# 复杂分支和同步模式的Pi演算描述

郭小群 郝克刚 侯 红 丁剑洁
(西北大学信息科学与技术学院 西安 710069)

**摘 要** Pi演算是一种描述和分析动态并发系统的计算模型。对Pi演算进行研究后,提出了以Pi演算作为工作流形式化的基础,并利用Pi演算对复杂分支和同步工作流模式进行了详细的描述。

**关键词** Pi演算,工作流模式,形式化方法
**中图法分类号** TP311.52 **文献标识码** A

## Formalizing Advanced Branching and Synchronization Patterns Using Pi Calculus

GUO Xiao-qun HAO Ke-gang HOU Hong DING Jian-jie
(School of Information Science and Technology, Northwestern University, Xi'an 710069, China)

**Abstract** Pi calculus is a computer model which can be used to model concurrent and dynamic systems. The Pi calculus was proposed as a formal foundation for workflow after researching Pi calculus, further more advanced branching and synchronization patterns workflow patterns were described by using the Pi calculus in detail.

**Keywords** Pi calculus, Workflow patterns, Formal method

Aalst等人通过对多种当时主流的WFMS所能支持的控制流结构进行研究、对比和分析,归纳整理出了6类20种典型的控制流结构,并把它们上升到模式的高度加以抽象和描述[1]。在此基础上,通过不断的发展完善控制流模式,又补充了23个新的控制流模式[2]。为了使这些模式的语义精确化,并使它们在各种不同的工作流系统中的实现一致,必须对其提供一种形式化的描述方法。但Pi演算和Petri网谁更适合作为工作流的基础,引发了一场争论[3]。文献[4]中,用Pi演算对20种经典的控制流模式作了描述。本文主要研究用Pi演算描述新提出的复杂分支和同步模式。

## 1 Pi演算简介

Pi演算[5,6]起源于1991年,是由图灵奖得主Robin Milner参照物理学大统一理论提出的。它是一种描述和分析并发系统的计算模型,用动态演化结构表示过程间的间歇性的相互作用,名称是基本的概念,值、变量和管道通过名称引用。

**定义1** Pi演算过程如下:

$P::=M|P|P'|\nu zP|!P$

$M::=0|\pi.P|M+M'$

1)0表示这个过程不做任何动作。

2)$\pi.P$表示当前面的动作执行完后,才开始执行过程$P$;$\bar{x}y.P$表示名称$x$发送名称$y$后,开始执行$P$;$x(z).P$表示名称$x$收到任何名称后,执行$P$并用收到的名称替换$z$;$\tau.P$表示从$\tau.P$不可见的演化到$P$,即可认为$\tau$是一个内部动作。

3)$M+M'$表示$M$和$M'$只能有一个执行。

4)$P|P'$表示$P$和$P'$独立执行并且可通过共有的名称相互通信。

5)$\nu zP$表示名称$z$的作用范围是$P$。

6)$!P$表示可无限重复执行$P$,即认为$!P=P|P|P\cdots$或$!P=P|!P$。

**定义2** 优先原语指的是下面加了下划线的原语,在规约中让优先原语先规约。

例如:$x\langle y\rangle.P|x\langle u\rangle.Q|\underline{\bar{x}(z).R}\rightarrow x\langle y\rangle.P|Q[z/u]|R$而不能$x\langle y\rangle.P|x\langle u\rangle.Q|\underline{\bar{x}(z).R}\rightarrow P[z/y]|x\langle u\rangle.Q|R$。

## 2 复杂分支和同步模式

WP28 阻塞鉴别器(Blocking Discriminator):在工作流过程中的一点,两个或多个分支合并到一个分支,这些分支在过程模型中早先经过一次或多次分流。当第一个输入分支被激活时,控制就被传递给后续分支。当所有的输入分支对于同一过程实例被激活一次后,阻塞鉴别器复位。在阻塞鉴别器复位前,输入分支的后继激活被阻塞。如图1所示,A,B,C代表3个不同的输入分支,而且这3个分支间为或关系。在以下各图中,+表示或,⊕表示互斥,*表示与,弧线外的符号表示所有分支间的关系,弧线内的符号表示相邻分支间的关系。

