计算机科学 2002Vol. 29№. 10

并发计算范型:CCS 和 π-演算*)

The Paradigms of Concurrent Computations:CCS and π-Calculus

杜旭涛[1] 李舟军[1,2]

(国防科技大学计算机学院 长沙 410073)[1] (武汉大学软件工程国家重点实验室 武汉 430072)[2]

Abstract Concurrent theory is one of the most important and hot fields in theoretical computer science. In this paper, we first take a look at the semantic bases of sequential computations and concurrent computations. Using an example we demonstrate that function, which is actually the basic semantic idea of sequential computations, cannot be used as the semantic idea of concurrent computations. Then we introduce the Calculus of Communicating Systems (CCS) and the π-calculus together with a brief comparison between them. Finally, we show some basic ideas of typed π-calculus, a fairly new direction in the π-calculus.

Keywords Concurrent computations, CCS, π-calculus

1 引言

并发现象和并发系统在生活中随处可见:网络通信、移动电话系统、银行的信息流动、超市的物流系统都是典型的并发系统。所谓并发系统就是存在并发事件的系统。顺序计算是并发计算的特例,相比于并发计算是一个小得多的领域,其复杂性也小得多。

函数被用来作为顺序计算的公共语义框架的基础。λ-演算就是一个著名的原型。一个顺序程序从语义上可以看作是一个从状态到状态的函数。例如顺序程序 $P1$ 和 $P2$:

$P1:X:=2$

$P2:X:=1;X:=X+1$

其中,";"是顺序程序设计的基本连接符,表示顺序执行。显然,顺序程序 $P1$ 和 $P2$ 结束时,X 的值都将为 2。因此它们是相等的函数,所以,在顺序程序设计中,$P1$ 和 $P2$ 被视为等价的。

在并发系统中,情况会复杂得多。如果将 $P1$ 和 $P2$ 分别与另一个顺序程序并发执行,结果可能会出乎预料。考虑

$P3:X:=2$

我们使用|表示并发执行。那么,考虑如下的并发程序 $R1$ 和 $R2$

$R1:X:=2|X:=2$

$R2:(X:=1;X:=X+1)|X:=2$

它们的计算结果显然是不相等的,$R1$ 总将 X 映射为 2,而 $R2$ 可能产生的结果会有多个。

这种现象说明了一个重要的问题:在并发系统中,我们不能再将进程看作函数;$P1$ 与 $P2$ 不再是相等的,它们必须被区别开。而 $P1$ 与 $P2$ 的区别在于它们与存储器的交互情况不同。因此,在并发系统中,计算不再是函数计算,而是交互计算。交互计算的主体是进程,而进程的行为就是它与环境的交互。

基于以上观点,R. Milner 于 70 年代末提出通信系统演算 CCS(The Calculus of Communication Systems)作为并发系统的语义范型。后来,为了描述通信拓扑结构动态变化的系统,他又与 D. Walker 和 J. Parrow 于 80 年代末合作提出了 π-演算。

2 通信系统演算 CCS

为了研究方便,一开始 CCS 进程的每次通信都只是一次交互,其中没有数据的传递,这就是纯 CCS。后来,加入了对数据值的处理,使得输入输出都包含数据值,这就是传值 CCS。

2.1 纯 CCS

为了给出 CCS 的语法,我们首先假定下列记号:

- $\mathscr{A}$ 为名字的可数无穷集,其元素用 a,b,c 表示,每个名字代表一个接收动作;
- $\bar{\mathscr{A}}=\{\bar{a}|a\in\mathscr{A}\}$:为补名集,其元素用 $\bar{a},\bar{b},\bar{c}$ 表示,补名代表发送动作;
- $\mathscr{L}=\mathscr{A}\cup\bar{\mathscr{A}}$:为标号集,其元素用 l,l' 表示;
- $Act=\mathscr{L}\cup\{\mathscr{T}\}$:为动作集,其元素用 α,β 表示。其中,标号 l 表示进程的外部可见动作,$\mathscr{T}$ 代表进程的内部动作,对外不可见;
- $f:\mathscr{L}\to\mathscr{L}$ 为换名函数;
- $\mathscr{K}$:为进程标识符(常量)集,其元素用 A,B 表示。

