第7卷第4期　　软 件 学 报　　Vol.7, No.4
1996年4月　　JOURNAL OF SOFTWARE　　Apr. 1996

一个面向对象的实时分布式语言的指称语义*

左志宏　　龚天富

（电子科技大学计算机系　成都　610054）

摘要　本文给出了一个面向对象的实时分布式语言的指称语义，在不同层次上给出了语句、对象和程序的清晰描述.提出了实时状态的概念.借助于它，在指称语义的框架内，简洁地刻画了语言的实时特性.

关键词　对象，实时，分布式，并行，指称语义.

Mini CSP－R[1]是由CSP[2]发展起来的一种抽象的面向对象的实时分布式语言.它能简单地扩展成CSP－R[1]，并能模拟Ada语言的实时特性.文献[1]给出Mini CSP-R的复合指称语义，但不支持面向对象.采用文献[3,4]给出的办法，本文首次给出Mini CSP－R的完整的指称语义.它以分层的方式清晰地给出了语句、对象和程序的语义，同时以一种简明的方式刻画了语言的实时特性.该文的结果已在“八五”预研项目“分布式实时语言”中得到应用.

1　Mini CSP－R语言

用 *Stat* 表示Mini CSP－R语言的语句的集合.语句 $s \in Stat$ 如下定义：

$$
\begin{aligned}
s ::= {} & x := e \\
& \mid O!\ e \mid *!\ e \\
& \mid O?\ x \mid *?\ x \\
& \mid wait\ d \\
& \mid s_1;\ s_2 \\
& \mid \mathop{\Box}_{j=1}^{n} g_j \rightarrow s_j \\
& \mid *[\mathop{\Box}_{j=1}^{n} g_j \rightarrow s_j]
\end{aligned}
$$

其中 e 是表达式，d 是整型表达式，x 是变量，g_j 是如下的卫哨：

$$
\begin{aligned}
g ::= {} & b \\
& \mid a \\
& \mid b;a \\
& \mid wait\ d
\end{aligned}
$$

---

* 作者**左志宏**，1966年生，讲师，主要研究领域为语言及支撑环境.**龚天富**，1938年生，教授，主要研究领域为语言及支撑环境.

本文通讯联系人：左志宏，成都610054，电子科技大学计算机系

本文1995－01－29收到修改稿

$|b;wait\ d$

其中 b 是布尔表达式，α 是纯 I/O 指令，即 $O!\,e$，$O?\,x$，$*!\,e$，$*?\,x$. 对于 b,e,d，不指定它的语法. 用 $Var,GExp,BExp,GInst$ 分别表示变量的集合、一般表达式的集合、布尔表达式的集合和卫哨指令的集合.

程序是对象的一个序列，$P::=\langle O_1::s_1\parallel\cdots\parallel O_n::s_n\rangle,n\geqslant 1$. 它表示对象 $O_i(i=1,\cdots,n)$ 的并行. 用 $ONam$ 表示对象名的集合，有 $ONam^+=ONam\cup\{*\}$. 程序的集合用 $Prog$ 表示.

除了 $wait\ d$ 和 $\overset{n}{\underset{j=1}{\Box}}g_j\rightarrow s_j$ 以外，Mini CSP－R 语言和 CSP 完全一样. 延时指令 $wait\ d$ 将程序挂起 d 个单位时间. 卫哨语句 $\overset{n}{\underset{j=1}{\Box}}g_j\rightarrow s_j$ 的执行如下：首先，如果没有一个卫哨为真（对于延时指令和纯 I/O 指令，总假定为真），则语句终止. 如果至少有一个纯布尔卫哨为真，则任意选择其中一个执行. 如果没有纯布尔卫哨为真，但至少有一个卫哨为真，它的执行如下：如果没有 $wait$ 指令，$waitvalue$ 的值定义为无穷大，否则定义为 $wait$ 指令中的最小值（如果为 0，则 $waitvalue$ 等于 1）. 在 $waitvalue$ 个单位时间中，如果某个为真的通讯发生，则选择它（如果有多个，则任选一个）. 如果在 $waitvalue$ 个单位时间内没有通讯发生，则在具有最短 $waitvalue$ 值的卫哨中选择一个. 在以上的各种情形中，如果 g_j 被选择，则执行 g_j 后，执行相应的 s_j.

