计算机科学1993 Vol.20№.4

面向对象并发程序设计的基础理论

胡国定 王永革 （南开大学数学研究所 天津300071）

摘 要

We give a description of the basic model, Actor Model, for concurrent object-oriented paradigm. By comparing it with other models of concurrency such as CSP, FP, GHCL etc., we make a discussion of this model in the following aspects: concurrent laws, semantic systems and characteristics of Actor languages, i. e., history-sensitive behavior, joint continuation, delayed evaluation and inherent concurrency. In the last part of the paper, we discuss the reason why Actor system can be regarded as the theoretical model for multicomputer system.

1. 引言

面向对象是一种程序设计方法学，面向对象的程序设计风范受到了国内外计算机专家的密切关注，各种面向对象的语言也应运而生。为了探讨面向对象的并发理论，对面向对象的基础理论进行研究势在必行，本文介绍的演员系统（Actor System）就是作为面向对象的基本模型而提出来的。

我们的目标是建立一种并发模型，它将在并发结构中起到图灵机在冯·诺依曼体系结构中所起的作用。下面，先回顾一下各种计算模型，比较它们可作为通用并发计算模型的可能性。

1.1 通信顺序进程（CSP）

CSP[1]是Tony Hoare和其同事们为解决顺序进程的通信问题而提出的。在这个模型中通信是同步的，进程的拓扑结构是静态的。它有二个缺点：第一，无法定义递归程序：一个忙于发送消息的进程不能接受消息。第二，不提供重构形(reconfiguration)功能，无法进行动态扩展，特别地，该语言不适于表达并发系统结构的需求，因为将一个进程从一个处理器移到另一个处理器时需要具有重构形的表达功能。

1.2 函数程序设计

自J.Backus[7]以来，函数程序理论得到迅速发展，它在并发理论中具有很重要的地位并具有其他模型所无法比拟的优点。函数程序设计中的并发来源于如下事实：函数的所有变元可并发求值。函数程序设计的另一重大贡献在于它指出了赋值指令所产生的很多困难与不便，一般来说，分析一个带有赋值指令的并发系统需要考虑每个过程中的指数多次指令交错。由于函数程序满足合流性（Church-Rosser性质），所以一个程序中表达式的求值顺序是无关的，但也有一个困难困扰着函数程序设计，那便是函数程序缺乏“历史”[7]。

函数程序具有延迟求值的特点，但在现实的分布式计算中这并不能很好地工作。数据库的最大特点是数据共享，而函数值却仅仅被一个用户所使用（除非第二次激活该函数）。考虑另一问题，为了实现并发系统，程序模型需要具有定义可共享的有历史敏感行为（history-sensitive behavior）的计算实体的功能。现代某些函数语言，如Haskell提供了合并（merging）通信。但很可惜这

胡国定 教授，博士导师，从事信息科学与理论计算机科学研究。王永革 博士生，从事并发理论及理论计算机科学研究。收到日期：92-11-24。

已超出了λ演算所提供的语义系统。所以为了保证语言的函数特性，一般都限制合并的使用，例如在数据流语言中。由于实现同步是一个系统是否支持并发处理的基本条件，所以程序模型应与创建可修改的共享结构的能力有机地结合起来。

1.3 选择提交卫式Horn子句语言(GHCL)

并发逻辑语言作为一种颇具特色的语言已出现很久，有很多变种，如Parlog，并发Prolog等。这类语言我们统记为GHCLs (Guarded Horn Clause Languages)。

GHCL中的语言都以一阶逻辑作为基础，但Hewitt与Agha在[11]中指出逻辑并不能充分地充当GHCL中语言的程序行为模型。这是因为程序的动态性质不能从程序的逻辑阅读中推出，这些动态性质导致了从源程序中无法判定的推理，关于这方面的例子请参阅[11]。GHCL中的共享变元是一阶对象，变元的一致化使得系统可以重构形，但如果将一致化作为原语来使用，那将带来极大的低效性；而如果限制一致化的使用，那么基语言将显得不必要的复杂。

1.4 基于对象的程序设计

有两种观点来看待我们所处的世界。一是我们的世界由很多被动的对象组成，函数能够改变它们的行为。在某些层次上，这是函数程序设计风格所采用的观点。二是用演员模型所采用的基于对象的观点来看待世界。这种观点最先由Kristen Nygaard和Ole-Johann Dahl在其Simula中提出，后来被Xerox PARC的研究者们所青睐，并开发出Samlltalk。这种观点认为组成世界的对象是主动的，函数（方法）存在于对象中，并且仅能作用于包含它的对象。