用 ε 表示进程表达式的集合,其元素用 E,F 表示。进程表达式可由如下 BNF 规则给出:

$E::=\alpha.E|\Sigma_{i\in I}E_i|E_1|E_2|E\backslash L|E[f]|A$

进程构造算子的直观意义如下:

- $\alpha.E$:动作前缀,其中 $\alpha\in Act$;
- $\Sigma_{i\in I}E_i$:非确定选择,其中 I 表示任意的标记集。当 $I=\varnothing$ 时,记 $\Sigma_{i\in I}E_i$ 为 0,当 $I=\{1,2\}$ 时,记 $\Sigma_{i\in I}E_i$ 为 E_1+E_2;
- $E_1|E_2$:并发合成;
- $E\backslash L$:限制,表示 $L\cup\bar{L}$ 中的动作被禁止,其中 $L\subseteq\mathscr{L}$;
- $E[f]$:换名,$W[f]$的行为方式与 E 相同,但其中的动作名都经过 f 的替换;
- 每个进程常量 A 均有相应的定义式:$A\Leftarrow P$,其中 P 为进程表达式,可递归定义。我们可用结构化操作语义(SOS)方法定义进程表达式的语义。令 $P\xrightarrow{\alpha}Q$ 表示进程 P 可执行动作 α 演变为 Q,则进程表达式的语义可由图 1 中的迁移规则给出(其中省略了关于|的对称规则 Com_2)。

*)本文得到国家自然科学基金项目(No. 60073001, No. 69933030)和高等学校重点实验室访问学者基金的资助。

$$\text{Act}\ \frac{}{a.E \xrightarrow{a} E} \qquad \text{Sum}_j\ \frac{E_j \xrightarrow{a} E_j'}{\sum_{i\in I} E_i \xrightarrow{a} E_j'}\ j\in I$$

$$\text{Com}_1\ \frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F} \qquad \text{Com}_3\ \frac{E \xrightarrow{l} E',\ F \xrightarrow{\bar{l}} F'}{E|F \xrightarrow{\mathscr{T}} E'|F'}$$

$$\text{Res}\ \frac{E \xrightarrow{a} E'}{E\backslash L \xrightarrow{a} E'\backslash L}\ a\notin L\cup\bar{L} \qquad \text{Rel}\ \frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$$

$$\text{Con}\ \frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'}\ A\Leftarrow P$$

图 1　CCS 的迁移语义

其中，最重要的规则是 Com_3，它表示进程 E 和进程 F 进行一次内部通信。$\mathscr{T}$ 表示通信是内部的，外界不可见。而实际上，进程的所有动作都是通信，任何一个进程的任何一个动作都是与其它进程的通信。比如，进程 $a.E$ 可以作 a 动作然后成为进程 E，那么它实际上必然是与某个可以作 $\bar{a}$ 动作的进程进行了一次通信。如前文所述，并发计算的本质就是交互，也就是进程间的通信。

CCS 的核心问题是进程的等价性问题，即在什么意义下两个语法不同(即表达上有差异)的进程可以看成是行为等价(相等)的？CCS 能成功地应用于并发系统建模，主要得益于基于互摸拟概念的行为等价语义理论的建立(D. Park[1])和完善(R. Milner[2])。由于互模拟关系具有清晰的物理背景和良好的数学性质，为并发系统的分析和验证提供了一种强有力的手段和方法，故在进程代数中占有显著地位。

定义 2.1　对称二元关系 $R\subseteq\varepsilon\times\varepsilon$ 称为强互模拟，如果对任意 $(P,Q)\in R$ 有：