为了简洁起见，假定每个语句的执行至少需要一个单位时间，并且赋值语句只需要一个单位时间.

2　Mini CSP－R 中语句的语义

假定每个变量能够存储一个值，值的域记为 Val. 自然数集 $\mathbf{N}\subseteq Val$，$\mathbf{N}^\infty=\mathbf{N}\cup\{\infty\}$. 假定存在一个特殊的元 $nil\in Val$，程序开始执行时，每个变元都有值 nil.

状态的集合 $\Sigma=Var\rightarrow Val$，任意 $\sigma\in\Sigma$ 称为一个状态. 实时状态的集合 $\Sigma_{RT}=\Sigma\times\mathscr{P}(\mathbf{N})$，其中 $\mathscr{P}(\mathbf{N})$ 是 $\mathbf{N}$ 的幂集，$(\sigma,T)\in\Sigma_{RT}$ 称为一个实时状态，一般写为 σ_T，T 表示到达该状态时的时刻的集合.

对于一般表达式和布尔表达式，假定分别有求值函数：

$[\![\ \ ]\!]_G:GExp\rightarrow\Sigma_{RT}\rightarrow Val$

$[\![\ \ ]\!]_B:BExp\rightarrow\Sigma_{RT}\rightarrow\{\mathbf{tt},\mathbf{ff}\}$

对于卫哨，求值函数 $[\![\ \ ]\!]_{Guard}:GInst\rightarrow\Sigma_{RT}\rightarrow\{\mathbf{tt},\mathbf{ff}\}$ 定义如下：

$[\![\,b\,]\!]_{Guard}(\sigma_T)=[\![\,b\,]\!]_B(\sigma_T)$

$[\![\,\alpha\,]\!]_{Guard}(\sigma_T)=\mathbf{tt}$，

$[\![\,b;\alpha\,]\!]_{Guard}(\sigma_T)=[\![\,b\,]\!]_B(\sigma_T)$，

$[\![\,wait\ d\,]\!]_{Guard}(\sigma_T)=\mathbf{tt}$

$[\![\,b;\ wait\ d\,]\!]_{Guard}(\sigma_T)=[\![\,b\,]\!]_B(\sigma_T)$

其中 b 是布尔指令，α 是 I/O 指令. 在不引起混乱的情况下，常省掉下标 $G,B,Guard$，而简写成 $[\![\ \ ]\!]$.

语句的指称域 $SProc$ 是满足下面方程的唯一完备度量空间（见第 5 节）：

$$SProc \cong \{p_0\} \cup (\Sigma_{RT} \times \mathscr{P}(SProc)) \cup (ONam^+ \times Val \times \Sigma_{RT} \times \mathscr{P}(SProc))$$
$$\cup (ONam^+ \times (Val \to \Sigma_{RT}) \times (Val \to \mathscr{P}(SProc)))$$

它的元素称为一个语句进程. p_0 代表终止进程. $[\sigma_T, P]$代表一个内部计算, $\sigma_T \in \Sigma_{RT}$表示完成该计算时的实时状态. $P \in \mathscr{P}(SProc)$表示执行该计算后程序的行为(即该计算后面的程序的语义). $[O, v, \sigma_T, P]$代表消息发送, $O \in ONam^+$是接收对象, $v \in Val$ 是发送的值, σ_T 是完成消息发送时的实时状态. $[O, \varphi, f]$表示消息接收, O 是接收对象. φ和 f 分别是从 Val 到 Σ_{RT}和 $\mathscr{P}(SProc)$的函数. 一般地将 φ 写成 φ_T, φ是从 Val 到 Σ 的函数.

