Occam 语言的时态语义*

耿 骏 谭新明

(武汉水运工程学院)

A TEMPORAL SEMANTICS FOR OCCAM

Geng Jun and Tan Xinming

(Wuhan University of Water Transportation Engineering)

ABSTRACT

A compositional temporal semantics is given for a subset of Occam language. Local environments, local stores, and local channel states are introduced to form a true concurrency model. In the semantics various properties of distributed processes can be discussed.

摘 要

本文在给定的 *Occam* 子集上建立它的组合式时态语义。为了处理并发问题引入了局部环境、局部存储和局部通道状态，使其成为真正的并发语义，并在此语义下讨论了进程的各种分布式性质。

§1. 引 言

Occam 是一种包括顺序、并发和实时的分布式程序语言。它是在 CSP [1] 的基础上由 INMOS 公司推出的 [2]，已用于通信处理器(transputer)网络等领域。本文建立了 Occam 的一种时态语义，为了简明本文去掉了 Occam 中数组、重复和字节等部分；由于实时部分的时态语义定义尚有一定困难，本文也回避了它。故本文只给出了 Occam 的一个语言子集的时态语义。

1984 年周巢尘发表了顺序程序语言的结构式时态语义 [3]，1985 年他又发表了 CSP

* 1989年6月17日收到。

的组合式时态语义[4]，读者会看到本文采用了上述论文中的一些方法.

本语义中使用三类变量 ρ，s，ψ，这些变量随着时间的推移而改变它们的取值，称为时态变量. 任意时刻所有三类变量的一个取值决定了进程在该时刻的一个瞬时状态. 由常元标识符、变元标识符、通道标识符、过程标识符所对应的时态变量，统称为环境变量，即 ρ 变量. 环境变量用来记载进程执行过程中各个时刻常元标识符的取值、变元标识符所分配的存储单元、通道标识符所分配的通道码、过程标识符所对应的过程体. 存储单元所对应的时态变量称为存储变量，即 s 变量. 存储变量决定各个时刻各个存储单元所存的值. 通道码所对应的时态变量称为通道状态变量，即 ψ 变量. ψ 记载通道码两端的瞬时状态，以二元组表示，第一分量表示通道码输入端状态，第二分量表示通道码输出端状态，它们可是 req(表示某端要求通信)，可是 rej(表示某端拒绝通信)，可是一个数据(表示某端正在接受或者传输该数据).

在这个领域中，1985年 Howard Barringer 等用二阶时态逻辑给出了一种类 CSP 语言的组合式时态语义[5]. 这种类 CSP 语言与本文将给出的 Occam 子集有些相似，但在语义定义时两者的方法是不同的. 在[5]中引入时态标志变量 λ，λ 的取值可是 E(它进程操作)、π(内部操作)、c?a(输入操作)和 c!a(输出操作). 在[5]中明确地讨论了 λ 变量各种取值下进程的行为，因此进程的语义依赖于它进程的操作；但我们定义的进程语义独立于外部进程，只是由进程本身决定的；从这点上看，我们的语义比[5]中的语义更完整和抽象. 在[5]中引入二阶时态逻辑，用存在量词屏蔽局部变元和局部通道，使并发进程的语义定义中只出现单一的存储空间；但我们的语义使用一阶时态逻辑，对每个变元和通道分配存储单元和通道码，并发进程的每个子进程使用不同的状态空间，再由每个子进程状态空间构成统一的并发进程的状态空间；从这点看，[5]中的语义比我们的语义抽象，但我们的语义更接近于计算机实现.

1984年 A. W.Roscoe 用一个扩充的 failure 模型给出了一个 Occam 子集的指称语义[6]，在这个语义框架中不能表示进程的某些分布式性质，如公平性，但这在我们的时态语义中是可表示的. 而且，在[6]中把无限循环、活锁和静态出错统称为发散，无法区别发散的具体原因，然而在我们的时态语义中这些是可以区分的. 从上可看出在描述进程分布式性质方面本语义的表达能力更强些. 实际上在我们的时态语义中可定义[6]的指称语义，详见本文第五节.

本文第二节给出 Occam 子集的语法，第三节简单地介绍一阶时态逻辑，第四节阐述本语义的主要思想，第五节以本语义为框架定义[6]中的指称语义及各种分布式性质，在附录中列出本文所使用的语言的语法、语义和分布式性质的详细定义.

§2. Occam 语言

Occam 是一种支持并发应用的新型程序设计语言. 它以清晰的并行结构为特点，用通道通信来实现消息传输，明显地引入了局部时钟来支持实时应用，其非确定性机制也有所创新.

Occam 语言中，程序被称为进程(process)，以后我们用 P(带或不带下标)来表示. 为了表叙方便，本文对 Occam 语言的书写格式作了一些改动，其中最主要的一点是使之从

列式变成行式．例如，Occam 进程 P_1，P_2 的顺序组合

```
SEQ
  P1
  P2
```

将被改成 SEQ P_1 P_2.

Occam 保留了传统程序设计语言中的基本语言成分，如变元、常元说明(declaration)，赋值进程(assignment process)，顺序进程(sequential process)，条件进程(conditional process)，循环进程(repetitive process)，及类似于过程的命名进程(named process).

下面给出了一些例子:

VAR x	变元说明
DEF numb = 10	常元说明
PROC p = P	命名进程说明，指明p代表进程P．p是命名进程标识符．Occam 中不允许出现递归进程，所以 p 不能直接或间接地在 P 中以自由标识符出现．
x: = 1	赋值进程
SEQ x: = 1 y: = 2	顺序进程
IF b_1 P_1 b_2 p_2	条件进程，其中b_1，b_2是布尔表达式．该进程的执行过程是：先看 b_1，b_1 成立做 P_1，b_1 不成立再看 b_2，b_2 成立做 P_2，b_2 也不成立则本进程结束．
WHILE b P	循环进程
p	命名进程．p类似于传统程序设计语言中的过程调用．Occam 要求命名进程是静态约束的．

在 Occam 语言中，输入、输出被作为最基本的成分．输入进程(input process)和输出进程(output process)完成的是进程间的消息传输，而这种传输是经过通道进行的．一条通道只能连接两个不同的进程，数据传输只能沿单向进行，即输出端的进程只能往通道上放数据，输入端的进程只能从通道上取数据．通道在使用前必须说明，形如

