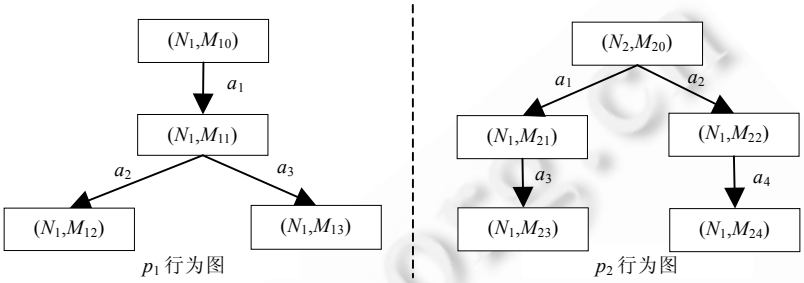



Fig.4 Behavior graphs of software evolution process models $p_1$ and $p_2$

图 4 软件演化过程模型 $p_1$ 和 $p_2$ 的行为图

### 4.4 进程项的行为空间

在EPMM-A的上下文中,软件演化过程的行为空间由进程项集和EPMM-A中满足变迁规则的变迁集构成.而 EPMM-A 上的变迁集由表 2 中给出的变迁规则所决定.

设 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$和$(a_1\bullet a_3)+(a_2\bullet a_4)$是 EPMM-A 的进程项,其中,$a_1,a_2,a_3,a_4\in A$,$\delta$是死进程.

根据 EPMM-A 的变迁规则,$a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的行为空间为

$$\{a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)\xrightarrow{a_1}(a_2\bullet\delta+a_3\bullet\delta),(a_2\bullet\delta+a_3\bullet\delta)\xrightarrow{a_2}\delta,(a_2\bullet\delta+a_3\bullet\delta)\xrightarrow{a_3}\delta\}.$$

$(a_1\bullet a_3)+(a_2\bullet a_4)$的行为空间为 $\{(a_1\bullet a_3)+(a_2\bullet a_4)\xrightarrow{a_1}a_3,a_3\xrightarrow{a_3}\surd,(a_1\bullet a_3)+(a_2\bullet a_4)\xrightarrow{a_2}a_4,a_4\xrightarrow{a_4}\surd\}$.

同时,选取 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta),(a_1\bullet a_3)+(a_2\bullet a_4)$作为顶点,进程 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$和$(a_1\bullet a_3)+(a_2\bullet a_4)$的行为图如图 5 所示.

从图4和图5可以看出,软件演化过程模型$p_1$行为图中的旁标与进程项$a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$行为图中的旁标相同.从行为空间的角度看,$p_1$ 的行为和 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的行为是等价的.两者差别在于描述进程的数学方式不同:$p_1$ 的进程域是网域,而 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的进程域是项域.可以这样理解:虽然软件演化过程模型 $p_1$ 和进程项 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的数学描述方式不同,但描述的是同一个软件演化过程的行为.这个发现不仅有助于定义进程间的互模拟关系(见第 4.5 节),还有助于使用进程项指定软件演化过程模型的代数语义(见第 5.1 节).

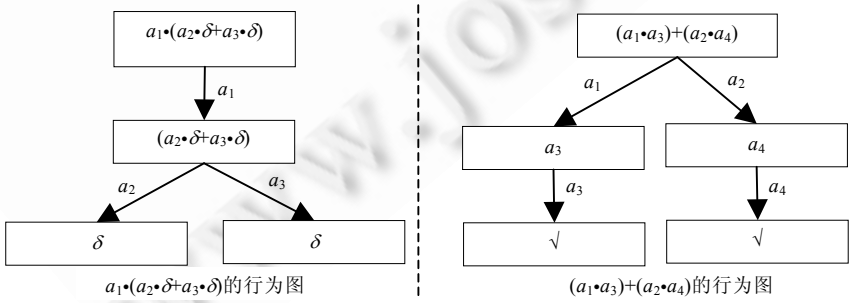



Fig.5 Behavior graphs of $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$ and $(a_1\bullet a_3)+(a_2\bullet a_4)$

图 5 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$和$(a_1\bullet a_3)+(a_2\bullet a_4)$的行为图

### 4.5 互模拟关系

所谓行为等价性是指两个不同的系统具有相同的行为.这个问题不仅是一个理论问题,对于实践也至关重要.Milner 认为,只有当我们清楚了什么是行为的相似或不同,我们才能声称我们了解“行为”的意义;在此之前,我们甚至无法准确地说明我们的系统是做什么的[29].只有解决了这个理论问题,在对软件演化过程模型进行行为验证时,才能找到行为比较的标准.也就是说,当过程模型的行为与过程规约规定的行为满足什么样的比较标准时,过程模型才被认为是正确的.