Actor语言不仅保留了函数程序设计的一切优点，而且使得传统的基于对象的概念与函数程序的概念得到统一。Actor语言与其它基于对象的语言的主要差别在于前者是固有并发的。下边列表作一比较：

程序模型	计算原语	交互模式
顺序进程	进程，状态	同步消息
函数程序	表达式，代入	函数激活
GHCL	子句，一致化	变元共享
基于对象的程序设计	对象，变迁系统	异步消息

在蕴含并行的分布式系统中，一个重要的法则是时间依赖于具体观测者，这条基本法则也是现代物理学的基本法则。事件的时态顺序仅仅有两条不变因素：第一，在某一空间中到达同一地点的事件之间必须有一个确定的顺序。第二，若一事件引发另一事件，则前者一定在时间上先于后者。上述二因素独立于具体观测者而存在，这一点在演员模型中得到了具体体现。

2. 演员模型

2.1 演员是什么？

模型化一个对象的通用语义方法是将对象的行为视为输入通信的函数，这也就是演员模型的观点。演员是计算系统中一种自包含的、交互式的、相互独立的实体，演员之间用异步消息来通信。

每一个演员都包含一个邮递地址和一个行为，演员仅能通过向别的演员发送信息来影响另一演员的行为，而且演员α能够向演员β发送消息当且仅当α知道β的邮递地址。当多个演员同时向同一演员发送消息时该怎么办呢？解决这个问题的一种办法是提供缓冲区，一般地假定一个演员具有一个无界的缓冲区，就如同下推自动机的栈无界一样。另外假定演员的消息能够最终被传送（guarantee of delivery），这样就存在一个演员系统，它的行为无法用一非确定图灵机来模拟。

2.2 事件图

在演员模型中，有两种对象（即演员和通信）需要被表示。在最抽象级上，演员的计算可表示为事件之间的关系。这里一个事件e可认为是某个演员α对某一通信的处理，

其中α称为e的目标，事件之间的顺序有两种：

1．到达顺序：到达同一目标α的事件按其得到处理的先后确定了其间的顺序关系。

2．激活顺序：若一事件引起另一通信的发送，则这一通信得到处理而产生的事件被认为是由前一事件激发的，从而形成了事件之间的激活顺序。

激活顺序和到达顺序集合的传递闭包称为事件上的联合顺序（combined order）。

演员模型的并发计算可借助事件图来描述（图1），这些图用以模拟演员的行为。图中每一条垂线表示一个演员所接受处理的所有通信，并称之为生活线（lifeline）。一种事件是通信的接受，另一种事件是新演员的创建，图中用生活线上加开圆弧来表示新演员的创建这种事件。生活线间的连接表示事件间的激活顺序，挂起事件表示已发送但尚未得到处理的通信，图中用矩形小块表示。

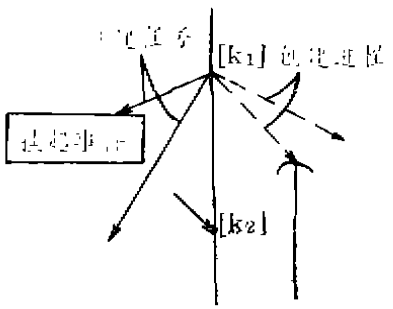

图1 事件图

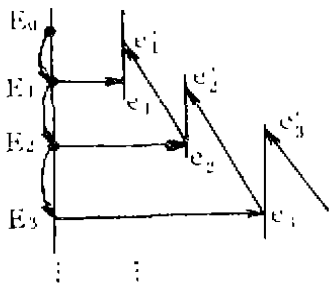

图2 不可行的事件图

2.3 并发计算法则

并不是所有的事件图都表示可行的计算，如图2中在事件E_0于e之间有无穷多个事件发生，这在现实计算中是不可行的，因此我们需要一些法则来保证事件图在计算上的可行性。演员系统中有两类法则：顺序法则与局部法则。