语句的语义函数为 $\mathscr{M}_s: Stat \to Cont \to \Sigma_{RT} \to \mathscr{P}(SProc)$

对于语句 s, $\mathscr{M}_s[\![s]\!]$ 依赖于 2 个变量, 后续(Continuation) $h \in Cont$ 和实时状态 $\sigma_T \in \Sigma_{RT}$, 实时状态 σ_T 表示语句 s 执行时的状态和时刻. 后续的集合 $Cont = \Sigma_{RT} \to \mathscr{P}(SProc)$, 每个 $h \in Cont$代表语句 s 执行后程序其余部分可能发生的行为.[5]

$\mathscr{M}_s$ 的定义由以下各个分句的定义构成:

• 赋值语句 $x := e$

$$\mathscr{M}_s[\![x := e]\!](h)(\sigma_T) = \{[\sigma'_{T'}, h(\sigma'_{T'})]\}$$

其中 $\sigma' = \sigma\{[\![e]\!](\sigma_T)/x\}$, $T' = \{t+1 : t \in T\}$. $\mathscr{M}_s$ 对于赋值语句的语义和通常的语义函数有一点重要的不同, 即考虑到了赋值语句的时间关系, 如果赋值语句开始执行的时刻是 t $(\in T)$, 则完成的时刻是 $t+1(\in T')$(我们假定赋值语句只花费一个单位时间).

• 发送语句 $O!\ e$, $O \in ONam^+$

$$\mathscr{M}_s[\![O!\ e]\!](h)(\sigma_T) = \{[O, [\![e]\!](\sigma_T), \sigma_{T'}, h(\sigma_{T'})]\}$$

其中 $T' = \{t+\delta : t \in T, \delta \in \mathbf{N}, \delta > 0\}$, 由于消息的发送依赖于运行环境(消息的传播方式, 运行时系统的拓扑结构等), 所以完成一个发送的时间是不确定的, 可以是任意多个单位时间. 如果消息发送前的时刻是 $t(\in T)$, 则完成的时刻可能是 $t+\delta(\in T')$, $\delta > 0$, $\delta \in \mathbf{N}$.

• 接收语句 $O?\ x$, $O \in ONam^+$

$$\mathscr{M}_s[\![O?\ x]\!](h)(\sigma_T) = \{[O, \lambda v. \sigma\{v/x\}_{T'}, \lambda v. h(\sigma\{v/x\}_{T'})]\}$$

其中 T' 的定义和意义都同发送语句. 如果 O 接收到值 v, 则完成该语句时的实时状态为 $\sigma\{v/x\}_{T'}$, 程序其余部分的行为是 $h(\sigma\{v/x\}_{T'})$.

• 顺序复合 $s_1; s_2$

$$\mathscr{M}_s[\![s_1; s_2]\!](h)(\sigma_T) = \mathscr{M}_s[\![s_1]\!](\mathscr{M}_s[\![s_2]\!](h))(\sigma_T)$$

后续 h 是一个从 Σ_{RT}到 $\mathscr{P}(SProc)$的函数, 表示语句 $s_1; s_2$ 后程序的行为. $\mathscr{M}_s[\![s_2]\!](h)$同样是一个从 Σ_{RT}到 $\mathscr{P}(SProc)$的函数, 它表示语句 s_2 和 s_2 后面的程序的行为, 可以作为一个后续. $\mathscr{M}_s[\![s_1]\!](\mathscr{M}_s[\![s_2]\!](h))(\sigma_T)$, 表示先执行 s_1, 然后执行 s_2, 最后是程序的其余部分.

• 延时语句 $wait\ d$, d 是整型表达式

$$\mathscr{M}_s[\![wait\ d]\!](h)(\sigma_T) = \{[\sigma_{T'}, h(\sigma_{T'})]\}$$

其中 $T' = \{t + [\![d]\!](\sigma_T) : t \in T\}$. 延时语句不改变状态, 使程序提起 $[\![d]\!](\sigma_T)$个单位时间, 如果它于时刻 $t(\in T)$开始执行, 则完成时的时刻是 $t + [\![d]\!](\sigma_T)(\in T')$.

• 卫哨语句 $\bigsqcup_{j=1}^{n} g_j \to s_j$

为了给出不确定分枝语句的语义, 需要几个辅助定义. 首先, 定义

$$waitvalue(g,\sigma_T)=\begin{cases}0, & \text{如果 } g\equiv b\wedge [\![b]\!](\sigma_T)\\ max\{[\![d]\!](\sigma_T),1\}, & \text{如果 } g\equiv wait\ d\\ & \vee(g\equiv b;wait\ d\wedge [\![b]\!](\sigma_T))\\ \infty, & \text{否则}\end{cases}$$

其中 g 是卫哨，$\sigma_T\in\Sigma_{RT}$. 对于一个卫哨的集合 G 和 $\sigma_T\in\Sigma_{RT}$，$minwait(G,\sigma_T)=min\{waitvalue(g,\sigma_T):g\in G\}$.