系统的行为属性涉及多个方面,如执行顺序、随机性和时间属性等.建模者应根据应用环境和关注面,定义不同的并发语义.根据关注面的不同,并发语义可以分为 4 种[27]:(1) 线性时间和分时时间的对比;(2) 交织语义和偏序语义的对比;(3) 进程内部活动的不同抽象处理;(4) 对递归的不同表示方法.文献[30]对并发语义做出了详细的总结和比较.为了刻画软件演化过程间的行为是否等价,本文借鉴了 Park 提出的双模拟思想[28],定义了进程间的互模拟关系,从分时时间、交织语义、对递归的不同表示这 3 个方面来区分并发语义.文献[31,32]认为,互模拟关系是行为等价关系中最好的一种,原因在于:(1) 在理论上,互模拟关系统一了进程代数中各种等价性概念,不仅要求进程执行相同的操作,还必须具有相同的分支结构;(2) 在实践上,为过程模型的行为验证提供了一个行为比较的标准.

基于上述认识,本文采用互模拟关系作为行为比较的一个标准.

**定义 18(互模拟关系)**. 互模拟关系$\mathfrak{R}$是直积 $P\times P$ 的子集当且仅当对于任意的 $p,p',q,q'\in P$ 和 $a\in A$ 有:

(1) $p\mathfrak{R}q\wedge p\xrightarrow{a}p'\Rightarrow(\exists q':q'\in P:q\xrightarrow{a}q'\wedge p'\mathfrak{R}q')$;

(2) $p\mathfrak{R}q\wedge q\xrightarrow{a}q'\Rightarrow(\exists p':p'\in P:p\xrightarrow{a}p'\wedge p'\mathfrak{R}q')$;

(3) $p\mathfrak{R}q\Rightarrow p\xrightarrow{a}\surd\Leftrightarrow q\xrightarrow{a}\surd$.

其中,$P$ 是进程集,$A$ 是活动集.

两个进程是互模拟的,记为 $p\sim q$.

同样地,这里的进程集 $P$ 是一个抽象概念.在 EPMM 的上下文中,进程的数学域是网域,$P$ 指的是软件演化过程模型的集合;在 EPMM-A 中,进程的数学域是项域,$P$ 指的是进程项的集合.

由定义 18 可知:(1) 两个进程是互模拟的当且仅当两者可以相互模拟对方的行为;(2) 若两个进程是互模拟的,两者既具有相同的行为,又具有相同的分支结构.

软件演化过程模型 $p_1$ 和进程项 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的行为图,如图 6 所示.

为了直观起见,图 6 中将存在互模拟关系的进程用虚线连接起来.不难证明,这些进程之间存在互模拟关系:

$$(N_1,M_{10})\mathfrak{R}(a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)),(N_1,M_{11})\mathfrak{R}(a_2\bullet\delta+a_3\bullet\delta),(N_1,M_{12})\mathfrak{R}\delta,(N_1,M_{13})\mathfrak{R}\delta.$$

值得注意的是,若软件演化过程模型 $p_1$ 在情态 $M_{12}$ 和 $M_{13}$ 处没有任何活动可以发生点火操作,则$(N_1,M_{12})$和$(N_1,M_{13})$是死进程.

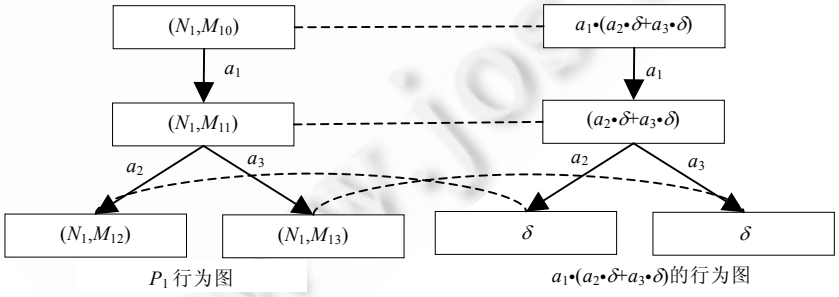



Fig.6 Behavior graphs of a software evolution process model $p_1$ and a process term $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$

图 6 软件演化过程模型 $p_1$ 和进程项 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的行为图

## 5 软件演化过程模型的代数语义

为了从形式语义的角度,支持软件演化过程模型的行为验证,并使验证方式从模型推导转变为代数推导,本节主要讨论软件演化过程模型的代数语义和代数推导.

首先讨论定义软件演化过程模型代数语义的基本思路.从进程理论的角度看,本文使用两种进程对软件演化过程的行为加以刻画,即由EPMM建模产生的软件演化过程模型和基于EPMM-A定义的进程项.这两种进程语法不同,但描述的是同一个软件演化过程的行为,即行为语义相同.因而,可以使用一种进程的行为去解释另一种进程的行为,如图7所示.

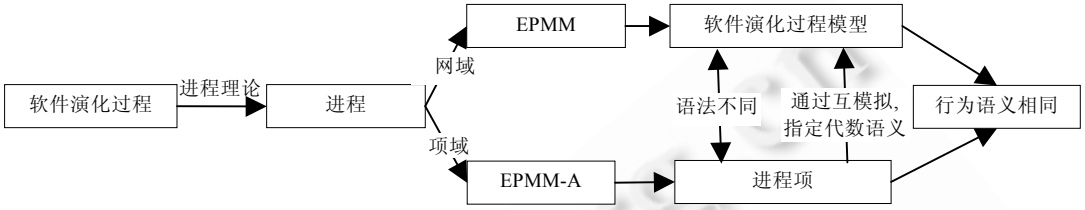



Fig.7 A basic idea

图7 基本思路

其次,为了说明软件演化过程模型代数语义的正确性,从互模拟关系的角度,需要证明软件演化过程模型与其代数表示具有相同的行为空间.

