Stream 和 Object

——面向对象的并行计算模型及其语言探讨(1)

Yoshida Kaul 和 近山隆

(日本新一代计算机技术开发研究机构，ICOT)

摘 要

AIUM从Stream计算(或半序集演算体系)出发，微观地研究计算和对象抽象化，是一种面向对象的并行程序设计语言。本文拟分为两部分；第一部分介绍面向对象的并行计算模型及其语言AIUM的基本文法和相关例子；第二部分（连载）进一步介绍AIUM的扩充文法和例子以及给并行语言带来了强的描述能力和处理手段的Stream和Object的融合。

1. 前 言

并行计算机系统已成为一个广泛研究的课题。尽管现有的大多数分布式系统采用顺序语言的改进版本进行程序设计，但各种各样的并发语言正得到广泛的使用。并行程序设计语言的研究愈来愈为人们所重视。本文介绍的AIUM是日本ICOT研制的并行推理机PIM核心语言的一个组成部分，它是一种利用消息流进行通信和同步的面向对象的并行语言。

本文从并行系统的微观分析出发，逐步揭示出并行、并行系统、并行计算模型以及并行程序设计的本质，从Stream的观点出发，以Stream的演算为基础研究并行计算和对象的抽象化，提出了面向对象的并行计算模型及其语言。

2. 计算模型

2.1 并行系统和半序集合

所谓并行系统是能独立产生一个以上事件的系统，而所谓的"独立"意味着无法规定这些事件之间的顺序关系。

例如，3个事件a、b、c构成的系统中，有这样的序关系：·a必须在b之后发生；·c和a、b是独立的。

一般，我们称任意元素间都能规定顺序关系的集合为全序关系或链(chain)，其中有一部分元素间不能规定顺序关系的集合，称为半序集合（poset)。

a、b、c分别对应集合A＝{a}，B＝{b}，C＝{c}，上述系统S的半序集合演算可以表示为：S＝(A⊕B)＋C。其中，⊕为序数和(ordinal sum)，＋为基数和(cardinal sum)，分别表示半序集合的两种不同的加法算子，它们的运算结果也是半序集合[Birkhoff 79]。也就是说，并行系统是事件的半序集，且可以作为半序集的演算体系来定义。

链仅仅由顺序和⊕构成，半序集能分解为链的基本和。这样，我们可以使用具体化了的Stream来自然地表示链。

首先，我们从身边的普通例子(如：水流(河)、电子流(电流)、车流(交通)等)出发，来表示Stream的具体形象。图1所示为河流。

河由水，而水由水粒即水滴组成，水流相连，最后形成流。从上流向下流看，支流

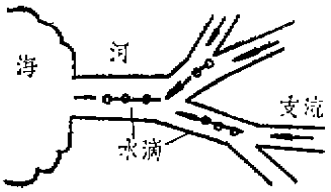

图1 河和海

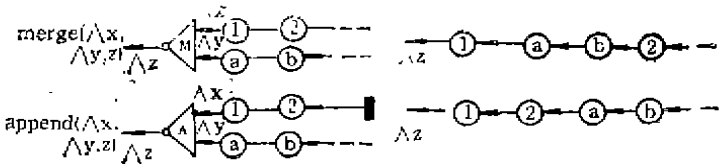

图4 Joint

合为主流，最后主流流入大海。这里，所谓上流和下流是表示流向，并不知道这一支流中的一粒水和另一支流中的一粒水谁先到达海洋。

各水滴流入海洋，改变着海洋的状态。

AIUM中，视河为处理的消息流，视海为对象，视水滴为消息。

2.2 Stream

Stream是消息的链，描述消息顺序关系，Stream之方向由输入端(inlet)和输出端(outlet)两个端点表示(图2)。其中，输入端用变量加前缀"∧"(如∧x)表示，输出端也用变量(如x)表示。

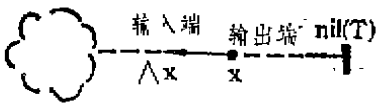

图2 Stream表示

可以这样来理解Stream的两端。我们知道与Stream有关的对象为：从Stream接收消息的对象和向Stream发送消息的对象。从对象的角度来看，按进来的端点或出去的端点来区分，称头部为输入端，尾部为输出端。