辅助函数：　$G:GInst\times\mathscr{P}(GInst)\rightarrow Cont\rightarrow\Sigma_{RT}\rightarrow\mathscr{P}(SProc)$

对于 $h\in Cont$，$\sigma_T\in\Sigma_{RT}$，A 是一个卫兵的集合，它的定义如下：

$$G[\![b,A]\!](h)(\sigma_T)=\begin{cases}\{[\sigma_{T'},h(\sigma_{T'})]\}, & \text{如果 } [\![b]\!](\sigma_T)\\ \varphi, & \text{否则}\end{cases}$$

其中 $T'=\{t+1:t\in T\}$；

$$G[\![wait\ d,A]\!](h)(\sigma_T)$$
$$=\begin{cases}\{[\sigma_{T'},h(\sigma_{T'})]\}, & \text{如果 } max\{[\![d]\!](\sigma_T),1\}=minwait(A\cup\{wait\ d\},\sigma_T)\\ \phi, & \text{否则}\end{cases}$$

其中 $T'=\{t+[\![d]\!](\sigma_T):t\in T\}$；

$$G[\![O!\ e,A]\!](h)(\sigma_T)$$
$$=\begin{cases}\{[O,[\![e]\!](\sigma_T),\sigma_{T'},h(\sigma_{T'})]\}, & \text{如果 } minwait(A,\sigma_T)\neq 0\\ \phi, & \text{否则}\end{cases}$$

其中 $T'=\{t+\delta:t\in T,1\leqslant\delta\leqslant minwait(A,\sigma_T)\}$；

$$G[\![O?\ x,A]\!](h)(\sigma_T)$$
$$=\begin{cases}\{[O,\lambda v.\sigma\{v/x\}_{T'},\lambda v.h(\sigma\{v/x\}_{T'})]\}, & \text{如果 } minwait(A,\sigma_T)\neq 0\\ \phi, & \text{否则}\end{cases}$$

其中 $T'=\{t+\delta:\tau\in T,1\leqslant\delta\leqslant minwait(A,\sigma_T)\}$；

$$G[\![b;g,A]\!](h)(\sigma_T)=G[\![b,A]\!](G[\![g,A]\!](h))(\sigma_T)$$

其中 $g\equiv O!\ e$ 或者 $g\equiv O?\ x$ 或者 $g\equiv wait\ d$.

以上按照卫哨 g 的不同形式分别定义了函数 G 的行为. ϕ 是空集，表示相应的卫哨不被选择. 对于纯布尔卫哨 b，如果 $[\![b]\!](\sigma_T)$ 为真，则这个纯布尔分枝可能被选择，并且需要一个单位时间. 在上下文关系 A 中，一个纯延时卫哨只能在下面条件下被选择：它的延时时间等于最小延时时间，并且在这期间内没有通讯发生. 一个为真的 I/O 指令，能够在最小延时时间内发生(因而被选择)，并且执行相应的通讯.

卫哨语句的语义如下：

$$\mathscr{M}_s[\![\mathop{\square}_{j=1}^{n} g_j\rightarrow s_j]\!](h)(\sigma_T)$$
$$=\begin{cases}\bigcup_{j=1}^{n} G[\![g_j,\{g_k:1\leqslant k\leqslant n,k\neq j\}]\!](\mathscr{M}_s[\![s_j]\!](h))(\sigma_T), & \text{如果 } \bigvee_{j=1}^{n}[\![g_j]\!](\sigma_T)\\ \{[\sigma_{T'},h(\sigma_{T'})]\}, & \text{否则}\end{cases}$$

其中 $T'=\{t+1:t\in T\}$. T' 之所以如此定义，是因为假定每个语句至少需要一个单位时间.

- 循环语句 $*[\mathop{\square}_{j=1}^{n} g_j\rightarrow s_j]$

循环语句重复执行卫哨语句$\square_{j=1}^{n} g_j \to s_j$，直到没有一个卫哨为真，它的语义如下给出：

$$\mathscr{M}_s[\![*[\square_{j=1}^{n} g_j \to s_j]]\!](h)(\sigma_T)$$

$$=\begin{cases} \mathscr{M}_s[\![\square_{j=1}^{n} g_j \to s_j]\!](\mathscr{M}_s[\![*[\square_{j=1}^{n} g_j \to s_j]]\!](h))(\sigma_T), & \text{如果} \bigvee_{j=1}^{n} [\![g_j]\!](\sigma_T) \\ \{[\sigma_{T'}, h(\sigma_{T'})]\}, & \text{否则} \end{cases}$$

其中 $T'=\{t+1:t\in T\}$.