最后,为了把软件演化过程模型的行为验证方式从模型推导转变为符号演算,支持软件演化过程的形式验证,提出了代数推导.

### 5.1 代数语义

从图7可以看出,软件演化过程模型 $p_1$ 和进程项 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$的数学形式不同.但从互模拟关系的角度,可以证明两者是互模拟的,即行为空间相同.故而,可以使用进程项 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$来解释软件演化过程模型 $p_1$ 的行为.在此认识的基础上,本文借鉴文献[33]的思想,定义了软件演化过程模型的代数语义.

**定义19(代数语义)**. 设$\Sigma_1=(N_1,M_1)$是一个由EPMM建模产生的软件演化过程模型,其中$N_1=(C,A;F)$.$\Sigma_1$的代数语义记为 $S[\Sigma_1]$,定义如下:$S[\Sigma_1]=\lambda_{M_1}(\|a_i{:}a_i\in A{:}V[a_i])$,其中,$a_i=t_1|t_2|t_3|\ldots|t_n,n>0,0<i<n,i,n\in N,t_i\in T,V[a_i]=(t_1|t_2|t_3|\ldots|t_n)*\delta=a_i*\delta,\|a_i{:}a_i\in A{:}V[a_i]=(V[a_1]\|V[a_2]\|\ldots\|V[a_i],i\in N)$,$A$是活动集,$T$是任务集,$N$是自然数集.

从定义19可以看出,在EPMM-A的上下文中,一个软件演化过程模型在情态$M$处的行为由所有使能活动的并发执行所描述,而一个使能活动的行为又由同步合成的任务的无数次迭代执行所描述.

为了说明定义19的正确性,从互模拟关系的角度,需要证明软件演化过程模型与其代数表示具有相同的行为空间,见定理1.

**定理1**. 任给一个由EPMM建模产生的软件演化过程模型$\Sigma_1=(N_1,M_1)$和$\Sigma_1'=(N_1,M_2)$,其中,$N_1=(C,A,F)$表示软件演化过程模型的结构,存在进程项$p$和活动$a_1\in A$,满足以下条件:

(1) $\Sigma_1=(N_1,M_1)\xrightarrow{a_1}\Sigma_1'=(N_1,M_2)\Rightarrow S[\Sigma_1]\xrightarrow{a_1}S[\Sigma_1']$;

(2) $S[(N_1,M_1)]\xrightarrow{a_1}p\Rightarrow(\exists M_2:M_2\in PowC:p=S[(N_1,M_2)]\wedge(N_1,M_1)\xrightarrow{a_1}(N_1,M_2))$.

其中,$PowC$表示条件集$C$的幂集.

证明:

(1) 假设$\Sigma_1=(N_1,M_1)\xrightarrow{a_1}\Sigma_1'=(N_1,M_2)$.根据软件演化过程模型的变迁定义,存在一个活动$a_1\in A$,且$a_1=t_1|t_2|t_3|\ldots|t_n,n>0,t_i\in T$,满足$a_1$在情态$M_1$是使能的,即${}^\bullet a_1\subseteq M_1\wedge a_1{}^\bullet\cap M_1=\varnothing$.$a_1$发生点火操作后,情态由$M_1$变为$M_2$,且$M_2=(M_1-{}^\bullet a_1)\cup a_1{}^\bullet$.也就是说,软件演化过程模型$\Sigma_1$执行活动$a_1$后,演变为软件演化过程模型$\Sigma_1'$.因此,活动$a_1$的执行可以成功终止,则有$a_1\xrightarrow{a_1}\surd$.根据表2中的变迁规则$\dfrac{p\xrightarrow{a}\surd}{p*q\xrightarrow{a}p*q}$,令$p=a_1,a=a_1,q=\delta$,则

$a_1*\delta \xrightarrow{a_1} a_1*\delta$ 成立.

根据定义 19,活动 $a_1$ 的代数语义记为 $V[a_1]=(t_1|t_2|t_3|\ldots|t_n)*\delta=a_1*\delta$,其中,$t_i\in T$,$0<i<n$,$n\in N$.因为 $a_1\in A$,根据表 2 中并发复合操作符的变迁规则 $\dfrac{p\xrightarrow{a}p'}{p\parallel q\xrightarrow{a}p'\parallel q}$,令 $p=a_1*\delta$,$a=a_1$,$q=(\|a_i{:}a_i\in A\wedge a_i\neq a_1{:}V[a_i])$,又 $a_1*\delta \xrightarrow{a_1} a_1*\delta$ 成立,则 $(\|a_i:a_i\in A{:}V[a_i]) \xrightarrow{a_1} (\|a_i:a_i\in A{:}V[a_i])$ 成立.根据表 1 中因果状态操作符$\lambda$的定义以及条件${}^\bullet a_1\subseteq M_1\wedge a_1{}^\bullet\cap M_1=\varnothing$,则 $\lambda_{M_1}(a_i:a_i\in A{:}T[a_i]) \xrightarrow{a_1} \lambda_{M_2}(\|a_i:a_i\in A{:}T[a_i])$ 成立,其中,$M_2'=(M_1-{}^\bullet a_1)\cup a_1{}^\bullet$.