CHAN c	指明 c 是通道标识符．

输出进程是往通道上送数据，如

c ! 5	将数值 5 放到通道 c 上．

输入进程是从通道上取数据，如

c ? x	取通道 c 上的值赋给变元标识符 x．

SKIP 是 Occam 的一个基本进程，它总能被执行，产生的唯一效果是本进程终止．

Occam 中设置了局部时钟，记为 NOW．时钟通过 WAIT 进程得到使用，如

```
WAIT NOW AFTER 100
```

该进程在 NOW 之值不大于 100 时一直处于等待，当 NOW 大于 100 时就执行，执行的结果是该进程终止．

Occam 中两进程 P_1，P_2 的并发结构形如 PAR P_1 P_2．它使得 P_1，P_2 能同时执行；

当 P_1，P_2 都终止时，整个并行进程(parallel process)才成功地结束．进程间的通信是通道上的消息传输，也就是以前面讲到的输入、输入进程为基础实现的，采用的是同步通信机制，即当通道的一端执行输入(输出)进程时，必须等待另一端执行相应的输出(输入)进程，这时并行进程才能执行，并实现通道上的消息传输，而且消息的发出和接收被认为是同时发生的，没有时间上的滞后．

Occam 中的选择进程(alternative process)在 E. W. Dijkstra 提出的卫士命令[7]上进行了扩充，使其卫士(guard)能是 SKIP、WAIT、INPUT 进程，或前面带布尔表达式的 SKIP、WAIT、INPUT 进程．选择进程是由若干个带卫士的进程(guarded process)组成的，而且选择进程本身也可作为带卫士的进程．例如：

卫士 `c ? x` 输入进程作为卫士，引入了外部选择．若通道c的输出端进程准备向 c 输出数据，则称该卫士就绪(Ready)．

`b & c ? x` 布尔表达式b与输入进程一起作为卫士．b成立且输入进程就绪，则该卫士就绪．

带卫士的进程 `b & c ? x P` 若卫士b & c ? x就绪，则该带卫士的进程就绪。

选择进程 `ALT g1 P1 g2 P2` 有两个带卫士的进程组成的选择进程，其中 g_1、g_2 是卫士．若两个带卫士的进程中有一个就绪，就执行该进程，先执行卫士，然后执行卫士后的进程；若两个都就绪，执行其中任意一个．

```
ALT g1 P1
    ALT g2 p2 g3 p3
```

这是一个由两个带卫士的进程组成的选择进程，其中一个带卫士的进程本身也是一个选择进程．g_2 P_2 与 g_3 P_3 中有一个就绪，ALT g_2 P_2 g_3 P_3 亦就绪．

下面给出一示意性 Occam 进程的例子，它是将两个单位缓存拼成一个两单位缓存．

例 1. CHAN comms: c1→ [x] →comms→ [x] →c2

```
PAR
WHILE TRUE
    VAR x:
    SEQ
      c 1 ? x
      comms ! x
WHILE TRUE
    VAR x:
    SEQ
      comms ? x
      c 2 ! x
```

P_1 P_2

该进程是由两个无限循环的顺序进程并发起来的．第一个循环不断地从通道 c1 上取

数据，然后往通道 comms 上送数据；第二个循环不断地先从 comms 上取数据，再向 c2 输出数据。

以上简要地介绍了 Occam 语言最主要、最基本的内容。Occam 语言的完整定义及应用请参见[2]、[8]。

本文所用的语言是 Occam 的一个子集。在该语言中没有时钟(clock)、向量(vector)等概念，因而在语法上也做了一些改动。我们这样做有的是为了简化，并不涉及问题的实质；有的则是因为在目前的语义框架内难以处理，如时钟。

本文所用语言的语法定义详见附录一。

§3. 时态逻辑

本文使用关于未来时间的线性时态逻辑，以下简单介绍用到的时态连接词和算子，详见[9]。

本时态逻辑中的基本变量有两类：一类是常规变量，另一类是时态变量。时态变量 t 表示一个随时间变化而改变值的动态变量，t 在各个时刻的值决定了 t 的一个取值，即 t 的取值为按时间次序构成的一个无穷序列。设无穷序列 $\sigma = t_0, t_1, t_2, \cdots$ 为 t 的一个取值，则 t_0 规定了 t 在当前时刻的值；t_1 规定了 t 在下个时刻的值；t_k 则是 t 在第 k 个时刻的值。关于 t 的一个时态公式就规定了满足这一公式 t 的所有可能取值，即无穷序列的一个集合。以下用 $\sigma \models A$ 表示 σ 满足公式 A。本逻辑中原子公式是由项和等词构成的公式。常规连接词有 ¬、∧、∨、⇒、≡、∃ 和 ∀(量词只辖常规变量)。时态连接词有○(下一时刻)、◇(某一时刻)、□(任意时刻)、U(直到)和 U⁻(除非)。

设 A 和 B 是公式，形式定义如下：

$\sigma \models A$ 当且仅当 $t_0 \models A$ (A 为原子公式)：

$\sigma \models \neg A$ 当且仅当 $\sigma \not\models A$：

$\sigma \models A \wedge B$ 当且仅当 $\sigma \models A$ 且 $\sigma \models B$：

$\sigma \models \exists x A(x)$ 当且仅当 存在 x 的取值 v 使得 $\sigma \models A(v)$：

∨、⇒、≡和∀的定义从略：

$\sigma \models \bigcirc A$ 当且仅当 $\sigma_1 \models A$ 其中 $\sigma_i = t_i, t_{i+1}, t_{i+2}, \cdots$：

$\sigma \models \Diamond A$ 当且仅当 $\exists i \geq 0 \quad \sigma_i \models A$：

$\sigma \models \Box A$ 当且仅当 $\forall i \geq 0. \sigma_i \models A$：

$\sigma \models A U B$ 当且仅当 $\exists i \geq 0 (\sigma_i \models B \& \forall j\ k \geq 0. \sigma_k \models A)$：

$A U^{-} B \equiv (A U B) \vee \Box A$。

○亦可作为时态算子作用于时态变量，表示该变量下一个时刻的取值。如：$(\bigcirc t) = 0$ 表示下一个时刻 t 的取值为 0，即 $\sigma \models (\bigcirc t) = 0$ 当且仅当 $t_1 = 0$。

本文中还使用 $\bigcirc^k A$ 表示第 k 个时刻起 A 真，类似地有 $\bigcirc^k t$ 表示第 k 个时刻 t 的值。

当出现多个时态变量时，如时态变量是多元组 $(x_1, x_2, x_3, \cdots, x_n)$，则其在某个时刻所取的值由各个变量在该时刻的值所决定，即多个时态变量取值为由多元组所构成的一个无穷序列。