3 对象的语义

对于一个对象来说，由于它对数据的封装，它的内部操作及内部状态是不可见的. 因此，一个对象从外部来看，可以简单看成一个消息发送、接收序列(的集合). 通过对它内部计算的屏蔽，它的语义函数可以由语句的语义函数 $\mathscr{M}_s$ 得到. 对象的域 $OProc$ 是满足方程：

$$OProc \cong \{q_0\} \cup (ONam^+ \times Val \times \mathscr{P}(\mathbf{N}) \times \mathscr{P}(OProc))$$
$$\cup (ONam^+ \times \mathscr{P}(\mathbf{N}) \times (Val \to \mathscr{P}(OProc)))$$

的唯一完备度量空间. 它的元素称为对象进程. 实际上，每个对象对应一个对象进程的集合. 如下定义的抽象算子 $\alpha: SProc \to OProc$，将屏蔽所有的内部计算：

$$\alpha(p_0)=q_0$$
$$\alpha([\sigma_T, P])=\alpha^*(P)$$
$$\alpha([O,v,\sigma_T,P])=[O,v,T,\alpha^*(P)]$$
$$\alpha([O,\varphi_T,f])=[O,T,\lambda v.\alpha^*(f(v))]$$
$$\alpha([\sigma^1_{T_1},\{[\sigma^2_{T_2},\cdots]\}])=q_0$$

其中 α^* 代表将 α 扩展到 $SProc$ 的幂集上的函数，对于 $P\subseteq SProc$，$\alpha^*(P)=\{\alpha(p):p\in P\}$. 对象的语义函数 $\mathscr{M}_o: Stat \to \mathscr{P}(OProc)$定义如下：

$$\mathscr{M}_o[\![s]\!]=\alpha^*(\mathscr{M}_s[\![s]\!](\lambda\sigma_T.\{p_0\})((\lambda x.nil)_{\{0\}}))$$

其中 $\lambda\sigma_T.\{p_0\}$代表空后续，表示语句 s 后程序不做任何事. $(\lambda x.nil)_{\{0\}}$是一个实时状态，表示 s 开始执行时，所有变量被初始化为 nil，时刻为 0.

4 程序的语义

程序的语义由程序本身与外界的通讯事件来描述. 假定 $ONam$ 中有 2 个特定元素 O_{in}，O_{out}，分别代表环境的输入和输出. 它们由 $OProc$ 中的对象进程 q_{in}和 q_{out}表示. q_{in}依赖于值的有限或无限序列 $w\in Val^\infty$，它被作为程序的输入： $q_{in}(w)=q^{RT}_{in}(N,w)$

对于任何 $T\in\mathscr{P}(\mathbf{N})$，$v\in Val$，$w\in Val^\infty$，$q^{RT}_{in}$定义如：$q^{RT}_{in}(T,\langle\rangle)=\{q_0\}$

$$q^{RT}_{in}(T,v\cdot w)=\{[*,v,T,q^{RT}_{in}(T+1,w)]\}$$

其中 $T+1$ 是$\{t+1:t\in T\}$的缩写.

q_{out}定义为： $q_{out}=q^{RT}_{out}(\mathbf{N})$

对于任意 $T\in\mathscr{P}(\mathbf{N})$，$q^{RT}_{out}(T)=\{[*,T,\lambda v.q^{RT}_{out}(T+1)]\}$.

程序的域 GProc 定义为：

$$GProc = \{r_0\} \cup \mathscr{P}(GStep)$$
$$GStep = (ONam^+ \times Val \times \mathscr{P}(\mathbf{N}) \times GProc)$$
$$\cup (ONam^+ \times ONam \times \mathscr{P}(\mathbf{N}) \times (Val \to GProc))$$
$$\cup (Comm \times GProc)$$
$$Comm = ONam \times Val \times ONam$$

域 $GProc$ 的元素称为全局进程，r_o 代表终止进程，其它的全局进程分为 3 个类型：发送、接收和通讯. 和 $SProc$ 不同，在发送 $[O_1, O_2, v, T, r]$ 和接收 $[O_1, O_2, T, f]$ 中，发送对象和接收对象都被标记. $[c, r]$ 代表一个成功的通讯，$c = [O_1, v, O_2]$ 表示值 v 从对象 O_1 传送到对象 O_2.