2.3 Stream的演算

Stream的生产和消费组成Stream的演算(图3)。

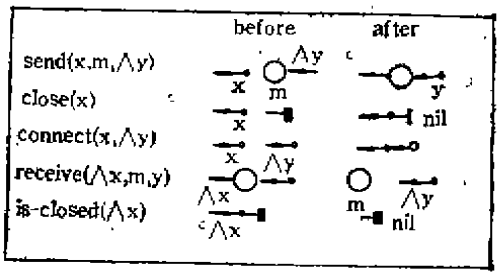

图3 基本演算

发送(send)：向输出端x发送消息m，与该输出端相连的Stream的输入端为∧y。

闭锁(close)：闭锁输出端x。这时，闭锁输出端后的状态称为空(nil)，表示该Stream已不能存取。

连接(connect)：连接输入端∧y到输出端x。这时，与输入端∧y连接的消息可连接到输出端x之前的位置。

接收(receive)：从输入端∧x接受消息m，与其(m)相连的Stream的输出端为y。

闭锁检验(is-closed)：检验输入端∧x闭锁的情况。

如上所述，Stream演算可以使用输入端、输出端以及空三种用语说明。

2.4 Joint

为了构造Stream树，我们定义称为Joint的两种Stream演算(图4)。

合并(merge)：分别将输入端∧x和∧y的消息以非确定的顺序合并，传到输出端z。其中，保持Stream∧x和Stream∧y的消息的顺序关系不变。

这些演算(公开，合并演算)，可用两种函数的复合g∘f表示。

函数f：chain × chain——poset

函数g：poset——chain

链一般化即为半序集合，则合并演算对应基本和，连结演算对应顺序和。

在Stream是链的情况下，称Joint演算构成的半序集为channel。

2.5 消息

消息(message)用消息名和Stream端点(输入端、输出端)参数表示。不带参数即仅有消息名的消息称为原子消息。带有参数

的消息称为合成消息。一个消息可由消息名、参数个数以及接收者看到的各参数的方向所标识。两个消息，即使消息名一致，参数的个数或方向不同，仍为不同的消息。

消息可以看成Stream的接续器。该接续器连接送信者作为实参传送的Stream端点，和接收者作为形参传送的Stream端点。两条Stream的连接，必须给出一方的输入端和另一方的输出端。所以，消息的发送者和接收者需要针对每个参数指定相反方向。

例如，图5的m(x, ∧y, z)有消息名m和两个输出端点x和z，一个输入端∧y三个参数。从而，其识别名为m（−，+，−）。消息的发送者如指定m(∧u,v,∧w)，则∧u连接x，v连接∧y，∧w连接z。

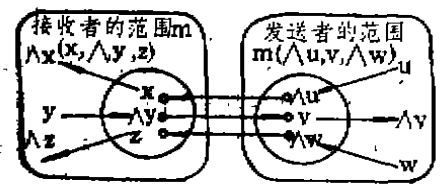

图5 连接Stream的消息

2.6 对象

对象抽象了重复的计算。一个对象，包括与外部接口的接口Stream和一条输入端，内部有表示对象状态的槽群。

从发送者角度看，传送到某对象的Stream（输出端）代表该对象。所谓了解某对象是指得到了该对象的Stream的输出端。所谓向对象B介绍对象A，是分解流到A的Stream，并将一端告诉B。

2.6.1 **对象的生成** 向类传送实例(instance)生成消息new(∧obj)，可生成该类的实例对象。这时，对刚生成的对象(第0代)，传送启动消息initiate，与此相连接的Stream的输入端为∧obj。启动消息用于对象的内部状态初始化。

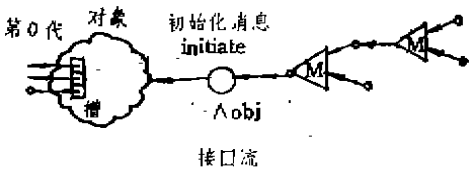

图6 对象的外观

2.7 世代 (generation)

对象以下述两阶段循环为一周期。我们称周期为对象的世代。

1。受动期：等待关于接口Stream的事件，即消息的接受或闭锁检验的阶段。

2。能动期：在观测到接口Stream的事件以后，实施对应观测事件的动作的阶段。这些动作包括：

·送信、闭锁、继续、合并、连结以及基底对象的生成动作*)。

·一个向下的世代下降。

各动作可以顺序或并行执行。

一世代是接口Stream的输入端和表示对象内部状态的Stream端所组成的Stream端点的集合体。

2.7.1 **方法 (method)** 一世代动作的定义称为方法。方法由两部分组成，一部分是关于观测事件的受动部分的描述，另一部分对应于观测事件相应动作的能动部分的描述。