令 $S[(N_1,M_1)]=\lambda_{M_1}(a_i:a_i\in A{:}V[a_i])$ 和 $S[(N_1,M_2)]=\lambda_{M_2}(\|a_i:a_i\in A{:}V[a_i])$,则有 $S[\Sigma_1]\xrightarrow{a_1}S[\Sigma_1']$.

综上所述,条件(1)得证.

(2) 假设 $S[(N_1,M_1)]\xrightarrow{a_1}p$.根据定义 19,存在活动 $a_1\in A$,$a_1=t_1|t_2|t_3|\ldots|t_n$,$n>0$,$t_i\in T$,有

$$\lambda_{M_1}(\|a_i:a_i\in A{:}V[a_i])\xrightarrow{a_1}p.$$

根据表 1 中因果状态操作符$\lambda$的定义,活动 $a_1$ 必满足${}^\bullet a_1\subseteq M_1\wedge a_1{}^\bullet\cap M_1=\varnothing$,否则,变迁无法发生.对于软件演化过程模型$(N_1,M_1)$而言,由于活动 $a_1$ 满足点火规则,即 $a_1$ 满足${}^\bullet a_1\subseteq M_1\wedge a_1{}^\bullet\cap M_1=\varnothing$,所以软件演化过程模型$(N_1,M_1)$可以发生点火操作,执行活动 $a_1$.根据软件演化过程模型的变迁定义,记为 $(N_1,M_1)\xrightarrow{a_1}(N_1,M_2)$.

根据表 2 中因果状态操作符$\lambda$的变迁规则定义,由 $\lambda_{M_1}(\|a_i:a_i\in A{:}V[a_i])\xrightarrow{a_1}p$ 可得 $p=\lambda_{M_2}(q)$ 和 $(\|a_i:a_i\in A{:}V[a_i])\xrightarrow{a_1}q$,其中,$q$ 是进程项,$M_2=(M_1-{}^\bullet a_1)\cup a_1{}^\bullet$.

根据表 2 中并发复合操作符的变迁规则定义,由 $(\|a_i:a_i\in A{:}V[a_i])\xrightarrow{a_1}q$ 可以推出 $q=b\|(\|a_i{:}a_i\in A\wedge a_i\neq a_1{:}V[a_i])$和 $V[a_1]\xrightarrow{a_1}b$.因为活动 $a_1$ 的代数语义为 $V[a_1]=(t_1|t_2|t_3|\ldots|t_n)*\delta=a_1*\delta$.根据 EPMM-A 的变迁规则 $\dfrac{p\xrightarrow{a}\surd}{p*q\xrightarrow{a}p*q}$,令 $p=a_1$,$q=\delta$,由假设 $S[(N_1,M_1)]\xrightarrow{a_1}p$ 可知活动 $a_1\xrightarrow{a_1}\surd$ 成立,所以 $a_1*\delta\xrightarrow{a_1}a_1*\delta$ 成立.因此,$b=V[a_1]=a_1*\delta$,$a_1*\delta$.继而,$q=b\|(\|a_i{:}a_i\in A\wedge a_i\neq a_1{:}V[a_i])=(\|a_i{:}a_i\in A{:}V[a_i])$.

将 $q=b\|(\|a_i{:}a_i\in A\wedge a_i\neq a_1{:}V[a_i])=(\|a_i{:}a_i\in A{:}V[a_i])$代入 $p=\lambda_{M_2}(q)$ 可得 $p=\lambda_{M_2}(\|a_i{:}a_i\in A{:}V[a_i])$.

所以,$p=S[(N_1,M_2)]$成立.

综上所述,条件(2)得证.

综上条件(1)、条件(2)所述,定理 1 得证. □

定理 1 说明,从互模拟关系的角度来看,一个软件演化过程模型与其代数表示具有相同的行为空间.也就是说,一个软件演化过程模型的行为可以被其代数表示所模拟,反之也一样.

为了简便起见,今后在证明的过程中使用 EPMM-A 中的公理 C2 时简记为 C2.

Petri 网和进程代数都可以描述并发系统的动态行为.两者的区别在于:Petri 网描述真并发语义;进程代数支持交织语义.因此,在使用进程项指定软件演化过程模型的代数语义过程中,核心问题是要将 Petri 网描述的真并发语义形式转换为 EPMM-A 所支持的交织语义,见定理 2.

**定理 2**. 对任意的 $x_0,x_1,\ldots,x_n\in P$,$n>0$,有 $(\|i:0\leqslant i\leqslant n:x_i)=(+i:0\leqslant i\leqslant n:x_i\llcorner(\|j:0\leqslant j\leqslant n\wedge j\neq i:x_j))$.