并行算子 $\|: GProc \times GProc \to GProc$ 定义如下：

Ⅰ) $r \| r_0 = r_0 \| r = r$

Ⅱ) $r_1 \| r_2 = \{\pi \|\_ r_2 : \pi \in r_1\} \cup \{\pi \|\_ r_1 : \pi \in r_2\}$

$\quad \cup \cup \{\pi_1 | \pi_2 : \pi_1 \in r_1, \pi_2 \in r_2$ 或者 $\pi_1 \in r_2, \pi_2 \in r_1\}$

Ⅲ) $[O_1, O_2, v, T, r] \|\_ r_2 = [O_1, O_2, v, T, r_1 \| r_2]$

$[O_1, O_2, T, f] \|\_ r_2 = [O_1, O_2, T, \lambda v. (f(v) \| r_2)]$

$[c, r] \|\_ r_2 = [c, r \| r_2]$

Ⅳ)

$$\pi_1 | \pi_2 = \begin{cases} \{[(O_1, v, O_2), f(v) \| r]\}, & \text{如果 } \pi_1 = [O_1, O_2^+, v, T_1, r], \pi_2 = [O_1^+, O_2, T_2, f], \\ & \text{并且存在 } t_1 \in T_1, t_2 \in T_2, t_1 = t_2 \\ \phi, & \text{否则} \end{cases}$$

2 个全局进程的并行包括它们之间并行的所有可能性（交叉并行模型）. 当进程 r 和终止进程 r_0 并行时，得到 r 本身. 如果 r_1 和 r_2 都不是终止进程时，则有 3 种情况发生：首先执行 r_1，或者首先执行 r_2，或者执行一个 r_1 和 r_2 间成功的通讯. $\|\_$ 称作左合并（*Left Merge*）算子，它定义了前 2 种情况发生时的行为. 在 $|$ 的定义中，考虑了实时因素，当发送 $[O_1, O_2, v, T, r]$ 和接收 $[O_1, O_2, T_2, f]$ 都表示一个从 O_1 到 O_2 的通讯时（匹配），但并不能保证是一个成功的通讯，它们必须能够在同一时刻完成，才是一个成功的通讯，即存在 $t_1 \in T_1, t_2 \in T_2, t_1 = t_2$.

算子 $\omega: OProc \to ONam \to GProc$，它将一个对象进程和一个对象名转换到一个全局进程，如下定义：

$$\omega(q_0)(O') = r_0$$
$$\omega([O, v, T, q])(O') = \{[O', O, v, T, \omega^*(q)(O')]\}$$
$$\omega([O, T, f])(O') = \{[O, O', T, \lambda v. \omega^*(f(v))(O')]\}$$

其中 ω^* 是将 ω 扩展到 $OProc$ 的幂集上的函数，即对于任意 $V \in \mathscr{P}(OProc)$，

$$\omega^*(V)(O') = \cup \{\omega(q)(O') : q \in V\}$$

程序的语义函数 $M_G: Prog \to Val^\infty \to GProc$：

$$\mathscr{M}_G [\![\langle O_1 :: s_1 \| \cdots \| O_n :: s_n \rangle]\!] (\omega) = \omega^*(\mathscr{M}_o [\![s_1]\!])(O_1)$$
$$\| \cdots \| \omega^*(\mathscr{M}_o [\![s_n]\!])(O_n) \| \omega^*(q_{in}(\omega))(O_{in}) \| \omega^*(q_{out})(O_{out})$$

5　域和语义函数

下面给出有关语义域和语义函数的 2 个定理.

定理 1. $SProc, OProc, GProc$ 是满足以下方程的唯一完备度量空间.