§4. 时态语义

本文中用环境记载各种标识符的作用范围，用存储记录进程的各变元值的变化，用通道状态描述进程间的通信．这样，环境、存储和通道状态刻划了 Occam 进程的整个行为．故我们用以环境、存储和通道状态为时态变量的时态逻辑公式来定义各语法范畴的语义．

Occam 中同名标识符处于进程中不同位置可以有不同的含义．我们把一标识符具有相同含义的语法区域称为该标识符的作用范围．如例 1 中，P_1 中的 x 与 P_2 中的 x 虽然同名，但是两个完全不同的变元标识符．第一个说明 VAR x 中的 x 的作用范围是 P_1，第二个 VAR x 中 x 的作用范围是 P_2．为了记载各标识符的作用范围，我们将进程的每个标识符 id 按其类别(如变元、常元等)对应于某一时态变量 $\rho(id)$，用 $\rho(id)$的取值变化来标明标识符 id 的作用范围．所有这些时态变量就构成了进程当前的环境，记为 ρ．

当 id 是变元标识符时，$\rho(id)$的取值为存储单元，不同时刻可分配不同的存储单元，从而区分同名标识符的不同作用范围，即在不同的作用范围内，同一标识符以不同的存储单元为其局部地址．例 1 中的第一个说明 VAR x 可解释成将 x 与一存储单元 l_1 联系起来，执行完该说明后 x 所对应的时态变量 $\rho(x)$的值为 l_1，记为从下一时刻起 $\rho(x)$取值为 l_1，即$\bigcirc\rho(x)=l_1$．从进程环境的角度看，ρ 在下一时刻的值除了保持原有其它标识符所对应的时态变量的值不变外，还确定了 x 所时应的时态变量，其下一时刻的值为 l_1，这一事实记为$\bigcirc\rho=\rho[l_1/x]$．当遇到第二个 VAR x 时，应将该 x 与一新的存储单元 l_2 联系起来，表明这一 x 与上一 x 完全不同，上一 x 的作用范围到此结束．从下一时刻起，x 所对应的时态变量取值为 l_2，环境的变化记为$\bigcirc\rho=\rho[l_2/x]$．

本文让常元标识符所对应的时态变量取值为 Occam 表达式所有可能的取值，称为 Occam 值．常元标识符 i 所对应的时态变量 $\rho(i)$的取值也可随着 i 的作用范围的不同而改变．对常元标识符说明 DEF i=10，环境的变化可记为$\bigcirc\rho=\rho[10/i]$，表示 i 在当前的作用范围内取值为 10．让通道标识符所对应的时态变量取值为通道码．通道码表示实在的通道名(本文认为各实在的通道是不同名的)，同名的通道标识符在不同时刻可分配不同的通道码．对通道标识符说明 CHAN c，其环境的变化可记为$\bigcirc\rho=\rho[cd/c]$，表示通道标识符 c 所对应的时态变量 $\rho(c)$自下一时刻起取值为通道码 cd．

由于 Occam 要求命名进程是静态约束的，所以处理命名进程说明 PROC p=P 时，除了要记载命名进程体 P 外，还要记录下定义命名进程时的环境 ρ，因此执行完命名进程说明后，p 对应的时态变量的值是二元组(ρ, P)，这一事实记为$\bigcirc\rho=\rho[(\rho, P)/p]$．以 $\rho'(p)$和 $\rho''(p)$分别表示二元组的第一和第二个元素．

变元标识符在进程执行过程中所取的值是由其所对应的存储单元中存放的内容决定的．存储单元不同时刻可存放不同的内容，故被看成是一个时态变量．本文将存储单元 l 所对应的时态变量记为 s(l)，让 s(l)在 occam 值集 Val 上取值，于是 s(l)的当前值就是存储单元当前的内容，变元标识符 x 的当前值就是 x 所联系的存储单元当前的内容，即 $s(\rho(x))$．进程当前用到的存储单元所对应的时态变量的全体就是当前的存储，记为 s．

Occam 中直接完成改变变元标识符值的进程有赋值进程和输入进程．如赋值进程 x:

=2，执行完该进程后，x 之值变成 2，这一事实记为○s(ρ(x))=2，用存储的概念又可写成○s=s[2/ρ(x)]。至于输入进程如何改变存储的取值，请看输入进程的语义定义。

通道上的通信可用通道两端的状态来描述[4]。输入进程占有通道的输入端，相应的输出进程占有通道的输出端。通道端有三类状态：一是拒绝通信，记为 rej；一是请求通信，记为 req；还有一类是传输数据，这时端状态是被传输的数据 v，v∈Val。通道一端的进程只能控制属于自己一端的端状态，两个端的状态结合起来就描述了通道的行为。

环境中我们让通道标识符 c 与通道码 cd 联系起来，以区分同名但含义不同的通道标识符。这里我们将通道码 cd 所对应的时态变量记为 $\psi(cd)$，$\psi(cd)$的取值是一个二元组，二元组的第一个元素代表通道码 cd 的输入端的状态，记为 $\psi'(cd)$，第二个元素代表其输出端的状态，记为 $\psi''(cd)$。进程当前用到的通道码所对应的时态变量的全体称为通道状态，记为 ψ。

下面看看如何用通道状态来描述输入、输出进程的通信行为。

可以想象输入进程 c ? x 是如下执行的：通道端平常都处于 rej 状态，进程 c ? x 为了得到数据先在 c 的输入端发出请求信号 rcq，然后等待，期望 c 的输出端也出现请求输出数据的同样信号。一旦 c 的输出端也出现 req，则两进程达到同步，输出端的进程就沿 c 输出一数据 v，与此同时输入端的进程接收数据，故两端的状态都取值为 v。该数据被 c ? x 接收后赋给变元标识符 x 而改变存储，随后 c 的输入端又变成 rej 状态；在 c ? x 要求输入而 c 的输出端总不出现 req，则进程 c ? x 中 c 的输入端将永远处于请求通信状态。这一过程可用时态公式表示为

$$\psi'(\rho(c)) = req \quad U^- \quad \exists v.\psi'(\rho(c)) = \psi''(\rho(c)) = v \wedge \bigcirc\, s = s[v/\rho(c)] \wedge \bigcirc\, \psi'(\rho(c)) = rej$$

类似地输出进程 c ? 2 的执行过程可表示为

$$\psi''(\rho(c)) = req \quad U^- \quad \psi'(\rho(c)) = \psi''(\rho(c)) = 2 \wedge \bigcirc\, s = s \wedge \bigcirc\, \psi''(\rho(c)) = rej$$