2.3.1 顺序法则

A．引发性：在联合顺序中任一事件不能先于它本身；

B．可数性：最多有可数个事件；

C．离散性：任二事件间最多有有穷个事件。

由于演员系统在概念上是分布式的，所以它没有全局时钟，到达顺序表示了每一个演员的局部时钟所记录的事件发生顺序。

函数g:E→I（这里E为事件集。I为整数集）称为全局时钟，是指g确定了E上的一个全序，并保持了原来的联合顺序。

我们可以证明事件图满足以上三条法则A,B,C当且仅当其事件上存在一个全局时钟g，当然这个全局时钟并不一定是唯一的。

2.3.2 局部性法则 在分布式系统中，每个演员都有自己所私有的信息。为了在演员系统中实现数据封装，我们需要引进局部性法则。局部性法则允许演员α_1向演员α_2发送消息当且仅当α_1知道α_2的邮递地址。我们称α_1是α_2的相识（acquaintance），是指α_2知道α_1的邮递地址。

A．有穷参与法则：一个事件的参与者在数目上是有穷的。

B．相识法则：一个新创建的演员的相识必须是该事件中的参与者或是由同一事件所创建的演员。

C．有效通信目标：一个演员收到消息之后，它仅可以向事件的参与者或向该事件所新创建的演员发送消息。

D．相识演变（Acquaintance Evolution）：在一个事件中，一个演员的相识仅可以是该演员生活线上前边事件的参与者或是由前边事件所创建的演员。

2.4 公平性

为了实现并发计算的同步，我们需要演员在接到通信后具有非确定的合并功能，这就可能使到达演员α的某些通信永远得不到

处理，因而也就产生了公平性问题。在演员模型中我们要求演员在处理通信时是公平的，任一通信不可能被无限延迟。换句话说，演员模型中要有保证通信最终传递的功能，即一个演员所发送的消息将在有限的时间之后达到目标，且对消息到达的次序不作任何假设和限制（这类似于邮政系统而非电话系统），这样也就导致了某些演员系统的行为无法用非确定的图灵机模拟。

2.5 演员系统的指称语义

在程序语言的语义学中，通过将程序映射到数学对象而给程序赋予语义。对于确定的顺序程序来讲，其操作语义被解释为程序的执行过程。在非确定语言中，其操作语义通过树来表示。与操作语义相对照，指称语义通过先确定指称物，然后给出语言成份至指称物的映象来确定语言的语义。

在并发系统中，程序是非确定性的，我们借助于幂论域理论来给出指称语义。其直觉是，每一个非确定函数可被描述为一确定函数，它将论域中一个元素映为论域的一个子集。

在演员系统中，用事件图的集合作为其基本论域，对于二事件e_1，e_2，其上顺序关系定义为$e_1<e_2$当且仅当e_1是e_2的一个前段历史，即e_2为e_1的一个可能扩展。这里还需定义一个辅助函数f^1_*，它将论域的子集映射为论域的另一个子集，确切地说，如果$f^1_*(A)=B$，则$\forall y\in B$，$\exists x\in A$，使得y为x的某一扩充，即挂起事件的执行。

有了上述的准备，利用前节所定义的法则，我们便可以研究演员系统的指称语义，细节请参阅[8]。

2.6 变迁系统（Transition System）

演员系统的格局包含两项内容：演员和任务，这里任务指尚未得到处理的通信。我们先引进如下定义：

定义一 任务。所有可能任务集$\mathcal{T}$是$\mathcal{T}=\mathcal{S}\times\mathcal{M}\times\mathcal{K}$。这里$\mathcal{S}$是所有可能的标签集，$\mathcal{M}$是所有可能的邮递地址集。$\mathcal{K}$是所有可能的通信集。若$\tau=(t, m, k)\in\mathcal{T}$，则t为τ的标签，m为τ的目标。

上述定义中使用标签的主要目的是为了区分发自同一演员并且到达同一演员的各个任务。为方便起见，我们利用字符串来表示标签，在联合顺序中，若e_1在e_2的前边，我们则规定e_1的标签是e_2的标签的前缀。

定义二 局部状态函数。映射l称为局部状态函数，如果$l: M\to\mathcal{B}$，这里M是$\mathcal{M}$的有穷子集，$\mathcal{B}$是所有可能行为的集合。

定义三 格局。演员系统的一个格局为二元组(l, T)。这里l为局部状态函数，T为有穷的任务集。且

1．T中任一任务的标签都不是T中任务或l定义域中某一邮递地址的前缀。

2．l定义域中的任一邮递地址不是l定义域中另一邮递地址或T中某一标签的前缀。

定义四 演员。所有可能的演员集合$\mathcal{A}=\mathcal{M}\times\mathcal{B}$，这里$\mathcal{M}$，$\mathcal{B}$的含义前边已给出。

