国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第18卷第1期 1996年3月 Vol. 18 No. 1

并发面向对象语言的演算语义研究*

李 侃 王兵山 李舟军
(国防科技大学计算机系 长沙 410073)

摘 要 本文对π-演算进行了扩展，使之能支持异步通讯，然后在此基础上给出了一种简单 Actor 语言的演算语义。该演算语义能较好地刻划异步通讯机制、演员的行为替换机制以及对象、类、封装、实例变量与临时变量等面向对象特征。

关键词 并发面向对象语言，π-演算，异步通讯
分类号 TP312

On Calculus Semantics of Concurrent Object-Oriented Language

Li Kan Wang Bingshan Li Zhoujun
(Department of Computer, NUDT, Changsha, 410073)

Abstract In this paper π-calculus is extended in order to support asynchronous communication, then the calculus semantics of a simple Actor language is given. This calculus semantics can express asynchronous communication, the actor's behaviour replacement and some important object-oriented features such as object, class, encapsulation, instance variable, temporary variable etc.

Key words concurrent object-oriented language, π-calculus, asynchronous communication

随着现代科学对计算要求的日益提高，并发性在计算机各个领域普遍受到重视。面向对象程序设计语言为程序员提供了一个较高的抽象层次。对象是一个自含的行为实体，对象之间通过互发消息共同协作完成计算任务，因而将并发机制引入面向对象语言中是一件极自然的事。国外关于COOL(concurrent object-oriented language)的研究经验表明：将并发机制与对象自身的各种语言特征融合起来并不是一件容易的事。很多 COOL 语言在设计甚至是实现完毕之后才发现存在着这样或那样问题。造成这种情况的一个重要原

* 国防科技预研基金和“863”计划资助课题
1995年11月19日收稿

因是大多数并发的面向对象语言缺乏严格的语义基础，使得我们不能确切地把握、衡量面向对象语言的各种特征，不能对语言中的一些基本性质进行推导。

关于COOL的经典语义研究有：P. America基于转换图给出了POOL语言的操作语义，基于度量空间（metric space）给出了POOL语言的指称语义。

用演算来刻划COOL语言的语义是另外一种有价值的间接语义描述方法。其优点在于：语义描述清晰，较容易理解。另外它还可以作为一种规范描述语言，用LISP或PROLOG语言可以较方便地实现一个基于该规范描述语言的证明器。Actor模型是一种面向对象的并发计算系统模型。较之POOL语言，基于Actor模型的Actor语言是一种更灵活更典型的COOL语言。下面试图给出一种简单Actor语言的演算语义。

1 演员模型与一个简单的演员语言SAL

演员模型的基本元素是演员和消息。演员（actor）是活动的对象，它的职能是处理发给它的各类消息，每个演员都是整个计算系统中一个自含的相互独立的实体，演员之间只能通过互发消息来影响对方的行为。演员之间的通讯是异步的。

可用一个二元组（通讯地址，当前行为模式）来表征某一时刻的演员。演员的通讯地址体现了演员作为一个独立实体而存在。一个演员只有知道另一个演员的通讯地址才可向它发消息；一个演员系统必须有完善的命名机制来保证各个演员的通讯地址互不相同。另外，在演员系统中，演员的通讯地址可像其它值一样通过消息被传递，这使得演员系统具有重构能力。一个演员在处理一条发给自己的消息时，将根据自己的当前行为模式决定自己所可能采取的动作。演员动作中有三条核心原语：一是向其它演员发消息；二是创建新演员；三是行为替换。行为替换使得演员具有历史敏感性。这是可变的数据对象所必须具备的。这使得演员系统克服了函数式程序语言的不足，不过，它又不同于顺序程序语言中的局部状态变换。关于演员模型内容繁杂，感兴趣者可参阅文[2]。