以上介绍了环境、存储和通道状态，及如何用它们来刻划进程的行为，但都只是孤立地从各自的特殊情况出发讲的，从定义说明和进程语义的角度来看是很不完整的，至少有两个问题：

①应定义说明和进程的全部行为，即采用时态公式来定义语义时，要定义每个时态变量在每一时刻的值；

②应提供表达说明或进程转入后续的手段，从而可由多个子说明或子进程顺序连接为一个说明或进程。

以变元标识符说明、赋值进程和输入进程为例，看如何定义它们的语义。

$$[\,VAR\ \ x\,] =_{df} \bigcirc\, \rho = \rho[new(s)/x] \wedge \bigcirc\, s = s \wedge \bigcirc\, \psi = \psi \wedge \bigcirc\, skip$$

该定义的第一部分从直观上可理解为对给定的存储 s，在 s 对应的存储单元中找出一未曾使用过的单元，这个单记为 new(s)，分配给变元标识符 x。第二部分○s=s∧○ψ=ψ是说变元标识符说明不影响存储和通道状态。最后部分中的 skip 是一给定的时态命题变量，用作转向后续说明或进程的标志。○skip 表示自下一时刻起，控制转向后续的说明或进程。

$$[\,x := e\,] =_{df} \bigcirc\, \rho = \rho \wedge \bigcirc\, s = s[\mathcal{E}[e]\ \rho s/\rho(x)] \wedge \bigcirc\, \psi = \psi \wedge \bigcirc\, skip$$

其关键部分是说将表达式 e 的值 $\varepsilon[e]\rho s$ 赋给变元标识符 x，这个值放在与 x 对应的存储单元 $\rho(x)$里，而其它都不改变。ε 是计算表达式值的函数，给出表达式 e 中的变元标识符 x、常元标识符 i 的值 $s(\rho(x))$、$\rho(i)$，可算出 e 的值。例如 x+y 的值为 $s(\rho(x))+s(\rho(y))$。本文对 ε 不作具体规定，假定它已存在，且计算规则与通常理解的一致。

$$[\,c?x\,] =_{df} \bigcirc \rho=\rho \wedge \bigcirc s=s \wedge \bigcirc \psi=\psi[(req,rej)/\rho(c)] \quad U^-$$
$$\bigcirc \rho=\rho \wedge \bigcirc s=s \wedge \bigcirc \psi\backslash\rho(c)=\psi\backslash\rho(c) \wedge \exists v.\ \bigcirc \psi(\rho(c))=(v,rej)\wedge$$
$$\bigcirc(\bigcirc \rho=\rho \wedge \bigcirc s=s[v/\rho(x)] \wedge \bigcirc \psi=\psi[(rej,rej)/\rho(c)] \wedge \bigcirc skip)$$

$\psi\backslash\rho(c)$表示在 ψ 中去掉 x 所对应的时态变量 $\rho(c)$后得到的新的通道状态。U^-前面的公式是说输入进程请求从通道码为 $\rho(c)$的通道上取得数据(通道的输入端被置成 req 状态，其它都保持不变)。U^-后面的公式是说，若通道上有数据通过，则 x 得到该数据，通道的输入端又成为 rej 状态，并转入后续进程。在该语义中，没有规定通信机制，通信可以是同步的，也可以是异步的[4]。

那么后续的说明或进程如何接过前面的说明或进程的结果和控制继续执行下去，也就是如何实现说明或进程的顺序组合呢？本文采取[3]中的方法，用逻辑中的公式代换来实现。设 A、B 是公式，$[A]^B_{skip}$ 表示将 A 中的 skip 用 B 来代替后得到的公式。说明的顺序组合和顺序进程可分别定义为

$$[\,D_1 : D_2\,] =_{df} [[\,D_1\,]]^{[\,D_2\,]}_{skip}$$

$$[\,(SEQ \quad P_1 \quad P_2)\,] =_{df} [[\,P_1\,]]^{[P_2]}_{skip}$$

例如：

$$[\,(SEQ \quad x:=1 \quad y:=2)\,]$$
$$=_{df}[\,\bigcirc \rho=\rho \wedge \bigcirc s=s[1/\rho(x)] \wedge \bigcirc \psi=\psi \wedge \bigcirc skip]^{\bigcirc \rho=\rho\wedge \bigcirc s=s[2/\rho(y)]\wedge \bigcirc \psi=\psi\wedge \bigcirc skip}_{skip}$$
$$\equiv \bigcirc \rho=\rho \wedge \bigcirc s=s[1/\rho(x)] \wedge \bigcirc \psi=\psi \wedge \bigcirc^2\rho=\bigcirc \rho \wedge \bigcirc^2 s=\bigcirc s[2/\bigcirc \rho(y)] \wedge \bigcirc^2\psi=\bigcirc$$
$$\wedge \bigcirc^2 skip$$

当给并行进程定义语义时遇到了两个问题：

①构成并行进程的各子进程应完全独立，形成真正的并行(true concurrency)。但若直接用前面定义的顺序进程作子进程构成并行进程的语义，会出现在某些时刻一些时态变量取值不一致而导致矛盾的情况；

②需为并行进程确定其后续，也就是给并行的各子进程找一公共的结束点。

为了使子进程独立，本文引入了局部环境、局部存储和局部通道状态，用局部于子进程的环境、存储和通道状态来描述子进程的行为。子进程的语义完全由它自己决定，各子进程之间不会出现因逻辑公式中出现同名变量的取值不同而产生矛盾的情况。通过给不同子进程以不同的名字，让环境、存储和通道状态带上子进程的名字作为下标形成局部环境、局部存储和局部状态。

命名是按 Occam 进程的层次结构逐层进行的。组成外层进程的子进程的层次被认为比外层进程深一层，最外层进程的名字为空；若第 j 层是并行进程，则第 j+1 层的各子进程的名字由第 j 层并行进程的名字拼上第 j+1 层的子进程在并行进程中的位置构成。