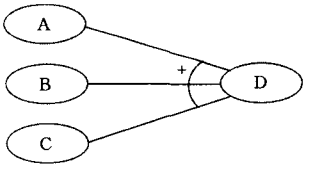


图1 阻塞鉴别器

到稿日期:2009-07-08 返修日期:2009-11-10 本文受国家高可信研究开发计划(2007AA010305)资助。

郭小群 女,讲师,主要研究方向为软件工程,E-mail:guoxiaoq@nwu.edu.cn;郝克刚 男,博士生导师,主要研究方向为软件工程;侯 红 女,副教授,主要研究方向为软件度量;丁剑洁 女,博士生,讲师,主要研究方向为软件度量。

在用 Pi 演算表示各种不同模式时，活动分支用过程表示，活动间的关系用管道表示。由于图形只是为了理解公式的一个示意图，因此公式中的活动分支和通道并没有全部在图上表示出来，例如 $D_0$，$D_1$，$\bar{h}$ 等，后面各图以及公式之间的关系亦如此。

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$C=!\ r_c.\bar{c}.0$

$D_0=(\nu h)(D_1|D_2)$

$D_1=a.\bar{h}.0|b.\bar{h}.0|c.\bar{h}.0$

$D_1=h.\bar{r}_d.h.h.D_0$

$D=!\ r_d.\sigma_D.D'$

WP28$=A|B|C|D_0|D$

公式里的 $\sigma_D$ 表示在节点 $D$ 执行的动作，后续各公式中出现的 $\sigma$ 的含义与此相同。

WP29 取消鉴别器(Canceling Discriminator)：在工作流过程中的一点，两个或多个分支合并到一个分支，这些分支在过程模型中早先经过一次或多次分流。当所有输入分支中的第一个完成后，控制就被传递给后续分支。激发鉴别器也会取消其他的输入分支的执行并使鉴别器复位。如图 2 所示，当 $A$，$B$，$D$ 3 个分支中的第一个完成后，就开始 $C$。

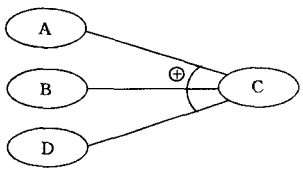


图 2　取消鉴别器

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$D=!\ r_d.\bar{d}.0$

$C_0=(\nu h)(C_1|!\ h.C_1)$

$C_1=a.\bar{r}_c.\bar{h}.(\underline{b}.\underline{\underline{d}}.0+\underline{d}.\underline{b}.0)+b.\bar{r}_c.\bar{h}.(\underline{a}.\underline{\underline{d}}.0+\underline{d}.\underline{\underline{a}}.0)+d.\bar{r}_c.\bar{h}.(\underline{a}.\underline{b}.0+\underline{b}.\underline{\underline{a}}.0)$

$C=!\ r_c.\sigma_C.C'$

WP29$=A|B|D|C|C_0$

其中，$\underline{x}$通信的优先级最高，$\underline{x}$次之，$x$ 最低。

WP30 结构化的部分合成(Structured Partial Join)：$m$ 个分支合并到一个分支上，当输入分支的 $n$ 个激发时，这里 $n$ 小于 $m$，控制线程被传递给后续分支，其余分支的激发不会导致线程的向后传递，当所有活动分支被激发后，合成节点复位。如图 3 所示，$A$，$B$，$D$ 3 个分支中的两个完成，就开始 $C$。

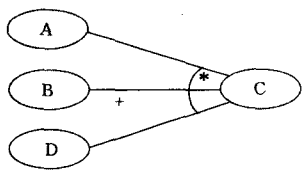


图 3　结构化的部分合成

$A=r_a.\bar{h}.A$

$B=r_b.\bar{h}.B$

$D=r_d.\bar{h}.D$

$C_0=h.h.\bar{r}_c.h.C_0$

$C=!\ r_c.\sigma_c.C$

更一般的描述为：$A_i=r_{ai}.\bar{h}.A_i\quad i=1,\cdots,m$

$C_0=h.\cdots.h.\bar{r}_c.h.\cdots.h.C_0$(在 $\bar{r}_c$ 前有 $n$ 个 $h.$，在 $\bar{r}_c$ 后有 $m-n$ 个 $h$)