定义五 行为。以m为邮递地址的演员的行为是$\mathcal{B}$的一个元素：

$\mathcal{B}=(\mathcal{S}\times\{m\}\times\mathcal{K}\to F_c(\mathcal{T})\times F_c(\mathcal{A})\times\mathcal{A})$

这里$F_c(\mathcal{T})$是$\mathcal{T}$的所有有穷子集的集合，$F_c(\mathcal{A})$是$\mathcal{A}$的所有有穷子集的集合。假如β是以m为邮递地址的演员α的行为，t为α的标签，k为被处理任务的通信，且$\beta(t, m, k)=(T, A, \gamma)$，并设$T=\{T_1, \cdots, T_n\}$，$A=\{\alpha_1, \cdots, \alpha_{n'}\}$，则如下条件成立：

1．被处理任务的标签t是所有新创建演员或任务的标签的前缀，即

$\forall i(1\leqslant i\leqslant n\Longrightarrow\exists m_i\in\mathcal{M}\ \exists k_i\in\mathcal{K}\ \exists t'_i\in\mathcal{S}\ (\tau_i=(t.t'_i, m_i, k_i)))$ $\forall i(1\leqslant i\leqslant n'\Longrightarrow\exists\beta_i\in\mathcal{B}\ \exists t'_i\in\mathcal{S}(\alpha_i=(t.t'_i,\beta_i)))$

2．设I为新创建任务标签的集合，M为新创建演员的邮递地址集，则$I\cup M$中任一元素不是另一元素的前缀。

3．总有一个替换行为，即：$\exists\beta'\in\mathcal{B}$ $(\gamma=(m,\beta'))$.

有了以上的准备，我们便可以研究演员模型的变迁系统及操作语义。例如可以形式

地表示从一个格局可以变迁到另一个格局的条件以及形式地表示上节所提到的消息传递的保证性等等，详细内容请参阅[3]。

2.7 Actor语言的结构

这一节我们主要介绍Actor语言在控制结构、历史敏感性、连接延拓、延迟求值及固有并发等方面的内容。Actor语言的原语有三种：1. creat：创建一个新的演员。2. send to：向另一个演员发送消息。3. become：完成一个任务后，用新的行为代替旧行为。

这些非常简单的原语具有非常强的功能，能够用以建立高层的抽象并建立并发程序设计风格。Actor的create原语在并发计算中的作用等同于函数抽象在顺序进程中的作用，它扩充了并发计算中函数抽象所提供的动态资源创建能力。become原语给予演员以历史敏感行为，这是可变的数据对象所必须具备的，从而使演员克服了本文开头提到的函数程序语言的不足。send to原语类似于函数应用，是异步通信的手段，它将消息放入另一演员的邮政信箱（消息流），应注意每一个演员的邮政信箱的地址(即邮递地址)是在该演员被创建的时刻所唯一确定了的。

每当演员处理完一个通信后，它便计算出它以后的行为，在纯函数程序设计中，一个函数的行为是不变的，即：替换行为＝原行为。

2.7.1 控制结构 注意到我们对演员行为做定义时，未用到递归、重复或其它循环算子。但是演员系统仍可以实现任意的控制计算，其原因是：控制结构是由特定的消息传送模式来实现的。例如对于求自然数的阶乘，可先创建一个阶乘演员：factorial和一个顾客演员：customer，顾客演员customer等候阶乘演员factorial的通信即计算结果。factorial的行为可描述如下。factorial接到消息n后，1. 创建一个演员，它的行为ψ(n, c) 是将n与下次接受到的整数相乘并将结果做为消息发送到以c为邮递地址的演员；2. 给自己发送一个求n−1的阶乘的消息。上述过程也可用图3来表示：

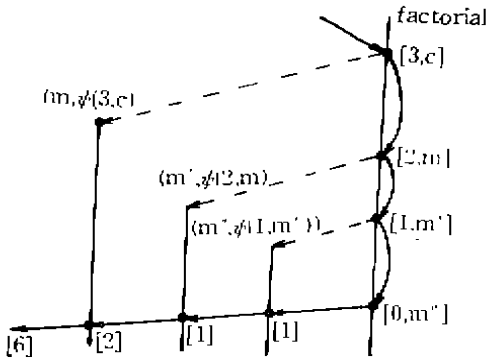

图3 用演员模型求阶乘

求阶乘的算法是一个串行性非常高的算法，因此上述过程并未显示出演员系统的优越性，它也许比传统程序中用堆栈的办法求阶乘还慢，这里我们只是举例说明在演员系统中是如何通过发送消息来实现控制的。