当不考虑并行进程中各子进程的公共结束点时，并行进程 PAR P_1 P_2 的语义可定义

为

$$[(PAR \quad P_1 \quad P_2)]_h =_{df} \bigwedge_{j=1}^{2}(\rho_{hj} = \rho \wedge s_{hj} = s \wedge \psi_{hj} = \psi \wedge [[P_j]_{hj}]_{\rho,s,\psi}^{\rho_{hj},s_{hj},\psi_{hj}}) \tag{1}$$

h 是进程 PAR P_1 P_2 的名字，ρ、s、ψ 可认为是 PAR P_1 P_2 的环境、存储、通道状态，称之为全局环境、全局存储、全局通道状态。h1(h2)是 $P_1(P_2)$的名字，$\rho_{h1}(\rho_{h2})$、$s_{h1}(s_{h2})$、$\psi_{h1}(\psi_{h2})$是 $P_1(P_2)$的局部环境、局部存储、局部通道状态。(1)是说：$P_1(P_2)$的局部环境、存储、通道状态从全局环境、存储、通道状态那里得到初值；$P_1(P_2)$的语义由它自己独立确定，但只作用在局部于它的环境、存储、通道状态上(语义公式中经代换出现的是局部环境、存储、通道状态)；P_1 与 P_2 语义的合取就是 P_1，P_2 并行执行的语义。

并行进程结束时，转入后续进程的应是全局环境、存储和通道状态。在并行进程的执行过程中，由于各子进程只作用于各自局部环境和局部通道状态，全局环境和全局通道状态的值没有改变，但并行进程的全局变元标识符可在且只可在一个子进程内被赋值，故全局存储的值随着并行进程的执行而改变，是各局部存储值的总和，用⊕来表示这种总和。于是并行进程的语义定义可改进为

$$[(PAR \quad P_1 \quad P_2)]_h =_{df} \exists \rho_0 \cdot s_0, \psi_0, \rho_0 = \rho \wedge s_0 = s \wedge \psi_0 = \psi \wedge [\bigwedge_{j=1}^{2}(\rho_{hj} = \rho \wedge s_{hj} = s \wedge \psi_{hj} = \psi \wedge [[P_j]_{hj}]_{\rho,s,\psi}^{\rho_{hj},s_{hj},\psi_{hj}})]_{skip}^{\rho=\rho_0 \wedge s=s_0 \oplus_{j=1}^{2} s_{hj} \wedge \psi=\psi_0 \wedge skip} \tag{2}$$

其中 $s_0 \oplus_{j=1}^{2} s_{hj}$ 的定义为

$$s_0 \oplus_{j=1}^{2} s_{hj}(\rho(x)) = \begin{cases} s_{h1}(\rho(x)), & 若\ s_{h1}(\rho(x)) \neq s_0(\rho(x)); \\ s_{h2}(\rho(x)), & 若\ s_{h2}(\rho(x)) \neq s_0(\rho(x)); \\ s_0(\rho(x)), & 否则。 \end{cases}$$

这里 ρ_0、s_0、ψ_0 是常规变量。它们记下了进入并行进程时，全局环境、存储和通道状态的值。并行进程执行完后，全局环境和全局通道状态之值应恢复成 ρ_0 和 ψ_0，全局存储之值是 $s_0 \oplus_{j=1}^{2} s_{hj}$。这种思想在处理命名进程和带说明的进程(D: P)时也要用到，因为涉及到新环境的引入和老环境的恢复(见附录二)。

(2)中的 skip 是并行进程的后续，即各子进程的公共出口。从直观上讲，每一子进程独立地执行，结束后应等待其它子进程做完，所有子进程都成功地结束后才能转入后续的进程。

为了标明子进程是否处于等待状态，本文在子进程的局部环境中设置一特殊标志 ω，此 ω 不能用作进程中的标识符，当 $\rho_{hj}(\omega)$取 ff 值时，说明子进程不处于等待状态，当 $\rho_{hj}(\omega)$取 tt 值时说明子进程处于等待状态。子进程处于等待状态时，除局部环境在 ω 处取 tt 外其它都保持不变，而且至少有一另外的子进程未完成自身的执行。例如(2)中的 P_1 处于待状态时应满足

$$\bigcirc \rho_{h1} = \rho_{h1}[tt/\omega] \wedge \bigcirc s_{h1} = s_{h1} \wedge \bigcirc \psi_{h1} = \psi_{h1} \wedge (\bigvee_{j=1}^{2} \rho_{hj}(\omega) = ff).$$

当所有子进程都进入等待状态时，并行进程就该转入后续进程，即

$$(\overset{2}{\underset{j=1}{\wedge}}\rho_{hj}(\omega)=tt)\wedge skip$$

用公式

$$\bigcirc\ \rho_{h1}=\rho_{h1}[tt/\omega]\wedge\bigcirc\ s_{h1}=s_{h1}\wedge\bigcirc\ \psi_{h1}=\psi_{h1}\wedge(\overset{2}{\underset{j=1}{\vee}}\rho_{hj}(\omega)=ff)\qquad U^-$$

$$(\overset{2}{\underset{j=1}{\wedge}}\rho_{hj}(\omega)=tt)\wedge skip$$

代替〔P_1〕$_{h1}$中的 skip，表示 P_1 处于等待直到由公共出口转入后续进程的可能性。

综合上述就可以写出并行进程完整的语义了。

$$[(PAR\quad P_1\quad P_2)]_h=_{df}\exists\rho_0,s_0,\psi_0.\rho_0=\rho\wedge s_0=s\wedge\psi_0=\psi\wedge$$
$$[\overset{2}{\underset{j=1}{\wedge}}(\rho_{hj}=\rho[ff/\omega]\wedge s_{hj}=s\wedge\psi_{hj}=\psi\wedge$$
$$[[P_j]_{hj}]^{\rho_{hj},s_{hj},\psi_{hj},wait(hj)}_{\rho,s,\psi,skip})]^{\rho=\rho_0\wedge s=s_0\overset{2}{\underset{j=1}{\oplus}}s_{hj}\wedge\psi=\psi_0\wedge skip}_{skip}$$

其中 wait(hj)为

$$\bigcirc\ \rho_{hj}=\rho_{hj}[tt/\omega]\wedge\bigcirc\ s_{hj}=s_{hj}\wedge\bigcirc\ \psi_{hj}=\psi_{hj}\wedge(\vee^2_{j=1}\rho_{hj}(\omega)=ff)\qquad U^-$$
$$(\wedge^2_{j=1}\rho_{hj}(\omega)=tt)\wedge skip$$

循环进程 WHILE b P 的语义从直观上可定义为

$$[WHILE\quad b\quad P]_h=_{df}\mathcal{B}[b]\rho s\wedge[[P]_h]^{[WHILE\quad b\quad P]_h}_{skip}\vee$$
$$\neg\mathcal{B}[b]\rho s\wedge skip$$