$C=r_c.\sigma_c.C$

WP30$=A_1|\cdots|A_m|C_0|C$

WP31 阻塞部分合成(Blocking Partial Join)：$m$ 个分支合并到一个分支上，当输入分支的 $n$ 个激发时，这里 $n$ 小于 $m$，控制线程被传递给后续分支，当同一个实例的所有活动分支都被激发后，合成节点复位。直到该合成节点复位完成后，剩余的分支才能被激发。如图 4 所示，当 3 个输入分支中有两个被激发后，则 $D$ 开始。

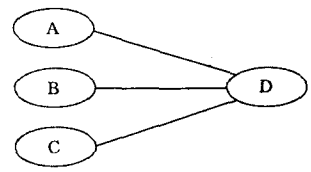


图 4　阻塞部分合成

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$C=!\ r_c.\bar{c}.0$

$D=(\nu h)(D_1|D_2)$

$D_1=a.\bar{h}.0|b.\bar{h}.0|c.\bar{h}.0$

$D_2=h.h.\bar{r}_d.h.D_0$

$D=!\ r_d.\sigma_d.0$

更一般的描述为：$A_i=!\ r_{ai}.\bar{a}_i.0i=1,\cdots,m$

$D_0=(\nu h)(D_1|D_2)$

$D_1=a_1.\bar{h}.0|\cdots|a_m.\bar{h}.0$

$D_2=h.\cdots.h.\bar{r}_d.h.\cdots.h.D_0$(在 $\bar{r}_d$ 前有 $n$ 个 $h.$，在 $\bar{r}_d$ 后有 $m-n$ 个 $h$)

$D=!\ r_d.\sigma_d.0$

WP31$=A_1|\cdots|A_m|D_0|D$

WP32 取消部分合成(Canceling Partial Join)：把 $m$ 个分支合并到一个分支上，当输入分支中有 $n$ 个被激发时，这里 $n$ 小于 $m$，控制线程就被传递给后续分支。激发合成活动会取消所有其它的输入分支的执行并进行合成节点的复位。如图 5 所示，在 $A$，$B$，$D$ 3 个分支中有两个完成后，就开始 $C$，而 $D$ 将被取消。

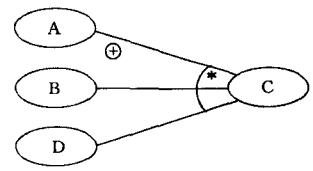


图 5　取消部分合成

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$D=!\ r_d.\bar{d}.0$

$C_0=(\nu h)(C_1|!\ h.C_1)$

$C_1=a.(\underline{b}.\bar{r}_c.\bar{h}.\underline{\underline{d}}.0+\underline{d}.\bar{r}_c.\bar{h}.\underline{\underline{b}}.0)+b.(\underline{a}.\bar{r}_c.\bar{h}.\underline{\underline{d}}.0+\underline{d}.\bar{r}_c.\bar{h}.\underline{\underline{a}}.0)+d.(\underline{a}.\bar{r}_c.\bar{h}.\underline{\underline{b}}.0+\underline{b}.\bar{r}_c.\bar{h}.\underline{\underline{a}}.0)$

$C=!\ R_c.\sigma_c.0$

WP32$=A|B|D|C_0|C$

(下转第 179 页)

[8] Knowles J D, Corne D W. The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimization[C]// Congress on Evolutionary Computation (CEC99). Piscataway, NJ, 1999

[9] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: A comparative case study and strength Pareto approach[J]. IEEE Transactions on Evolutionary Computation, 1999, 3: 257-271

[10] Laumanns M, Zitzler E, Thiele L. On the effects of archiving, elitism, and density based selection in evolutionary multi-objective optimization[C]// Evolutionary Multi-Criterion Optimization (EMO 2001). 2001

[11] Zitzler E, Laumanns M, Thiele L. Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization[C]// EUROGEN 2001, Evolutionary Methods for Design, Optimization and Contorl with Applications to Industrial Problems. September 2001

(上接第 140 页)

WP33 一般合并结构(Generalized AND-Join):两个或多个分支被合并到后继分支,当所有分支都被激发后,则控制线程被传递到后继分支。在合并期间,各个分支仍可接受激发(每个分支接受的激发数目要相同),并保存这些激发以将来使用。