2.7.2 历史敏感性 它主要体现在：演员能够改变自己的行为，从而达到具有可以接受各种不同消息的行为。在Rosettle Actor语言中，用一个关键词mutable来实现这一点。例如一个银行账户演员可定义为：

```
(define BankAccount
  (mutable [balance]
 [withdraw-from [amount]
  (become BankAccount (−balance amou-
nt) )
  (return' withdrew amount) ]
[deposit-to [amount]
  (become BankAccount (+balance amou-
nt) )
  (return' deposited amount) ]
[balance-query
  (return' balance-is balance) ]) )
```

图4 银行帐户演员实例

上例中的become原语定义了替换行为。

2.7.3 连接延拓(Joint Continuation) 分治并发模式总可以方便地以函数程序设计的方式表现出来，即在一个函数中各变量并发求值，然后集结起来求最后结果。由于演员系统保持了函数程序设计的优点，分治并发模式在演员系统中也可以用非常简单的形式表现出来，图5就是用分治模式求一列数乘积的演员系统事件图（限于篇幅，这里我

们略去用演员原语写的程序）：

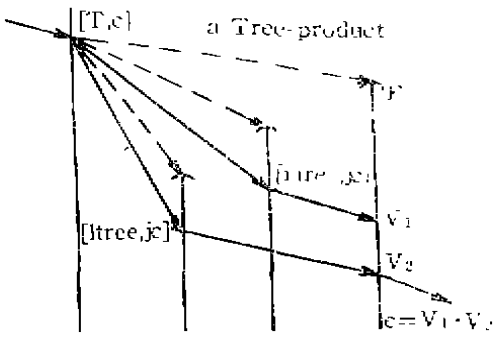

图5 求一列数乘积的演员系统事件图

其中假设这列数分布在一个树的叶上，演员Tree-product创建两个演员ltree与rtree分别用来求左边树叶上数的乘积与右边树叶上数的乘积，同时也创建演员joint-cont，它用来将两部分乘积最后乘到一起。这里joint-cont可能先收到由ltree发送来的值，也可能先收到由rtree发送来的值，其顺序并不重要。由于并发性要求Tree-product激活演员ltree与rtree的行为具有不确定性，因而joint-cont的行为无法用函数表达出来——尽管Tree-product本身的行为可用函数来表示。在这里使用joint-cont有很多优点，例如在joint-cont先收到值0后，它便向顾客发送结果0，而不再等待下一个消息。又如当joint-cont发现第一个输入消息（值）有异常现象时，便可立即激活异常处理演员，而不再等待下一消息。总之，有了演员joint-cont，便可对程序运行行为的正常与异常作出独立的判断并规定程序的下一行为。关于演员系统的延迟求值与固有并发限于篇幅我们不再讨论，读者请参阅[4]或[5]。

3．多机系统（multicomputer）

计算机专家已提出了多种并发计算机系统，它们大致可分为以下三种结构：同步计算机，存储共享计算机及多机系统。同步计算机如Connection Machine适于数据并行计算，但它们的目的性很强（special-purposed），且对并发作了多种限制，因而不具有通用性。存储共享计算机有多个处理器，并提供有全局共用的存储区，各个处理器也都有自己私用的超高速存储器，[9]的分析结果表明由于多个处理器要去访问同一存储区，所以就得对访问加上多种限制，最终导致并行度并不能随处理器个数的增加而成倍增长。

多机系统由大量可编程的小计算机（都有各自的存储器）组成，它们之间用消息传递网络连接。多机系统被用来支持演员模型[6]，其消息传递网络支持演员的邮递抽象，多机系统中存储器是分布式的且信息都归局部计算机所私有，保持通信局部性的过程可通过动态创建或删除小的对象得到简化，以上这些都是演员模型所具有的特征。