其意思是说，若 b 成立就做 P，然后在新的 ρ、s、ψ 下再做 WHILE b P；要是 b 不成立的话，循环进程就结束。$\mathcal{B}$ 是计算布尔表达式值的函数，给定 b 及 b 中变元标识符 x、常元标识符 i 之值 s(ρ(x))、ρ(i)，由 $\mathcal{B}$ 就可算出 b 的值。

但上面的定义中出现了递归，这样定义的〔WHILE b P〕$_h$是否存在，若存在又是什么呢？本文用指称方法中通用的处理递归定义的手段来解决这个问题。

在 well-formed 时态逻辑公式集合 TL 上定义偏序 $\sqsubseteq$，满足

$B\sqsubseteq A$ 当且仅当 $A\Rightarrow B$ $\qquad A,B\subseteq TL$

则(TL，$\sqsubseteq$)是完全偏序集。其最小元素是 true；若$\{A_j\}$是关于 $\sqsubseteq$ 的 ω 链，那么它的上确界是 $\forall j.Aj$。

设 $\theta\in TL$，$W(\theta)=_{df}\mathcal{B}$〔b〕ρs∧〔〔P〕$_h$〕$^{\theta}_{skip}\vee\neg\mathcal{B}$〔b〕ρs∧skip，可以证明 W 是连续的，W: TL$\to_c$TL，W 有最小不动点 fix(W)满足 fix(W)=W(fix(W))，这个最小不动点就是 $\forall n.W^n$(true)。故可用 $\forall_n$、W^n(true)作为〔WHILE b P〕$_h$的定义。

整个语言的语义定义详见附录二。

§5. 讨 论

在[6]中为定义进程的指称语义引入了刻划进程行为的三个重要概念，迹(记载所发生通信的通道码和传输的值)、拒绝集(拒绝通信的通道码集)和终止存储集(进程终止时所有

存储的集合)。事实上本时态语义已包含了为引入这三个概念所必须的基本内容。

设迹为 tr，它是一个通信序列串，其中每个元素形如：cd.v (cd 是通道码，v 是数据)，$tr=cd_1.v_1^{\frown}cd_2.v_2\cdots cd_{n-1}.v_{n-1}^{\frown}cd_n.v_n$ (n=1，2，3，…)或 tr=〈〉。在本语义中为定义迹可引入同名时态变元 tr，开始时 tr=〈〉。下一时刻 tr 的值，即○tr，与当前时刻有关，若当前时刻没发生通信，则○tr=tr；若当前时刻发生通信，如 pass(cd，v) (表示当前时刻通道码 cd 的输入端与输出端取值为数据 v)，则○tr=$tr^{\frown}cd.v$。

设拒绝集为 X，在[6]中用✓表示终止，给定一个进程 P，它的通道码集合为 P_{cd}，则任一时刻 $X\subseteq\{cd.v|cd\in P_{cd},\ v\in Val\}\cup\{\surd\}$。当某个 cd.v 属于 X 时，表示通道 cd 所对应的通道此时刻拒绝传递信息 v；当✓属于 X 时表示此刻进程不会终止。在本语义中进程终止表示为 skip，故为在本时态语义中定义拒绝集，必须规定□(skip⇒✓∉ X)。本文中有两类通道：一类是内部通道，另一类是外部通道。内部通道用于 P 中子进程间的通信，外部通道用于 P 进程与它进程间的通信。在[6]中外部进程从不拒绝和 P 进行通信。当 cd 对应是 P 的外部通道码时，若 P 控制 cd 的输入端，则有□(ψ'(cd)=req⇒X∩cd.Val=φ)；若 P 控制 cd 的输出端，那么 cd 所对应的通道不能拒绝传输值 v，即□((ψ''(cd)=req U $\neg\psi''$(cd)=V)⇒cd.v∉ X)。当 cd 是 P 的内部通道码时，若 cd 的两端都在要求通信，那么 cd 所对应的通道不能拒绝传输值 v，即□((ψ(cd)=ψ''(cd)=req U $\neg\psi'$(cd)=v)⇒cd.v∉ X，其中没考虑进程中 ψ 的名字 h。

给定一个进程 P，设其终止存储集为 store，若在某时刻 P 终止，则 store 为此时刻的存储，即□(skip⇒store=$\{S_h|$h 是进程名$\}$)；若在某时刻 P 不终止，则 store 为空集，即□(¬skip⇒store=φ)。由此可见本时态语义比之[6]有更强的表达性。

下面我们说明如何定义进程的一些分布式性质(详见附录)。

终止性：表示进程在某个时刻会终止，即◇skip。

无死锁：表示进程是终止的，即◇skip；或者总是在将来某个时刻有一个通道在进行通信，即□◇pass(cd，v)：或者在某个时刻以后总是不存在任何一个通道要求通信，即◇□(¬|∃ cd∈α.ψ'(cd)=req∧¬ ∃ cd∈β.ψ''(cd)=req)。其中，α，β 分别表示进程 P 的所有输入通道码集合与所有输出通道码集合。

无活锁：表示除非进程总是不进行通信，否则必须不时地进行外部通信。无活锁用时态公式表示为

◇□(¬∃ cd∈α∪β，v∈Val. pass(cd，v))∨

□◇(∃ cd∈(α−β)∪(β−α)，v∈ Val.pass(cd，v))。

公平性：表示一个通道码无穷多次就绪那么终究应会发生通信。假定外部环境是理想的，即外部总是准备与进程 P 中的通道进行通信。所以当 cd 是一个外部的输入通道码，它的就绪可定义为 ψ'(cd)=req；当 cd 是一外部的输出通道码，它的就绪可定义为 ψ''(cd)=req；当 cd 是一个内部的通道码，它的就绪为 ψ'(cd)=ψ''(cd)=rcq，其中没考虑 ψ 的进程名 h。本文把一个通道码 cd 的就绪记为 ready(cd)，故公平性可定义为

¬◇□(◇ready(cd)∧¬∃ v∈Val.pass(cd，v))。

笔者在 Occam 的时态语义的研究中始终得到导师周巢尘教授的精心指导。本文是笔

者硕士论文的一部分．在本文的撰写过程中，周巢尘教授提出了许多建设性的意见。在此，笔者谨致以衷心的感谢。

附录一　语言的抽象语法

X——变元标识符集　　x∈X

C——通道标识符集　　c∈C

I——常元标识符集　　i∈I

P——命名进程标识符集　　p∈P

Exp——表达式集　　e∈Exp

Bexp——布尔表达式集　　b∈Bexp

G——卫士集　　g∈G

g::=b&SKIP | b&c?x

GP——带卫士的进程集　　gp∈GP

gp::=g P | (ALT {gp})