若 $P \xrightarrow{a} P'$，则存在 Q' 使得 $Q \xrightarrow{a} Q'$ 且 $(P',Q')\in R$。

若存在一个强互模拟 R 使 $(P,Q)\in R$，则称 P 和 Q 是强互模拟的，记为 $P\sim Q$。

强互模拟是同余关系，具有很好的数学性质。它将所有动作平等看待，即使是只有内部可见的 $\mathscr{T}$ 动作也不例外。而有时候，我们并不希望把 $\mathscr{T}$ 动作与其它动作平等看待，例如：

$$\mathscr{T}.a.0 \not\sim \mathscr{T}.a$$

但是，外部观察者却无法看出两个进程的区别，因为 $\mathscr{T}$ 动作不可见。于是，我们有必要在互模拟的定义中忽略 $\mathscr{T}$ 动作，这样就得到了弱互模拟关系。

首先，我们引入下列记号：$\hat{\mathscr{T}}=\varepsilon$；若 $a\neq\mathscr{T}$，则 $\hat{a}=a$。令 $\overset{\epsilon}{\Rightarrow}=(\xrightarrow{\mathscr{T}})^*$（即 $\xrightarrow{\mathscr{T}}$ 的自反传递闭包），$\overset{a}{\Rightarrow}=\overset{\epsilon}{\Rightarrow}\xrightarrow{a}\overset{\epsilon}{\Rightarrow}$（即 $\overset{\epsilon}{\Rightarrow}$、$\xrightarrow{a}$ 和 $\overset{\epsilon}{\Rightarrow}$ 三者的合成）。

定义 2.2　对称二元关系 $R\subseteq\varepsilon\times\varepsilon$ 称为弱互模拟，如果对任意 $(P,Q)\in R$ 有：

若 $P \xrightarrow{a} P'$，则存在 Q' 使得 $Q\overset{\hat{a}}{\Rightarrow}Q'$ 且 $(P',Q')\in R$。

若存在一个弱互模拟 R 使 $(P,Q)\in R$，则称 P 和 Q 是弱互模拟的，记为 $P\approx Q$。

遗憾的是，弱互模拟不是同余关系，从而给进一步研究带来问题。所以，在弱互模拟的基础上，又给出了一个新的同余的互模拟关系，称为观察同余。

定义 2.3　称 P 和 Q 是观察同余的，记为 $P\simeq Q$，如果

若 $P \xrightarrow{a} P'$，则存在 Q' 使得 $Q\overset{\hat{a}}{\Rightarrow}Q'$ 且 $P'\approx Q'$。

对 Q 也有同样要求。

观察同余因为既具有外部观察者的视角(希望忽略 $\mathscr{T}$ 动作)，又具有良好的数学性质，成为了最重要的互模拟关系。对弱互模拟和观察同余有兴趣的读者请参考文[2]。R. Milner 的著作“Communication and Comcurrency”[2]是关于 CCS 的一部经典著作，我们强烈推荐有兴趣的读者阅读。

2.2　传值 CCS

CCS 最初是作为描述“通信”进程的语言而提出的，“通信”指进程间通过通道传送数据以实现交互和合作。为了数学处理的方便，进程间的数据传送被简化为单纯的通过共享通道同步。传值 CCS 就加入了数据值(value)的概念，每次通信都进行值的传递。我们首先假定所有的值都属于一个集合 V。

在给出完整的语法之前，让我们先看一个例子：

$$C \overset{def}{=} in?\ x.C'(x)$$

$$C'(x) \overset{def}{=} out!\ x.C$$

其中 x 是数据变元，C 可以看作是容量为 1 的一个缓冲区：它可以从信道 in 接收任意一个值，比如 q，然后进入 $C'(q)$ 状态，这时，它的容量已满，不能再输入，只能将 q 从信道 out 输出然后又回到 C。