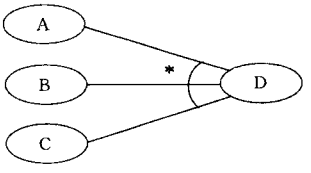


图 6 一般合并结构

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$C=!\ r_c.\bar{c}.0$

$D_0=(\nu h)(D_1|D_2)$

$D_1=a.\bar{h}.0|b.\bar{h}.0|c.\bar{h}.0$

$D_2=h.h.h.\bar{r}_d.D_0$

$D=!\ r_d.\sigma_d.D'$

$WP33=A|B|C|D_0|D$

WP37 无循环同步合并(Acyclic Synchronizing Merge):早期被分解的两个或多个分支合并到一个后继分支上,当每一个活动的输入分支被激活时,控制线程都会被传递给后续分支。至于多少个分支需要同步则取决于合并结构能够获得的信息。如图 7 所示,$A,B,D$ 3 个分支是从同一个分支分解出来的且都处于活动状态,当它们中的任意一个完成时,都将会引起 $C$ 的执行,而且 $C$ 中还需要 $B,D$ 同步。

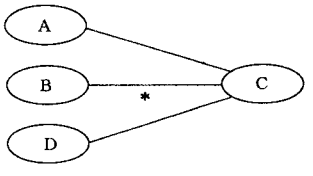


图 7 无循环同步合并

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$D=!\ r_d.\bar{d}.0$

$C=(\nu h)(C_1|C_2)$

$C_1=b.\bar{h}.0|c.\bar{h}.0$

$C_2=a.\sigma_c.C'|h.h.\sigma_c.C'$

$WP37=A|B|C|D$

WP38 通用的同步合并(General Synchronizing Merge):早期被分解的两个或多个分支合并到一个后继分支上,当满足下面两个条件之一时:(1)每一个活动的输入分支被激活时,(2)未被激活的分支在将来也不可能被激活,则控制线程被传递给后续分支。如图 8 所示,$A,B,D$ 是从同一个分支分解出来的,这里 $A,B$ 是两个活动的输入分支,$D$ 代表不被激活的分支。

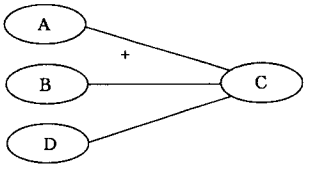


图 8 通用同步合并

$A=!\ r_a.\bar{a}.0$

$B=!\ r_b.\bar{b}.0$

$D=\bar{d}.0$

$C=a.\sigma_c.C'|b.\sigma_c.C'|d.\sigma_c.C$

$WP38=A|B|C|D$

**结束语** 本文利用 Pi 演算对复杂分支和同步工作流模式进行了详细描述,所用描述完全是形式化的,它不但使过程模型的语义更加精确,而且在模型建立后,还可利用 Pi 演算来推演系统的行为,验证模型的正确性,譬如发现系统行为不完整、死锁或缺少同步等。

## 参 考 文 献

[1] Aalst W M P, Barros A P, Hofstede A H M, et al. Advanced workflow patterns[C]// Proceedings of the Fifth IFCIS International Conference on Cooperative Information Systems (CoopIS' 2000). volume 1901 of LNCS. Eilat, Israel, Springer, 2000: 18-26

[2] Russell N. Foundations of Process-Aware Information Systems Faculty of Information Technology, Queensland University of Technology[J]. Brisbane, Australia, 2007

[3] van der Aalst W M P. Pi Calculus Tjersus Petri Nets: Let Us Eat Humble Pie Rather Than Further inflate the Pi Hype[J]. BPTrends, 2005, 3(5): 1-11

[4] Puhlmann F, Weske M. Using the Pi-Calculus for Formalizing Workflow Patterns[C]// BPM 2005. volume 3649 of LNCS. Berlin: Springer-Verlag, 2005: 153-168

[5] Milner R, Parrow J, Walker D J. A calculus of Mobile Process Part Ⅰ[R]. Report ECS-LFCS-89-85. Laboratory for Foundation of Computer Science, Computer Science Department, Edinburgh University, 1989

[6] Milner R, Parrow J, Walker D J. A calculus of Mobile Process Part Ⅱ[R]. Report ECS-LFCS-89-85. Laboratory for Foundation of Computer Science, Computer Science Department, Edinburgh University, 1989