同时应注意，晶片计算机（transputer）也属多机系统，只不过它的理论模型是CSP，而不是演员模型。

4．结论

演员系统提供了构造并发系统的基本建筑模块，演员系统是固有并行的并提供了最大的并发性，异步通信表明演员不必等待消息接受者接受消息便可开始下一步工作，动态创建功能相当于分布式存储——允许部分工作分配给新创建的演员。演员的替换行为不仅使我们避免了低级结构上的赋值语句，也使我们不必担心资源的分配情况，同时也避免了函数模型中的一些缺点。

演员模型最引人注目的地方是使程序员从一些细节问题中解放出来，这些细节包括从何时或何地起实行并行等等一些问题，使程序员有可能直接去考虑算法本身的复杂性。

总之，我们确信演员模型提供了一种很好的编程抽象，其它一些语言模型要么在语义上有较强的限制，要么增加了一些并不能增加计算功能的额外构造，而且面向对象的设计风范可以建立在演员模型上并提供交互系统所需要的合理的方法。（参考文献见P29）

MS-1实现了两种调度算法：调度算法1是一种轮转法；调度算法2则是一种争用法，很适合成员系统在单机环境下的并发执行。

④解释部分：用于为每个成员的执行提供各种支持，包括匹配部分的解释执行、冲突消解及右手部动作的解释执行。

MS-1解释器的工作原理及过程如下：

①成员系统程序装入内存过程中进行编译，建立成员名索引表和每个成员的运行环境，其中包括对每个成员的产生式规则集建立左手部匹配网络Rete。

②在顶层上用start命令使该成员系统开始运行。

③按照一定的调度算法，选择一个可执行的成员。

④对该成员解释执行。

⑤若有通信动作，则通过通信原语与系统中其它成员通信，然后转向③。

⑥系统运行结束，可使用各种顶层命令。

通过运行典型例子（比如哲学家就餐问题）及开发一个实用研究程序（基于知识的空间站GaAs自动生产线协同故障诊断系统[28]），证明MS-1是一个比较成功的协同产生式系统程序设计语言。

五、结束语

迄今，在理论上比较完善并且在实践中获得广泛应用的人工智能程序设计语言首推以OPS5为代表的产生式系统。然而产生式系统一些固有的缺陷限制了它在复杂工程场合特别是实时处理、协同求解环境里的应用，使得一些人对产生式系统产生了怀疑，转而寻找其它问题求解的方法。可是为什么人工智能创始人Simon和Newell却将其做为人类认知心理学模拟人类思维的模型，并且有人已经通过心理学实验证实了产生式系统与人类思维方式相似呢？这说明产生式系统，或者确切地说，基于产生式系统的方法尚有许多潜力，这正是本文讨论目的之所在。述及的实时产生式系统方法、协同产生式系统方法代表了产生式系统的两个发展方向，也是挖掘产生式系统潜力，使其成为比较理想的人工智能理论模型及程序设计语言的诸多尝试之一。（参考文献共29篇略）

（接第页6）

参考文献

[1] Tony Hoare 著，周巢尘译，通信顺序进程，1989，北京大学出版社

[2] G.Agha,Supporting multiparadigm Programming on actor architectures. In Proceedings of Parallel Architectures and Languages Europe, Vol.Ⅰ,Espirit,Springer-Verlag, 1989

[3] G.Agha, Semantic considerations in the actor paradigm of concurrent computation. In Seminar on concurrency, Springer-Verlag, 1985

[4] G.Agha, The structure and semantics of actor languages. In Proceedings of the School/ Workshop on Foundations of Object-Oriented Languages. Springer-Verlag, 1992

[5] G.Agha, Concurrent object-oriented programming. CACM, 33(9), 1990

[6] W.Athas, Fine grain concurrent computations. Ph.D.dissertation, Computer Science Dept., California Institute of Technology, 1987. Also published as Tech. Rep. 5242:TR:87

[7] J. Backus, Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. CACM, 21(8), 1978

[8] W.D.Clinger,Foundations of Actor Semantics. AI-TR-633, MIT Artificial Intelligence Laboratory, May 1981

[9] W.Dally,A VLSI Architecture for Concurrent Data Structures. Kluwer Academic Press, 1986

[10] P.Henderson,Functional programming, applications and its implementation. Addison Wesley, Reading, MA., 1983

[11] C.Hewitt and G.Agha, Guarded Horn clause languages, are they deductive and logical. In Proceedings of Fifth Generation Computer Systems Conference, ICOT, Tokyo, Dec. 1988