下面引入一个小型语言SAL，它包含演员模型中三个最核心的原语。

SAL的文法如下：

程序定义：ProgramDecl::=TemplateDecl$_1$，…，TemplateDecl$_n$ {VarDecl S }

模板定义：TemplateDecl::=Template C VarDecl MethodDecl$_1$…MethodDecl$_s$

方法定义：MethodDecl::=Accept msg (z_1，…，z_i) {VarDecl S }

变量说明：VarDecl::=Var x_1，…，x_n

语句定义：

S::=S_1；S_2	顺序语句
| if E then S_1 else S_2	条件语句
| while E do S	While语句
| skip	异常终止
| x:=E	赋值语句
| send msg (E_1，…，E_i) to x	异步消息调用语句
| become C (E_1，…，E_n)	行为替换语句

表达式定义：

E::=x	一般变量
| Self	正在执行的对象

| n	一个整数常量
| b	一个布尔常量
| nil	空对象
| Create C (E_1, …, E_n)	按模板 C 创建的新演员
| E_1+E_2	算术表达式
| $E_1=E_2$	整数比较
| $E_1 \Longrightarrow E_2$	对象比较

其中 Template, Accept, Var, if, then, else, while, do, skip, send, to, become, create, nil 为 SAL 中的关键字。msg 为消息选择符。x, z 为变量。行为替换语句的作用有两个：一是改变演员中实例变量的值。在 SAL 中，只有行为替换语句能改变实例变量的值。二是完全改变演员的行为模板，即按语句中指定的模板 C 行动。它不同于“创建新演员”之处在于：它只改变演员的行为模式，但其通讯地址保持不变，即“旧瓶装新酒”。

2 一种扩展的 π-演算

π-演算是一种基于通讯的进程演算系统。在 π-演算中，一个计算系统被认为是一群相互独立的 agent 的集合，agent 之间通过相互共享的通道名进行通讯。π-演算特别适于模拟具有动态通讯结构的计算系统。因而选择 π-演算来描述 SAL 的语义是恰当的。但由于 π-演算不能表达异步通讯机制，不支持消息的并发接收，因而有必要对 π-演算进行扩展。

令 x，y，z 指代名字，A 指代 Agent 标识符，P、Q、R 指代 Agent.

$$P=\sum_{i\in I}P_i \mid \alpha.P \mid P|Q \mid (y)P \mid [x=y]P \mid A(\tilde{y})$$

这里，I 是指标集，$\alpha\in\{x(y),\bar{x}y,\tau,\leftarrow\bar{x}y,(x_1(y_1)|\cdots|x_n(y_n))\}$，$\tilde{y}=y_1,\cdots,y_n$(n 是 Agent A 的元(arity))，x(y).P，(y)P 中的 y 是受限的，其活动域为 P。

- $\sum_{i\in T}P_i$ 指只执行某个 $P_i(i\in I)$.
- $\bar{x}y$.P 指通过通道 x 输出 y，然后按 P 行动。x(y).P 指从通道 x 中接收一个名字 z，然后按 P{z/y}行动，即用 z 取代 P 中所有自由出现的 y。τ.P 指的是执行一个无声的内部动作，然后按 P 行动。$\leftarrow\bar{x}y$.P 和 $(x_1(y_1)|\cdots|x_n(y_n))$.P 属扩展部分（其解释稍见后），其作用在 SAL 的语义描述中体现得十分清楚。
- P|Q 指的是 P、Q 并发执行，通过通讯同步。
- (y) P 指的是：y 是 P 的私有通道，任何 Agent 不可以通过 y 与 P 发生联系，然而 P 中的子 Agent 可以通过 y 互相通讯。
- [x=y]P 指：如果 x=y，则按 P 行动，否则不行动。
- 每个 Agent 标识符与一个定义式相联系。若有定义式 $A(\tilde{x})=P$，则 $A(\tilde{y})$按 $P\{\tilde{y}/\tilde{x}\}$ 行动。

Agents 的行为通过 Agents 之间的“$\xrightarrow{\alpha}$”关系来描述，这里

$\alpha\in\{x(y),\bar{x}y,\bar{x}(y),\tau,\leftarrow\bar{x}y,(x_1(y_1)|\cdots|x_n(y_n))\}$

其中 $\bar{x}(y)$是(y)$\bar{x}y$ 的简写形式，指的是沿着通道传出一个私有名 y，而$\leftarrow\bar{x}y,(x_1(y_1)|\cdots|x_n(y_n))$是本文关于 π-演算的扩展部分，$\leftarrow xy$ 是为了支持异步通讯，$(x_1(y_1)|\cdots|x_n$