(E1) $$SProc \cong \{p_0\} \cup \{\Sigma_{RT} \times \mathcal{P}(id_{1/2}(SProc))\} \cup (ONam^+ \times Val \times \Sigma_{RT} \times \mathcal{P}_{cl}(id_{1/2}(SProc))) \cup (ONam^+ \times (Val \to \Sigma_{RT}) \times (Val \to \mathcal{P}_{cl}(id_{1/2}(SProc)))$$

(E2) $$OProc \cong \{q_0\} \cup (ONam^+ \times Val \times \mathcal{P}_{cl}(N) \times \mathcal{P}_{cl}(id_{1/2}(OProc))) \cup (ONam^+ \times \mathcal{P}_{cl}(N) \times (Val \to \mathcal{P}_{cl}(id_{1/2}(OProc)))$$

(E3) $$GProc \cong \{r_o\} \cup \mathcal{P}_{cl}(GStep)$$

$$GStep \cong (ONam^+ \times Val \times \mathcal{P}_{cl}(N) \times \mathcal{P}_{cl}(id_{1/2}(GProc))) \cup (ONam^+ \times ONam \times \mathcal{P}_{cl(N)} \times (Val \to id_{1/2}(GProc))) \cup (Comm \times id_{1/2}(GProc))$$

$$Comm \cong ONam \times Val \times ONam$$

证明:文献[6]中定理 5.4 的直接结论. □

令 $P = \Sigma_{RT} \to \mathcal{P}_{cl}(id_{1/2}(SProc))$. 循环语句语义的定义与如下定义等价:

$$\mathcal{M}_s [\![*[\mathop{\square}_{j=1}^{n} g_j \to s_j]]\!] = Fixed\ Point(\Phi),$$

其中 $\Phi:(P\to P)\to(P\to P)$ 定义如下:对任意 $p\in[P\to P]$,

$$\Phi(p) = \lambda\alpha \in P\lambda\sigma_T \in \Sigma_{RT}$$

$$\text{if } \bigvee_{j=1}^{n} [\![g_j]\!], \mathcal{M}_s [\![\mathop{\square}_{j=1}^{n} g_j \to s_j]\!] (p(\alpha))(\sigma_T)$$

$$\text{else}\{[\sigma_{T'}, \alpha(\sigma_{T'})]\}$$

定理 2. (Ⅰ)语义函数 $\mathcal{M}_s$ 是良定义的,即对任意语句 s, $\mathcal{M}_s[\![s]\!] \in P \to^1 P$.

(Ⅱ)Φ 是收缩函数,即 $\Phi \in (P\to P) \to^{1/2} (P\to P)$,有唯一的不动点.

证明:略.

对于 $\mathcal{M}_o$, $\mathcal{M}_G$ 有类似的定理及证明.

在以上给出的语义中,对并行采用的是交叉并行模型.这种模型对于分布式实时计算不是非常恰当.如何采用更恰当的分布式并行模型是今后进一步的研究工作.

致谢 在此,对舒敏同志给予本文的帮助,表示感谢.

参考文献

1 Koymans R, Shyamasundar R K, Roever W P *et al*. Compositional semantics for real-time distributed computing. Information and Computation, 1988, **79**: 210~256.

2 Hoare C A R. Communicating sequential processes. Communications of the ACM, 1978, **21(8)**: 666~667.

3 America P, Rutten J. A layered semantics for a parallel object-oriented language. Formal Aspects of Computing, 1992, **4**: 376~408.

4 America P, Bakker J, Kok J N *et al*. Denotational semantics of a parallel object-oriented language. Information and Computation, 1989, **83(2)**: 152~205.

5 Gordon M J C. The denotational description of programming languages: an introduction. Springer, 1979.

6 America P, Rutten J. Solving reflexive domain equations in a category of complete metric spaces. Journal of Computer and System Science, 1989, **39(3)**: 343~375.

DENOTATIONAL SEMANTICS OF AN OBJECT—ORIENTED DISTRIBUTED REAL—TIME LANGUAGE

Zuo Zhihong Gong Tianfu

(*Department of Computer Science University of Electronic Science and Technology Chengdu* 610054)

Abstract This paper gives a denotational semantics of an object—oriented distributed real—time language Mini CSP—R. At different layers, the semantics of statement, object and program are given concisely. By introducing real—time state, the real—time property of the language is described briefly in the frame of the denotational semantics.

Key words Object, real-time, distributed, parallel, denotational semantics.