CP——带条件的进程集　　cp∈CP

cp::=b P | (IF {cp})

Proc——进程集　　P∈Proc

P::=SKIP | x:=e | c?x | c!e

|(SEQ {P})|(PAR {P})

|(ALT {gp})|(IF {cp})

|WHILE b P | p | D:P

Dec——说明集　　D∈Dec

D::=VAR x

| CHAN c | DEF i=e

|PROC p=P |D:D

P 既用作进程标识符，又作为命名进程标识符集的名字，但这是在不同情况下，两者是可以区别的。{ } 用来表示括号内的语法成分可以重复出现多次．各集合上的变元可以带下标，如 P_1，P_2 等。

附录二　语言的时态语义

§1. 记号说明

1. N——自然数集　　j，m，n∈N
2. T——逻辑值集　　T={true，false}
3. Val——Occam 值集　　v∈Val
4. Estate——通道的端状态集

Estate={rej，req}∪Val

5. ID——标识符集　　id∈ID

$\dot{ID} = X \cup C \cup I \cup P$

6. Loc——存储单元集 $l \in Loc$

Loc被划分成若干段，各段的单元个数不一定相同，且假设段是足够多的，各段内的单元也是足够多的。

7. Code——通道码集 $cd \in Code$

对Code有类似于Loc的要求，但不要求通道码的个数与Loc中的单元个数一样多，也不要求两者的段之间、段内元素数量之间有某种对应关系。

8. Env——环境集 $\rho \in Env$

$$\rho(id] \in \begin{cases} Loc & \text{若 } id \in X; \\ Code & \text{若 } id \in C; \\ Val & \text{若 } id \in I; \\ Env \times Proc & \text{若 } id \in P. \end{cases}$$

9. Store——存储集 $s \in Store$

$s(l) \in Val$

10. Cstate——通道状态集 $\psi \in Cstate$

$\psi(cd) \in Estate \times Estate$

11. new $(s) \in Loc$

对于给定的存储s，在s所对应的存储单元段中找出一未曾使用过的单元，这个单元记为new(s)。

存储与存储单元段的对应关系要求是一一的。

12. new$(\psi) \in Code$

对给定的通道状态ψ，在ψ所对应的通道码段内找出一未曾使用过的通道码，这个通道码记为new(ψ)。

这里也有类似于11的假设。

13. H——进程名集 $h \in H$

$H = \{1,2 \cdots,9\}^*$

$h'h'' =_{df} h' h'' \qquad h', h'' \in H$

§2. 语义公式

1. $[VAR \quad x] =_{df} \bigcirc \rho = \rho[new(s)/x] \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi \wedge \bigcirc skip$

2. $[CHAN \quad c] =_{df} \bigcirc \rho = \rho[new(\psi)/c] \wedge \bigcirc s = s \wedge$
$= s \wedge \bigcirc \psi = \psi[(rej, rej)/new(\psi] \wedge \bigcirc skip.$

3. $[DEF \quad i = e] =_{df} \bigcirc \rho = \rho[\mathcal{E}[e]\rho s/i] \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi \wedge \bigcirc skip$

4. $[PROC \quad p = P] =_{df} \bigcirc \rho = \rho[(\rho, P)/p] \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi \wedge \bigcirc skip$

5. $[D_1 : D_2] =_{df} [[D_1]]_{skip}^{[D_2]}$

6. $[SKIP]_h =_{df} \bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi \wedge \bigcirc skip$

7. $[x := e]_h =_{df} \bigcirc \rho = \rho \wedge \bigcirc s = s[\mathcal{E}[e]\rho s/\rho(x)] \wedge \bigcirc \psi = \psi \wedge \bigcirc skip$

8.
$$[c?x]_h =_{df} \bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi[(req, rej)/\rho(c)] \quad U^-$$
$$\bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi \backslash \rho(c) = \psi \backslash \rho(c) \wedge \exists v. \bigcirc \psi(\rho(c)) = (v, rej) \wedge$$
$$\bigcirc (\bigcirc \rho = \rho \wedge \bigcirc s = s[v/\rho(x)] \wedge \bigcirc \wedge \psi = \psi[(rej, rej)/\rho(c)] \wedge \bigcirc skip)$$

9.
$$[c!e]_h =_{df} \bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi[(rej, req)/\rho(c)] \quad U^-$$
$$\bigcirc = \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi \backslash \rho(c) = \psi \backslash \rho(c) \wedge \bigcirc \psi(\rho(c)) = (rej, \mathcal{E}[e]\rho s) \wedge$$
$$\bigcirc (\bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi[(rej, rej)/\rho(c)] \wedge \bigcirc skip)$$

10. $[(SEQ \quad P_1 \cdots P_n)]_h =_{df} [\cdots [[P_1]_h]_{skip}^{[P_2]_h} \cdots]_{skip}^{[P_n]_h}$

11.
$$[(PAR/P_1 \cdots P_n)]_h =_{df} \exists \rho_0, s_0, \psi_0. \rho_0 = \rho \wedge s_0 = s \wedge \psi_0 = \psi \wedge$$
$$[\bigwedge_{j=1}^{n} (\rho_{hj} = \rho[ff/\omega] \wedge s_{hj} = s \wedge \psi_{hj} = \psi \wedge$$
$$[[P_j]h_j]_{\rho, s, \psi, skip}^{\rho_{hj}, s_{hj}, \psi_{hj}, wait(hj)})]_{skip}^{\rho = \rho_0 \wedge s = s_0 \oplus_{j=1}^{n} s_{hj} \wedge \psi = \psi_0 \wedge skip}$$

$$wait(hj) =_{df} \bigcirc \rho_{hj} = \rho_{hj}[tt/\omega] \wedge \bigcirc s_{hj} = s_{hj} \wedge \bigcirc \psi_{hj} = \psi_{hj} \wedge (\bigvee_{j=1}^{n} \rho_{hj}(\omega) = ff)$$
$$U^- \quad (\bigwedge_{j=1}^{n} \rho_{hj}(\omega) = tt) \wedge skip$$

$$s_0 \oplus_{j=1}^{n} s_{hj}(\rho(x)) =_{df} \begin{cases} s_{hj}(\rho(x)) & \text{若 } \exists j, 1 \le j \le n. s_{hj}(\rho(x)) = s_0(\rho(x)); \\ S_0(\rho(x)) & \text{否则。} \end{cases}$$