$(y_n))$是为了支持消息中参数的并发计算。其推演规则为：

(1) $\leftarrow\bar{x}y.P \xrightarrow{\tau} \bar{x}y|P$

(2) $(x_1(y_1)|\cdots|x_n(y_n)).P \xrightarrow{x_i(w_i)} (x_1(y_1)|\cdots|x_{i-1}(y_{i-1}|x_{i+1}(y_{i+1}|\cdots|x_n(y_n)).P\{w_i/y_i\}$

其中 $w_i \notin fn((y_i)P)$，即 w_i 不是$(y_i)P$ 中的自由变量。

为描述方便，还需预定义若干个常用进程：

·原子传递进程

$\bar{x}u,v.P|x(m,n).Q = (w)(\bar{x}w.\bar{w}u.\bar{w}v.P|x(w).w(m).w(n).Q$

$\cdot\alpha!P = \alpha.(P|\alpha!P)$

$\cdot x:[y_1 \Rightarrow P_1, y_2 \Rightarrow P_2, \cdots] = x(z).([z=y_1]P_1 + [z=y_2]P_2 + \cdots)$

3 SAL 的语义

下面用扩张后的演算给出 SAL 的语义。在描述时，充分利用了演算的约束名作为参数传递的特性，使得描述简洁有力。首先，看一下模板的语义描述。

记模板的语义描述物

$$[\text{Template C Var } x_1,\cdots,x_n \text{ MethodDecl}_1 \cdots \text{MethodDecl}_s]$$

为：TemplateC，则

TemplateC

$$= (x_1,\cdots,x_n,\text{DEAD})(\text{CREATE}_c(v_1,\cdots,v_n).\text{NEWID}(w).\overline{\text{OUTPUT}}_c w.$$
$$(((\text{Var}_{x_1}(v_1)|\cdots|\text{Var}_{x_n}(v_n)) + \text{DEAD})|\text{Scheduler}(w,c,x_1,\cdots,x_n)))|$$
$$\text{TemplateC} + (x_1,\cdots,x_n,\text{DEAD})(\text{BECOME}_c(v_1,\cdots,v_n).\text{RECEIVE}_c(w).$$
$$(((\text{Var}_{x_1}(v_1)|\cdots|\text{Var}_{x_n}(v_n)) + \text{DEAD})|\text{Scheduler}(w,c,x_1,\cdots,x_n)))|\text{TemplateC}$$

模板定义可看做是一个用递归方式定义的 agent。它能接受两类请求：一是创建新演员。此刻它要从通道 NEWID 中获得一个分配给该演员的唯一的标识符，然后建立实例变量和消息处理 Schduler. 另外一类请求是行为替换请求。从上式可以看出，行为替换引起的动作，在相当程度上类似于创建新演员引起的动作，只不过在演员标识符获取方式上有所不同。DEAD 主要用于控制实例变量的生存周期。当一个演员发生行为变换时，将统一建立代表新实例变量的 agent。而让代表旧实例变量的 agent 终止生存。

实例变量 agent：$\text{Var}_{x_1}(t_1)|\text{Var}_{x_2}(t_2)|\cdots|\text{Var}_{x_n}(t_n) + \text{DEAD}$，消息处理器 Scheduler，再加上演员的标识符，就完全地表示了该演员在某个时刻的状态。其中 $\text{Var}_x(t_i) = x(t^*).\text{Var}_x(t^*) + \bar{x}t.\text{Var}_x(t)$

消息处理器 Scheduler$(w,c,x_1,\cdots,x_n)$实际上反映了演员的行为(behaviour)，即能处理哪些消息，如何处理。其定义稍见后。