完整的传值 CCS 的语法可以定义如下：

$$E::=a.E\mid\Sigma_{i\in I}E_i\mid E_1\mid E_2\mid E\backslash L\mid E[f]\mid \text{if } b \text{ then } E\mid A(x_1,\cdots,x_n)$$

$$a::=a?\ x\mid a!\ e\mid\mathscr{T}$$

其中，if b then E 常写作 $b\rightarrow E$。结合刚才的例子，对这些语法项的意义不难有一个直观的认识。至于严格的定义，请参考文[2]和文[3]。

依照传统迁移语义，传值 CCS 的基迁移分为迟、早两种情况，并分别导致迟、早互模拟概念。例如考虑进程：

$$R_1: c?\ x.(x\geqslant 0\rightarrow P)+c?\ x.(x<0\rightarrow P)$$

$$R_2: c?\ x.P+c?\ x.0$$

其中 P 不是空进程 0。

早(early)观点认为：进程所执行的输入动作均是形如 $c?\ v(v\in V)$ 的基本动作，即在定义迁移语义时就对输入前缀 $c?\ x$ 进行实例化，故有 $c?\ x.t \xrightarrow{c?\ v} t[v/x]$。按此观点，$R_1$ 与 R_2 是互模拟等价的。

迟(late)观点认为：进程所执行的输入动作是更一般的、抽象的输入动作 $c?$ 或 $c?\ x$，即在定义迁移语义时不对输入前缀 $c?\ x$ 进行参数实例化，参数的实例化要推迟至真正需要的时候，即只在定义互模拟关系或需要进行数据传送时才进行输入参变量的实例化。按此观点，R_1 与 R_3 不是互模拟等价的。

关于传值 CCS 的迁移语义等请参考文[4]和文[5]。

2.3　CCS 的优点与局限

CCS 中的基本实体很少，在纯 CCS 中有两个实体：进程和动作；传值 CCS 加入了数据值。CCS 的操作子简洁明了，具有很好的代数性质。因此，CCS 具有很强的实用性，用它对并发系统建模的例子有很多。还有基于 CCS 的程序设计语言。

但是，CCS 中只能进行数据值的传递，这又使得它的表达能力受到限制。虽然它可以对很多并发系统很好地建模，但对一些情况，例如对传名式参数调用，它无法直观地表达。于是，需要对 CCS 进行扩充与改进，这就产生了 π-演算。

3　π-演算

π-演算[13]是 R. Milner 等人在 CCS 的基础上提出的传名

演算，可描述通信拓扑结构动态变化的分布式通信系统(例如移动电话系统)。它允许进程之间传送和接收通道名，并且可以引入和输出局部名。因此，π-演算不仅非常简洁，而且具有很强的表达能力。类型化 π-演算是一个新的研究方向，而不带类型的 π-演算也称为纯 π-演算。

3.1 纯 π-演算

名字是 π-演算中最基本的概念。与传值 CCS 不同，π-演算将数据值、数据变元、输入参变元以及传递数据的通道不加区别，一律作为名字来处理。直观上名字即对象的引用，一个进程所具有的自由名代表了该进程当前所拥有的关于其它进程的知识或与其它进程的联系。在 π-演算中，所有复杂数据类型以及基本函数，都可以通过编码定义，而不需要作为基本对象。

纯 π-演算又可以分为两种：一种是一元(monadic)π-演算，每个输入或输出动作只能输入或输出一个名字；另一种是多元(polyadic)π-演算，每个输入或输出动作可以输入或输出零个或多个名字。让 $a,b,x,y,\cdots$ 表示名字，t 表示进程，一元 π-演算的语法就可以由以下 BNF 给出：

$$t::=0\mid a.t\mid t+t\mid[x=y]t\mid(x)t\mid t\mid A(y_1,\cdots,y_n)$$

$$a::=\mathscr{T}\mid a(x)\mid\bar{a}x$$

以上的操作子大部分来自 CCS：0 是一个不能做任何动作的进程；$a.t$ 是动作前缀；+是不确定选择；$[x=y]t$ 对 x 和 y 进行比较，如果相等，它将成为进程 t，如果不等，它将成为 0；$(x)t$ 是名字限制，表示名字 x 是 t 的私有名字(局部名字)。每个标识符 A 都有一个定义方程 $A(x_1,\cdots,x_n)\overset{def}{=}t$。