12.
$$[(ALT \quad gp_1 \cdots gp_n)]_h =_{df} \neg ready(ALT \quad gp_1 \cdots gp_n) \wedge \bigcirc \rho = \rho \wedge \bigcirc s = s \wedge \bigcirc \psi = \psi$$
$$U^- \quad \bigvee_{j=1}^{n} (ready(gp_j) \wedge [gp_j]_h)$$

$$[g \quad P]_h =_{df} [[g]_h]_{skip}^{[P]_h}$$

$$[b\&SKIP]_h =_{df} [SKIP]_h$$

$$[b\&c?x]_h =_{df} [cc?x]_h$$

$$ready(ALT \quad gp_1 \cdots gp_n) =_{df} \bigvee_{j=1}^{n} ready(gp_j)$$

$$ready(g \quad P) =_{df} ready(g)$$

$$ready(b\&SKIP) =_{df} \mathcal{B}[b]\rho s$$

$$ready(b\&c?x) =_{df} \mathcal{B}[b]\rho s \wedge \exists j. \psi_j''(\rho(c)) = req$$

13. $[(IF\ cp_1 \cdots cp_n)]_h =_{df} \bigvee_{j=1}^{n}(\bigwedge_{m=1}^{j} \neg hold(cp_{m-1})) \wedge hold(cp_j) \wedge [cp_j]_h$

$$\vee (\bigwedge_{j=1}^{n} \neg hold(cp_j) \wedge skip$$

注：$hold(c\rho_0) =_{df} false$

$$[b \quad P]_h =_{df} [P]_h$$

$$hold(IF\ cp_1 \cdots cp_n) =_{df} \bigvee_{j=1}^{n} hold(cp_j)$$

$$hold(b \quad P) =_{df} B[b]\rho s$$

14.

$$[WHILE\ b\ P]_h =_{df} \forall n.W^n(true)$$

$$W : TL \rightarrow_c TL$$

15. $W(\theta) =_{df} B[b]\rho s \wedge [[P]_h]^{\theta}_{skip} \vee \neg B[b]\rho s \wedge skip$

$[p]_h =_{df} \exists \rho_0 \cdot \rho_0 = \rho \wedge [\rho_{h_1} = \rho \wedge$

16. $[D : P]_h =_{df} \exists \rho_0 \cdot \rho_0 = \rho \wedge [\rho_{h1} = \rho \wedge$

$$[\ \bigcirc\ \rho = \rho'(p) \wedge \bigcirc\ s = s \wedge \bigcirc\ \psi = \psi \wedge \bigcirc\ [\rho''(p)]_{h1}]^{\rho_{h1}}_{\rho}]^{\rho=\rho_0 \wedge skip}_{skip}$$

$$[[[D]]^{[P]_{h1}}_{skip}]^{\rho_{h1}}_{\rho}]^{\rho=\rho_0 \wedge skip}_{skip}$$

其中 P 不是带说明的进程。

附录三　进程的分布式性质

给定进程 P，设 H_p 为 P 的进程名集合，C_p 为 P 的通道标识符集合，α 为 P 的输入通道码集合，β 为 P 的输出通道码集合。

$\alpha =_{df} \{\rho_h(c) | h \in H_p \wedge c \in C_p \wedge \Diamond \psi'_h(\rho_h(c)) = req\}$

$\beta =_{df} \{\rho_h(c) | h \in H_p \wedge c \in C_p \wedge \Diamond \psi''_h(\rho_h(c)) = req\}$

通道码 cd 上传递数据 v 可定义为

$Pass(cd, v, h, h') =_{df} \psi'_h(cd) = \psi''_{h'}(cd) = v$

这里假定通信是同步进行的。

终止性(Ter)

$Ter(P) =_{df} \Diamond skip$

无死锁(DF)

$$DF(P) =_{df} \Diamond skip \vee$$

$$\Box \Diamond \exists h, h' \in H_p, cd \in \alpha \cup \beta, v \in Val.\ Pass(cd, v, h, h') \vee$$

$$\Diamond \Box \forall h \in H_p.(\neg \exists cd \in \alpha.\psi'_h(cd) = req \wedge$$

$$\neg \exists cd \in \beta.\psi''_h(cd = req)$$

无活锁(LF)

$$LF(P) =_{df} \diamond\Box\forall h, h' \in H_p, v \in Val. \neg\exists cd \in \alpha \cup \beta. Pass(cd, v, h, h') \vee$$
$$\Box\diamond\exists h, h' \in H_p, v \in Val, cd \in (\alpha - \beta) \cup (\beta - \alpha). Pass(cd, v, H, h')$$

公平性(Fair):

假设外部环是理想的，即外部环境总是准备与它进程发生同步通信。

$$ready(cd) =_{df} cd \in \alpha - \beta \wedge \psi'_h(cd) = req \vee$$
$$cd \in \beta - \alpha \wedge \psi''_h(cd) = req \vee$$
$$cd \in \alpha \cup \beta \wedge \psi'_h(cd) = \psi''_{h'}(cd) = req$$
$$Fair(P) =_{df} \forall cd \in \alpha \cup \beta.$$
$$\neg\diamond\Box(\exists h, h' \in H_p.(\diamond ready(cd) \wedge \neg\exists v \in Val. Pass(cd, v, h, h')))$$

参考文献

[1] C.A.R. Hoare, "Communicating Sequential Processes", CACM, Vol.21, No.8, pp.666–667, 1978.

[2] INMOS Ltd., Occam Progamming Manual, Prentice–Hall International, 1984.

[3] 周巢尘，"结构式时态语义"，计算机应用与软件，第 1 卷，第 1 期，第 2 期，1984。

[4] Zhou Chaochen, "A Temporal Semantics of Communicating Processes", Proc. of PPCC–1, Sept. 1985, Australia.

[5] H.Barringer, R.Kuiper & A. Pnueli, "A Compositional Temporal Approach to a CSP–like Language", Formal Models in Programming, E.J. Neuhold & G. Chroust (Editors), Elsevier Science Publishers, B.V (North–Holland), IFIP, 1985.

[6] A.W. Roscoe, "Denotational Semantics for Occam", Seminar on Concurrency, Springer Verlag LNCS 197, 1985.

[7] E.D. Dijkstra, A Discipline of Programming, Prentice–Hall International, 1976.

[8] G.Jones, Programming in Occam, Prentice–Hall International, 1987.

[9] Z.Manna, "Verification of Sequential Programs: Temporal Axiomatization", Theoretical Foundations of Programming Methodology, D.Reidel Publishing Company, pp.53—102, 1982.