证明:

$n=0$ 时,等式成立 $x_0=x_0$;

$n=1$ 时,由公理 M1 可知,$x_0\parallel x_1=x_0\llcorner x_1+x_1\llcorner x_0$ 等式成立;

设 $n=k>1$ 时,$(\|i:0\leqslant i\leqslant k:x_i)=(+i:0\leqslant i\leqslant k:x_i\llcorner(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j))$ 等式成立;

当 $n=k+1$ 时:

(1) $(\|i:0\leqslant i\leqslant n=k+1:x_i)=(\|i:0\leqslant i\leqslant k:x_i)\parallel x_{k+1}$ C2

(2) $(\|i:0\leqslant i\leqslant k:x_i)\parallel x_{k+1}=(\|i:0\leqslant i\leqslant k:x_i)\llcorner x_{k+1}+x_{k+1}\llcorner(\|i:0\leqslant i\leqslant k:x_i)$ M1

(3) $(\|i:0\leqslant i\leqslant k:x_i)\parallel x_{k+1}=(+i:0\leqslant i\leqslant k:x_i\llcorner(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\llcorner x_{k+1})+x_{k+1}\llcorner(\|i:0\leqslant i\leqslant k:x_i)$ 归纳假设

(4) $(+i:0\leqslant i\leqslant k:x_i^{\llcorner}(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)^{\llcorner}x_{k+1})=$
$(+i:0\leqslant i\leqslant k:x_i^{\llcorner}(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\|x_{k+1})$ M4 和 C3

(5) $(+i:0\leqslant i\leqslant k:x_i^{\llcorner}(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\|x_{k+1})=$
$(+i:0\leqslant i\leqslant k:x_i^{\llcorner}((\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\|x_{k+1}))$ C2

(6) $(\|i:0\leqslant i\leqslant k:x_i)\|x_{k+1}=(+i:0\leqslant i\leqslant k:x_i^{\llcorner}((\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\|x_{k+1}))+x_{k+1}^{\llcorner}(\|i:0\leqslant i\leqslant k:x_i)$
用上面(4)替换(3)中的 $(+i:0\leqslant i\leqslant k:x_i^{\llcorner}(\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j))^{\llcorner}x_{k+1})$

(7) $(+i:0\leqslant i\leqslant k:x_i^{\llcorner}((\|j:0\leqslant j\leqslant k\wedge j\neq i:x_j)\|x_{k+1}))+x_{k+1}^{\llcorner}(\|i:0\leqslant i\leqslant k:x_i)=$
$(+i:0\leqslant i\leqslant k:x_i^{\llcorner}(\|:0\leqslant j\leqslant k+1\wedge j\neq i:x_j)+x_{k+1}^{\llcorner}(\|i:0\leqslant i\leqslant k:x_i)=$
$(+i:0\leqslant i\leqslant n=k+1:x_i^{\llcorner}(\|j:0\leqslant j\leqslant n\wedge j\neq i:x_j))$ 数学代换

综上所述,定理 2 得证. □

**推论 1**. 对任意的非空原子活动集 $E\subseteq A$,有$(\|a:a\in E:a*\delta)=(+a:a\in E:a\bullet(\|a:a\in E:a*\delta))$.

证明:

若 $E$ 只含有一个元素 $a$,由公理 BKS1 可知,$a*\delta=a\bullet(a*\delta)$等式成立.

若 $E$ 含有多个元素:

(1) $(\|a:a\in E:a*\delta)=(+a:a\in E:(a*\delta)^{\llcorner}(\|e:e\in E\wedge e\neq a:e*\delta))$ 定理 2

(2) $(+a:a\in E:(a*\delta)^{\llcorner}(\|e:e\in E\wedge e\neq a:e*\delta))=$
$(+a:a\in E:a\bullet(a*\delta)^{\llcorner}(\|e:e\in E\wedge e\neq a:e*\delta))$ BKS1 和 A6

(3) $(+a:a\in E:a\bullet(a*\delta)^{\llcorner}(\|e:e\in E\wedge e\neq a:e*\delta))=$
$(+a:a\in E:a\bullet((a*\delta)\|(\|e:e\in E\wedge e\neq a:e*\delta)))$ M3

(4) $(+a:a\in E:a\bullet((a*\delta)\|(\|e:e\in E\wedge e\neq a:e*\delta)))=(+a:a\in E:\ a\bullet(\|e:e\in E:e*\delta))$ C2

综上所述,推论 1 得证. □

例子:令 $p_1=(N_1,M_1)$是软件演化过程模型,其中,$M_1=(c_1,c_2,c_4,c_5)$,$A=(a_1,a_2,a_3)$,如图 3 中的 $p_1$ 所示.根据定义 19,$p_1$ 的代数语义为 $S[p_1]$.