直接将变量名作为通道，x_1，…，x_n 被说明为约束名，其约束范围说明了其作用域与被访问范围。这体现了演员封装性 (Encapsulation)。

下面再看一看 ProgramDecl 的语义：

$$\{\text{TemplateDecl}_1,\cdots,\text{TemplateDecl}_n.\{\text{Var } x_1:t_1,\cdots,x_n:t_n \quad S\}\}$$

$$= \llbracket \text{TemplateDecl}_1 \rrbracket | \cdots | \llbracket \text{TemplateDecl}_n \rrbracket | \text{Idgenerater}_0 |$$
$$(\widetilde{\text{init}})(\text{Var}_{x_1}(\text{nil}) | \cdots | \text{Var}_{x_n}(\text{nil}) | \llbracket S \rrbracket (\widetilde{\text{init}}))$$

程序初启时，假设存在一个系统自动创立的演员。代表该演员的 agent 就是

$$(\widetilde{\text{init}})(\text{Var}_{x_1}(\text{nil}) | \cdots | \text{Var}_{x_n}(\text{nil}) | \llbracket S \rrbracket (\widetilde{\text{init}}))$$

Idgenerator 可看作一个系统调用。它用于产生唯一的标识符。一种定义方法是：

$$\text{Idgenerator}_n = \overline{\text{NEWIDACTOR}_n}.\ \text{Idgenerator}_{n+1}$$

接着我们就来看 Scheduler 的定义：

$$\text{Scheduler}(w, c, x_1, \cdots, x_n)$$
$$= w(\text{msgselector}, \text{PARCH}).$$
$$([\text{msgselector} = \text{msgselector}_1]\text{METHODBODY}_1(w, c, x_1, \cdots, x_n, \text{PARCH})$$
$$+ [\text{msgselector} = \text{msgselector}_2]\text{METHODBODY}_2(w, c, x_1, \cdots, x_n, \text{PARCH})$$
$$+ \cdots\cdots$$
$$+ [\text{msgselector} = \text{msgselectors}]\text{METHODBODY}_n(w, c, x_1, \cdots, x_n, \text{PARCH}))$$

其中

$$\text{METHODBODY}_i(w, c, x_1, \cdots, x_n, \text{PARCH})$$
$$= \llbracket \text{Accept msg}_i(y_1, \cdots, y_m)\{\text{Var } z_1, \cdots, z_i \quad S\} \rrbracket (w, c, x_1, \cdots, x_n, \text{PARCH})$$
$$= (y_1, \cdots, y_m, z_1, \cdots, z_i, \text{TERMINATE})(\text{PARCH}(v_1, \cdots, v_m).$$
$$((\text{Var}_{y_1}(v_1) | \cdots | \text{Var}_{y_m}(v_m) | \text{Var}_{z_1}(\text{nil}) | \cdots | \text{Var}_{z_i}(\text{nil})) + \text{TERMINATE})$$
$$| \llbracket S \rrbracket (w, c, x_1, \cdots, x_n))$$

请注意，PARCH 是参数通道。该通道是约束名。只传送与它相关的 msgselector 的参数。这一点在 SEND 的语义描述中可以看得更清楚。TERMINATE 主要用于控制临时变量的生存周期。一个方法体执行结束，即执行 skip 时，就通过通道 TERMINATE 发出临时变量消亡的信号。实例变量不同于临时变量。这主要体现在其生存周期，作用范围以及修改手段上，它们在本语义中都得到了体现。

下面给出语句的语义描述：

$$\llbracket S_1; S_2 \rrbracket (w, c, x_1, \cdots, x_n)$$
$$= \llbracket S_1 \rrbracket (w, c, x_1, \cdots, x_n) \text{before} \llbracket S_2 \rrbracket (w, c, x_1, \cdots, x_n)$$

其中　p before q = (DONE)(P | DONE. Q)