与 CCS 一样，$\mathscr{T}$ 表示了进程间的通信。输入前缀 $a(x).t$ 可以从信道 a(a 本身也是名字)接收一个名字，然后成为进程 t(将 t 中的 x 换名为刚接收的名字)。输出前缀 $\bar{a}x.t$ 可以从 a 发送名字 x 出去，然后成为进程 t。

在动作前缀 $a.t$ 和名字限制 $(x)t$ 中，x 在 t 的范围内是受限的(bound)。t 的自由名字集 $fn(t)$ 包含了那些在 t 中、但是却没有受限的名字。在自由名字集中的名字称为自由名字。自由名字在进程中称为自由出现的名字。

关于多元 π-演算，请参考文[6]。

作为一种传名演算，如果不加限制，π-演算可以获得两种移动性(mobility)：基于名字的移动性(一阶)和基于进程的移动性(高阶)。但是，π-演算进程间所能传送的对象被规定只能是通道名，而不能为进程，这表明 π-演算是一阶的。

这样做一部分是为了数学处理上的简单性，一部分是因为利用通道名作为指向进程的指针，很容易将高阶进程翻译到 π-演算中去。在图 2 中，给出了一个例子，说明用 π-演算可以编码高阶进程：

一个高阶进程	用一阶的 π-演算表示
$\bar{a}R.0\mid a(x).x$	$(q)(\bar{a}q.q.R)\mid a(x).\bar{x}.0$
↓	↓
R	$(q)(q.R)\mid\bar{q}.0$
	↓
	$(q)R$

图 2 π-演算编码高阶进程

在上面的例子中，$q.R$ 和 $\bar{q}.0$ 是一种简化，表示被传输的名字在这里并不重要，可以忽略；q 是由我们选取的不在 R 中出现的新名字，所以 $(q)R$ 可以归纳为 R。

π-演算的迁移语义(如图 3 所示)比 CCS 要复杂，这是它获得移动性的代价。

由于在 π-演算中抹杀了常量名和变量名、数据名和通道名之间的区别，为 π-演算引入基于互模拟概念的行为等价语义理论时将出现新的问题。直接根据强互模拟的基本定义得到的关系(称为基互模拟，记为$\dot{\sim}$)只是一个等价关系，而不是一个同余关系，它不能被名字替换所保持，因而也不能被输入动作所保持。例如考虑如下两个进程：

$$b(x).P\mid\bar{c}y.Q\dot{\sim}b(x).(P\mid\bar{c}y.Q)+\bar{c}y.(b(x).P\mid Q)$$

其中 Q 不含 x 的自由出现，且 x 不同于 y 和 c。若将所有自由名看成常量，即不同的自由名不能等同，则上式的左右两边显然是互模拟的。但 $(b(x).P\mid\bar{c}y.Q)[c/b]$ 与 $(b(x).(P\mid\bar{c}y.Q)+\bar{c}y.(b(x).P\mid Q))[c/b]$ 不是互模拟的，因为前者有 $\mathscr{T}$-迁移而后者没有。

很自然地，π-演算的互模拟同余关系(本文简称为互模拟)～可定义为能被所有名字替换保持的基互模拟关系，即：$P\sim Q$ 当且仅当对任意名字替换 σ，均有 $P\sigma\sim Q\sigma$。此时所有的自由名都看成变量。互模拟关系的这种二步定义法是 π-演算的一个有趣的现象和重要的特色。

$$\text{pre}\ \frac{}{a.P\xrightarrow{a}P}\qquad \text{match}\ \frac{P\xrightarrow{a}P'}{[x=x]P\xrightarrow{a}P'}$$

$$\text{par}\ \frac{P\xrightarrow{a}P'}{P\mid Q\xrightarrow{a}P'\mid Q}\ bn(a)\cap fn(Q)=\varnothing\qquad \text{com}\ \frac{P\xrightarrow{a(x)}P',Q\xrightarrow{\bar{a}y}Q'}{P\mid Q\xrightarrow{\mathscr{T}}(P'[y/x]\mid Q')}$$