(1) $S[p_1]=\lambda_{M_1}(\|a_i:a_i\in A:T[a_i])$ 定义 19

(2) $\lambda_{M_1}(\|a_i:a_i\in T:T[a_i])=a_1*\delta\|a_2*\delta\|a_3*\delta$ 定义 19

(3) $a_1*\delta\|a_2*\delta\|a_3*\delta=a_1\bullet X+a_2\bullet X+a_3\bullet X$,其中,$X=a_1*\delta\|a_2*\delta\|a_3*\delta$ 推论 1

(4) $\lambda_{M_1}(\|a_i:a_i\in A:T[a_i])=\lambda_{M_1}(a_1\bullet X+a_2\bullet X+a_3\bullet X)$,其中,$M_1=(c_1,c_2,c_4,c_5)$ 将(3)代入(2)

(5) $\lambda_{M_1}(a_1\bullet X+a_2\bullet X+a_3\bullet X)=\lambda_{M_1}(a_1\bullet X)+\lambda_{M_1}(a_2\bullet X)+\lambda_{M_1}(a_3\bullet X)$ CSO4

(6) $\lambda_{M_1}(a_1\bullet X)+\lambda_{M_1}(a_2\bullet X)+\lambda_{M_1}(a_3\bullet X)=a_1\bullet\lambda_{M_2}(X)+\delta\bullet\lambda_{M_1}(X)+\delta\bullet\lambda_{M_1}(X)$ CSO2 和 CSO3
其中,$M_2=(c_3,c_4,c_5)$

(7) $\delta\bullet\lambda_{M_1}(X)=\delta$ A7

(8) $a_1\bullet\lambda_{M_2}(X)+\delta\bullet\lambda_{M_1}(X)+\delta\bullet\lambda_{M_1}(X)=a_1\bullet\lambda_{M_2}(X)+\delta+\delta$ 将(7)代入(6)

(9) $a_1\bullet\lambda_{M_2}(X)+\delta+\delta=a_1\bullet\lambda_{M_2}(X)$ A6

(10) $a_1\bullet\lambda_{M_2}(X)+\delta\bullet\lambda_{M_1}(X)+\delta\bullet\lambda_{M_1}(X)=a_1\bullet\lambda_{M_2}(X)$ 将(9)代入(8)

(11) $\lambda_{M_1}(a_1\bullet X)+\lambda_{M_1}(a_2\bullet X)+\lambda_{M_1}(a_3\bullet X)=a_1\bullet\lambda_{M_2}(X)$ 将(10)代入(6)

(12) $\lambda_{M_2}(X)=\lambda_{M_2}(a_1*\delta\|a_2*\delta\|a_3*\delta)=\lambda_{M_2}(a_1\bullet X+a_2\bullet X+a_3\bullet X)$ 推论 1

(13) $\lambda_{M_2}(a_1\bullet X+a_2\bullet X+a_3\bullet X)=\delta\bullet\lambda_{M_2}(X)+a_2\bullet\lambda_{M_3}(X)+a_3\bullet\lambda_{M_4}(X)$ CSO2 和 CSO3
其中,$M_3=(c_5,c_6)$,$M_4=(c_4,c_7)$

(14) $\delta\bullet\lambda_{M_2}(X)=\delta$ A7

(15) $\lambda_{M_3}(X)=\delta$ 推导步骤如上,省略

(16) $\lambda_{M_4}(X)=\delta$ 推导步骤如上,省略

(17) $\lambda_{M_2}(a_1\times X+a_2\times X+a_3\times X)=\delta+a_2\times\delta+a_3\times\delta$ 将(14)~(16)代入(13)

(18) $\delta+a_2\bullet\delta+a_3\bullet\delta=a_2\bullet\delta+a_3\bullet\delta$ A6

(19) $\lambda_{M_2}(a_1\bullet X+a_2\bullet X+a_3\bullet X)=a_2\bullet\delta+a_3\bullet\delta$ 将(18)代入(17)

(20) $\lambda_{M_1}(a_1\bullet X)+\lambda_{M_1}(a_2\bullet X)+\lambda_{M_1}(a_3\bullet X)=a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$ 将(18)代入(17)

(21) $\lambda_{M_1}(a_1\bullet X+a_2\bullet X+a_3\bullet X)=a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$ 将(20)代入(5)

(22) $\lambda_{M_1}(\|a_i:a_i\in A:T[a_i])=a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$ 将(21)代入(4)

(23) $S[p_1]=a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$ 将(22)代入(1)

由此,软件演化过程模型 $p_1$ 的代数语义为 $a_1\bullet(a_2\bullet\delta+a_3\bullet\delta)$.

### 5.2 代数推导

互模拟关系的提出,为软件演化过程模型的行为验证找到了一个行为比较的标准.也就是说,当仍给一个由 EPMM 建模产生的软件演化过程模型和一个基于 EPMM-A 定义的过程规约,判断过程模型的行为与过程规约定义的行为是否等价时,可以分别构建各自的行为图,根据互模拟关系的定义进行行为比较.本文把这种方式称为模型推导,见定义 20.

**定义 20(模型推导)**. 当判断一个由 EPMM 建模产生的软件演化过程模型的行为与一个基于 EPMM-A 定义的过程规约规定的行为是否等价时,需要分别构建各自的行为图,找到互模拟关系,进行行为比较,这种行为验证方式称为模型推导.

从定义 20 可以看出,模型推导需要构建行为图,找出互模拟关系,工作量大,不利于数学推导,也不利于开发计算机辅助设计工具.显然,这种行为验证方式缺乏严格的形式化基础,具有相当的随意性.

为了把对软件演化过程模型的行为验证方式转变为符号演算,本文提出了代数推导,即在 EPMM-A 的上下文中,通过等式推理,判断两个进程项的行为是否等价,这种方式称为代数推导,见定义 21.