规定每个语句执行结束后，其最后一个动作是通过 DONE 输出一个信号。通过 DONE 同步，能确保 P 先执行 Q 后执行。

$$\llbracket \text{while E do S} \rrbracket (w, c, x_1, \cdots, x_n)$$
$$= (\text{CHANNEL})(\llbracket E \rrbracket (w, \text{CHANNEL}) | \text{CHANNEL}(\text{val}).$$
$$[\text{val} = \text{TRUE}] \llbracket S \rrbracket (w, c, x_1, \cdots, x_n) \text{before} \llbracket \text{while E do S} \rrbracket (w, c, x_1, \cdots, x_n))$$
$$\llbracket \text{if E then } S_1 \text{ else } S_2 \rrbracket (w, c, x_1, \cdots, x_n)$$
$$= (\text{CHANNEL})(E(w, \text{CHANNEL}) | \text{CHANNEL}(\text{val}).$$
$$([\text{val} = \text{TRUE}] \llbracket S_1 \rrbracket (w, c, x_1, \cdots, x_n) + [\text{val} = \text{FALSE}] \llbracket S_2 \rrbracket (w, c, x_1, \cdots, x_n)))$$
$$\llbracket \text{send msg}(E_1, E_2, \cdots, E_m) \text{to X} \rrbracket (w, c, x_1, \cdots, x_n)$$

$=(PARCN,CHANNEL_1,\cdots,CHANNEL_n,CHANNEL)$

$(\llbracket E_1\rrbracket(w,CHANNEL_1)|\cdots|\llbracket E_m\rrbracket(w,CHANNEL_m)$

$|\llbracket X\rrbracket(w,CHANNEL)|(CHANNEL_1(x_1)|\cdots|CHANNEL_m(x_m)|CHANNEL(x)).$

$\leftarrow\bar{x}msg,PARCH.\leftarrow\overline{PARCH}$

$x_1,\cdots,x_n,\overline{DONE})$

上述过程有两点值得注意：(1) 它体现了参数的并发计算；(2) 利用 π 演算的扩展部分，体现了异步消息传递。事实上，这里描述的通讯机制保证消息不会丢失，但不能保证公平性，不能保证消息接收次序一致于发送次序。

$\llbracket become\ C\ (E_1,E_2,\cdots,E_p)\rrbracket(w,c,x_1,\cdots,x_n)$

$=(CHANNEL_1,\cdots,CHANNEL_p)(\llbracket E_1\rrbracket(w,CHANNEL_1)|\cdots|$

$\llbracket E_p\rrbracket(w,CHANNEL_p)|(CHANNEL_1(v_1)|\cdots|CHANNEL_p(v_p)).$

$\overline{BECOME}_c v_1,\cdots,v_p.\overline{DEAD}.\overline{RECEIVE}_c w.\overline{DONE})$

赋值语句的处理涉及到表达式的处理。关于表达式的处理有以下两种观点：

(1)完全遵从演算的推导规则，用解码的方式表达整数及其运算。关于这方面可参考文[3]。但必须对它作一些补充[5]。

(2)采用实用主义的方法。将布尔量、整数看作常名，其上的运算是基本原语，由 DEMON 自动完成，而无需按照演算规则推导来模拟基本运算。这就象 λ-演算是函数式语言的基础，但是实际函数式语言关于算术等基本运算的实现，并不用编码的方式进行。将来基于该 π-扩展演算实现一个证明器时，肯定要走这样一条实用主义路线[5]。

3 结束语

本文基于扩展的 π-演算，较好地描述了 SAL 的异步通讯机制、演员的行为替换机制以及类、对象、封装、实例变量，临时变量等面向对象特征。下一步的工作有：对扩展的 π-演算进行更深入的理论研究，建立起一套关于进程的双模拟以及一系列等价关系的理论；实现一基于该演算的定理证明器；基于该演算，更好地研究 SAL 语言的高级特征。

参 考 文 献

1 Milner R, Parrow J, Walker D. A calculus of Mobile Processes, part Ⅰ, Ⅱ. Information and Computation. 1992, 100

2 Agha G. The Structure and Semantics of Actor Languages. Foundations of Object－oriented languages. Spring－Verleg Berlin Heidelberg, 1991

3 David Walker. π-calculus semantics of Object－orlented Languages. In Proceedings of the Conference on Theoretical Aspect of Computer Software (Japan). 1991, LNCS--526

4 Kehei Honda, Mario Tokoro. On Asynchronous Calculus. In Prpceedings of the Europen Conferences On OOP. LNCS－512

5 李侃. Actor 语言的语义研究. [硕士论文], 长沙：国防科技大学, 1996

6 陆汝钤，计算机语言的形式语义，北京：科学出版社，1992

7 王飓安，高阶进程演算及其应用 [博士论文]，北京航空航天大学

(责任编辑　潘　生)