$$\text{res}\ \frac{P\xrightarrow{a}P'}{(x)P\xrightarrow{a}(x)P'}\ x\notin n(a)\qquad \text{sum}\ \frac{P\xrightarrow{a}P'}{P+Q\xrightarrow{a}P'}$$

$$\text{open}\ \frac{P\xrightarrow{\bar{a}x}P'}{(x)P\xrightarrow{a(x)}P'}\ x\neq a\qquad \text{close}\ \frac{P\xrightarrow{a(x)}P',Q\xrightarrow{\bar{a}(x)}Q'}{P\mid Q\xrightarrow{\mathscr{T}}(x)(P'\mid Q')}$$

$$\text{id}\ \frac{P[\bar{y}/\bar{x}]\xrightarrow{a}P'}{A(\bar{y})\xrightarrow{a}P'}\ A(\bar{x})\Leftarrow P$$

图 3 π-演算的迁移语义

当然，与传值 CCS 类似，根据对输入动作的参数名的不同实例化策略，π-演算进程之间的基互模拟关系和相应的互模拟关系均有迟、早之分。

除迟互模拟和早互模拟外，π-演算还有一类重要的互模拟关系—开互模拟[7]。开互模拟把名字替换(蕴涵输入参数名的实例化)直接嵌入到互模拟的定义中，将上述互模拟定义中的两步合成一步，直接得到同余关系。开互模拟是一种比迟互模拟更精细(finer)的互模拟关系。

π-演算是对应用的广度(理论的一般性)和理论的完美同时追求的结果，通过较高抽象级来获得二者的统一。就概念简洁性和一般性而言，π-演算类似于作为函数计算基础的 λ-演算，因而可作为研究并发通信系统的基本工具。关于 π-演算的语法和语义理论参见文[2]，[8]和[5]。

3.2 类型化 π-演算

类型在编程语言中的作用众所周知。在顺序计算中，类型的研究已经很成熟。在并发计算中，类型的研究还是一个新的领域。

我们并不想给出严格的定义，只是介绍一下简单类型化的 π-演算：

名字 $a,b,\cdots,x,y,\cdots$

值

$v ::= a|n|true|false$ 名字，整数，布尔值

进程

$P ::= v(x).P$ 输入前缀

$|\bar{v}v.P$ 输出前缀

$|\cdots$

类型

$T ::= \#T$ 信道类型

$|\text{int}|\text{bool}$ 整数和布尔值

类型环境

$\Gamma ::= \Gamma, a:T|\varnothing$

我们用两个例子来对以上定义进行说明：

$\Gamma \vdash v:T$

表示在类型赋值 Γ 中，v 的类型是 T；

$\Gamma \vdash P$

表示进程 P 与类型赋值 Γ 是一致的。

下面是一些类型规则：

values: $\dfrac{\Gamma(v)=T}{\Gamma \vdash v:T}$

processes: $\dfrac{\Gamma \vdash P, \Gamma \vdash Q}{\Gamma \vdash P|Q}$

$$\frac{\Gamma \vdash w:\#T, \Gamma \vdash v:T, \Gamma \vdash P}{\Gamma \vdash \bar{w}v.P} \qquad \frac{\Gamma \vdash w:\#T, \Gamma, y:T \vdash P}{\Gamma \vdash w(y).P}$$

类型的研究是一个新的领域，有兴趣的读者请参考文[9]和文[10]。

结束语 随着计算机技术和网络通信技术的高速发展，以并发性、分布性、实时性、异构性和互操作性等为主要特征的并发分布式系统已成为当前计算机技术的主流方向，并已得到广泛应用。由于并发分布式系统本身非常复杂，因此其开发过程不仅难度大、效率低、周期长，而且很难避免和发现其中隐含的错误和缺陷。对于安全攸关系统(safety critical systems)，其正确性就更为重要。