**定义 21(代数推导)**. 在 EPMM-A 的上下文中,使用等式关系判定两个进程项的行为是否等价,这种方式称为代数推导.

从定义 21 可以看出,代数推导不需要构建进程的行为图,也不需要找出互模拟关系,只需使用 EPMM-A 的公理系统进行等式推导.与模型推导相比,代数推导有利于支持形式验证,提高验证效率.

## 6 结束语

为了支持文献[3]中所提出的软件演化过程模型的推理验证,本文扩展了 EPMM,引入 ACP 风格的进程代数,提出了 EPMM-A,既作为软件演化过程模型的代数语义域,又用来定义过程模型的过程规约;基于 EPMM-A,使用进程项指定软件演化过程模型的代数语义,并从互模拟关系的角度证明软件演化过程模型与其代数表示具有相同的行为空间,说明了代数语义的正确性;基于代数语义,将基于 Petri 网的软件演化过程模型形式转换为进程项,进而在 EPMM-A 的统一框架下研究过程模型的行为验证,使验证方式从模型推导转变为代数推导.总之,软件演化过程模型的代数语义有效地支持了软件演化过程的形式验证.

由于篇幅有限,本文只讨论了软件演化过程模型的代数语义,并未给出过程模型的结构、性质和行为验证的相关算法,也没有给出 EPMM-A 的可靠性证明,我们将在后续工作中进一步研究和阐述这些问题.

**References**:

[1] Yang FQ. Thinking on the development of software engineering technology. Journal of Software, 2005,16(1):1−7 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/1.htm

[2] Li MS, Yang QS, Zhai J. Systematic review of software process modeling and analysis. Journal of Software, 2009,20(3):524−545 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3432.htm [doi: 10.3724/SP.J.1001.2009.03432]

[3] Li T. An Approach to Modelling Software Evolution Processes. Berlin: Springer-Verlag, 2008.

[4] Bergstra JA, Klop JW. Process algebra for synchronous communication. Information and Control, 1984,60(1-3):109−137. [doi: 10.1016/S0019-9958(84)80025-X]

[5] Baeten JCM, Weijland WP. Process Algebra. Cambridge: Cambridge University Press, 1990.

[6] Hsueh NL, Shen WH, Yang ZW, Yang DL. Applying UML and software simulation for process definition, verification, and validation. Information and Software Technology, 2008,50(9-10):897−911. [doi: 10.1016/j.infsof.2007.10.015]

[7] Fei LS, Gu Q, Chen DX. A process definition model and its verification analysis. Computer Science, 2004,31(1):145−515 (in Chinese with English abstract).

[8] Hu K, Dong GZ, Tian Y, Liu JF. Flexible verification tool for software process model. Computer Engineering and Design, 2007, 28(7):1497−1500 (in Chinese with English abstract).

[9] Liu JF, Tang ZS. A study on software modeling languages. Journal of Software, 1996,7(8):449−457 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19960801.htm

[10] Yoon IC, Min SY, Bae DH. Tailoring and verifying software process. In: Proc. of the 8th Asia-Pacific Software Engineering Conf. on APSEC 2001. Washington: IEEE Computer Society, 2001. 202−209. [doi: 10.1109/APSEC.2001.991478]

[11] Lee SJ, Shim J, Wu CS. A meta model approach using UML for task assignment policy in software process. In: Proc. of the 9th Asia-Pacific Software Engineering Conf. on APSEC 2002. Washington: IEEE Computer Society, 2002. 376−382. [doi: 10.1109/APSEC.2002.1183007]

[12] National Institute of Standards and Technology. Integrated definition for function modeling (IDEF0). Federal Information Processing Standards 183, 1993.

[13] Fernström C. PROCESS WEAVER: Adding process support to UNIX. In: Proc. of the 2nd Conf. on the Software Process. Berlin: IEEE Computer Society, 1993. 12−26. [doi: 10.1109/SPCON.1993.236825]

[14] Harel D, Lachover H, Naamad A, Pnueli A, Politi M, Sherman R, Shtul-Trauring A, Trakhtenbrot M. STATEMATE: A working environment for the development of complex reactive systems. IEEE Trans. on Software Engineering, 1990,16(4):403−414. [doi: 10.1109/32.54292]

[15] Cobleigh JM, Clark LA, Osterweil LJ. Verifying properties of process definitions. ACM SIGSOFT Software Engineering Notes, 2000,25(5):96−101. [doi: 10.1145/347636.348876]

[16] Min SY, Lee HD, Bae DH. SoftPM: A software process management system reconciling formalism with easiness. Information and Software Technology, 2000,42(1):1−16. [doi: 10.1016/S0950-5849(99)00050-6]

[17] Yang QS, Li MS, Wang Q, Yang GW, Zhai J, Li J, Hou LS, Yang Y. An algebraic approach for managing inconsistencies in software processes. In: Proc. of the 1st Int'l Conf. on Software Processes. LNCS 4470, Berlin, Heidelberg: Springer-Verlag, 2007. 121−133. http://dl.acm.org/ citation.cfm?id=1763253 [doi: 10.1007/978-3-540-72426-1_11]