形式化方法被公认为是一种行之有效的减少设计错误、提高系统可信度的重要途径。传统的形式化方法以严格的数学理论为支撑，对顺序计算的本质作了深刻的诠释。并发已成为新一代计算范型的本质特征，对并发计算的研究引起了许多学者的兴趣并取得了巨大的成果。

以通信系统演算 CCS[2] 为代表的进程代数方法，因概念简洁，可用的数学工具丰富，在并发系统的规范、分析、设计和验证等方面获得了广泛应用。

CCS 之所以能成功地应用于并发系统建模和程序验证，除了强大的描述能力之外，主要得益于基于互模拟(bisimulation)概念的行为等价语义理论的建立(D. Park[1])和完善(R. Milner[2])，它提供了一种强有力的手段和方法，能很好地帮助人们分析和验证用它描述的并发系统。CCS 的提出者 R. Milner 本人也因其在 LCF、ML 和 CCS 三个方面开创性和奠基性的工作而荣获 1991 年图灵奖。

π-演算是在 CCS 基础上发展起来的，它具有更强大的表达能力，而其本身的实体却比 CCS 更少。对 π-演算的研究取得了许多重要成果：在编程语言方面有 Pict 和 ML2000 把 π-演算作为基础；在验证工具方面有自动的验证工具 MWB[11] 和交互式验证工具 PiM；在安全性领域有人在 π-演算基础上提出了 Spi-演算[12]；还有关于类型化 π-演算的研究[10]，等等。

参 考 文 献

1 Park D. Concurrency on automata and infinite sequences. In: P. Deussen, ed. Proc. of Conf. on Theoretical Computer Science, volume 104 of Lecture Notes in Computer Science. Springer-Verlag, 1981

2 Milner R. Communication and Concurrency. Prentice-Hall, 1989

3 Li Z, Chen H. Computing strong/weak bisimulation equivalences and observation congruence for value-passing processes. In: Proc. of TACAS'99, volume 1579 of Lecture Notes in Computer Science, Springer-Verlag, 1998. 300～314

4 Hennessy M, Lin H. Proof systems for message-passing process algebra. In: E. Best, ed. Proceedings of CONCUR'93, volume 715 of Lecture Notes in Computer Science, Springer-Verlag, 1993. 202～216

5 Li Zhoujun. The Verification Theory and Algorithms for the Value-Passing CCS and π-Calculus: [PhD thesis]. National University of Defence Technology, 1999

6 Milner R. The polyadic π-calculus: a tutorial: [Technical Report ECS-LFCS-91-180]. Oct. 1991

7 Sangiorgi D. A theory of bisimulation for the π-calculus. Acta Information, 1996, 33: 69～97

8 Li Z, Chen H. Checking strong/weak bisimulation equivalences and observation congruence for the π-calculus. In: Proc. of ICALP'98, volume 1443 of Lecture Notes in Computer Science, Springer-Verlag, 1998. 707～718

9 Pierce B C, Sangiorgi D. Typing and subtyping for mobile processes. Journal of Mathematical Structures in Computer Science, 1996, 6(5): 409～453

10 Sangiorgi D, Walker D. The pi-calculus: a theory of mobile processes. Cambridge University Press, 2001

11 Victor B, Moller F. The mobiligy workbench—a tool for the π-calculus. In: Proc. of CAV'94, volume 818 of Lecture Notes in Computer Science, Springer-Verlag, 1994. 428～440

12 Abadi M, Gordon A D. A calculus for crytographic protocol: The spi calculus. In: Proc. of the Fourth ACM Conf. on Computer and Communications Security. ACM press, April 1997

13 Milner R. Parrow J, Walker D. A calculus of mobile processes, part i and ii. Information and Computation, 1992, 100: 1～77

14 Li Z, Chen H. Symbolic transition graph and its early bisimulations checking algorithms for the π-calculus. Science in China(Series E), 1999, 42(4): 342～353