[18] Wallace C. Using alloy in process modelling. Information and Software Technology, 2003,45(15):1031−1043. [doi: 10.1016/S0950-5849(03)00131-9]

[19] Bröckers A, Gruhn V. Computer-Aided verification of software process model properties. In: Proc. of the Advanced Information Systems Engineering. London: Springer-Verlag, 1993. 521−546. http://dl.acm.org/citation.cfm?id=676532

[20] Basili VR, Rombach HD. Tailoring the software process to project goals and environments. In: Proc. of the 9th Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society, 1987. 345−357. http://dl.acm.org/citation.cfm?id=41804&dl=ACM&coll=DL&CFID=70432015&CFTOKEN=96597694

[21] Lerner BS. Model checking of software processes. Proc. of FSE, 2003,16(4):403−414.

[22] Bergstra JA, Klop JW. The algebra of recursively defined processes and the algebra of regular processes. In: Proc. of the ICALP'84. LNCS 172, London: Springer-Verlag, 1984. 82−95. http://www.springerlink.com/content/713l6833223m07p5/ [doi: 10.1007/3-540-13345-3_7]

[23] Lu RQ. Formal Semantics of Computer Languages. Beijing: Science Press, 1992 (in Chinese).

[24] Milner R. Synthesis of communicating behaviour. In: Proc. of the 7th. Int'l Symp. on Foundations of Computer Science. LNCS 64, Zakopane: Springer-Verlag, 1978. 71−83. http://www.springerlink.com/content/p263q137402168m2/ [doi: 10.1007/3-540-08921-7_57]

[25] Baeten JCM, Bergstra JA. Non interleaving process algebra. In: Proc. of the CONCUR'93. LNCS 715, Berlin, Heidelberg: Springer-Verlag, 1993. 308−323. http://www.springerlink.com/content/747l50t277845455/about/ [doi: 10.1007/3-540-57208-2_22]

[26] Bergstra JA, Bethke I, Ponse A. Process algebra with iteration and nesting. The Computer Journal, 1994,37(4):243−258. [doi: 10.1093/comjnl/37.4.243]

[27] Bergstra JA, Ponse A, Smolka SA. Handbook of Process Algebra. New York: Elsevier Science Inc., 2001.

[28] Plotkin GD. A structural approach to operational semantics. Technical Report, DAIMIFN-19, Departmnt of Computer Science, Aarhus University, 1981.
[29] Milner R. Communicating and Mobile Systems the Pi-Calculus. Cambridge: Cambridge University Press, 1999.
[30] van Glabbeek RJ. The linear time branching time spectrum. In: Proc. of the 1st Conf. on Concurrency Theory (CONCUR'90). LNCS 458, Berlin, Heidelberg: Springer-Verlag, 1990. 278−297. http://www.springerlink.com/content/44222x44m525510q/ [doi: 10. 1007/BFB0039066]
[31] De Bakker JW, Zucker JI. Processes and the denotational semantics of concurrency. Information and Control, 1982,54(1-2): 70−120. [doi: 10.1016/S0019-9958(82)91250-5]
[32] Milner R. A complete axiomatisation for observational congruence of finite-state behaviours. Information and Computation, 1989, 81(2):227−247. [doi: 10.1016/0890-5401(89)90070-9]
[33] Basten T, Voorhoeve M. An algebraic semantics for hierarchical P/T nets. LNCS 935, 1995. 45−65. [doi: 10.1007/3-540-60029-9_33]

**附中文参考文献**:

[1] 杨芙清.软件工程技术发展思索.软件学报,2005,16(1):1−7. http://www.jos.org.cn/1000-9825/16/1.htm
[2] 李明树,杨秋松,翟健.软件过程建模方法研究.软件学报,2009,20(3):524−545. http://www.jos.org.cn/1000-9825/3432.htm [doi: 10.3724/SP.J.1001.2009.03432]
[7] 费立蜀,顾庆,陈道蓄.一种过程定义模型及其验证性分析.计算机科学,2004,31(1):145−151.
[8] 胡旷,董广智,田勇,柳军飞.一种弹性的软件过程模型验证工具.计算机工程与设计,2007,28(7):1497−1500.
[9] 柳军飞,唐稚松.软件过程建模语言研究.软件学报,1996,7(8):449−457. http://www.jos.org.cn/1000-9825/19960801.htm
[23] 陆汝黔.计算机语言的形式语义.北京:科学出版社,1992.

**代飞**(1982－),男,四川乐山人,博士,讲师,CCF 会员,主要研究领域为软件过程方法与技术,软件工程.

**李彤**(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件过程方法与技术,软件工程.

**谢仲文**(1982－),男,博士生,CCF 学生会员,主要研究领域为软件过程.

**于倩**(1975－),女,讲师,CCF 会员,主要研究领域为软件过程.

**卢萍**(1984－),女,硕士,主要研究领域为软件工程.

**郁涌**(1980－),男,博士,讲师,主要研究领域为信息安全,软件工程.

**赵娜**(1982－),女,博士,讲师,CCF 学生会员,主要研究领域为软件工程.