2.7.2 **世代下降 (generation descending)** 某世代生成下一世代的动作称为世代下降。世代下降动作可以顺序或并行地和其它动作一起执行。

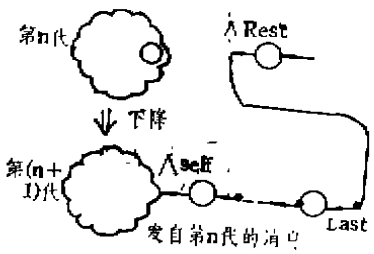

图7 世代下降

2.7.3 **自身 (self)** 作为某世代的对

*) 对象的生成，可分为类对象的生成和向该对象的消息传送。

象；自身意味着下一世代。向自身传递消息就是向联系下一世代的Stream输出端传送消息。

自身分解：分解自身，将其中之一的输出端与接收消息或传送消息的参数相连接，以便传送者或接收者在将来能够向该自身传送消息。

2.7.4 继续和终结 对象直到接收终结消息为止，才停止世代下降。

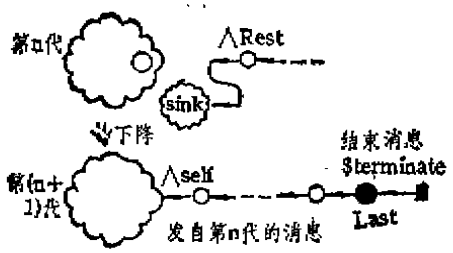

图8 对象的终结驱动

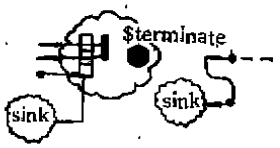

图9 Object的终结

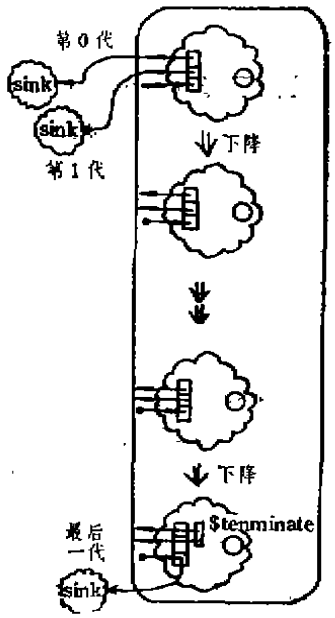

图10 世代链的对象

对象的终结：对象接收终结消息（$terminate）以后，将完整化它所具有的全部Stream端点。即输出端闭锁，输入端回收，接口Stream也同样回收。

从而，对象的一生可以表示为如下的世代链。

2.8 槽

对象内部具有表示其状态的槽群。各槽按名字联想并具有Stream端，输入端对应输入端槽，输出端对应输出端槽。在一个类中，槽名定义唯一。即各槽由类名和槽名标识。槽存取，是向对象自身即下一世代传送槽存取消息来实现的。

2.8.1 输入端槽

初始化：对象生成时，闭锁各输入端槽。

访问：向对象传送输入端槽访问消息get-inlet(Name,slot)，则当前的输入端与存取相关的输出端（消息的第二参数）连接。新的槽被闭锁。

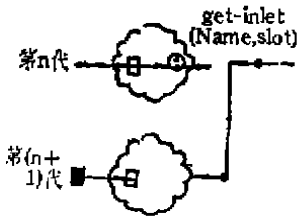

图11 输入端槽的访问

更新：向对象传送输入端槽更新消息set-inlet (Name,∧slot)，则将当前输入端与sink对象连接。另外，还要将存取相关的输入端（消息的第二参数）与新的槽连接。

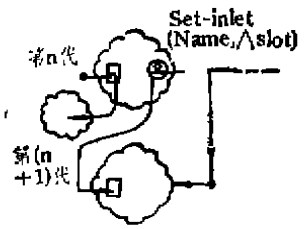

图12 输出端槽的更新

后期：对象终结时，各输入端槽与sink对象连接。

2.8.2 输出端槽

初始化：生成对象时，各输出端槽与sink对象连接。

访问：向对象传送输出端槽访问消息get-outlet (Name,∧slot)，则分解当前输出端的Stream，其中一部分与存取相关输入端（消息的第二参数）连接，另一部分与新的槽连接。

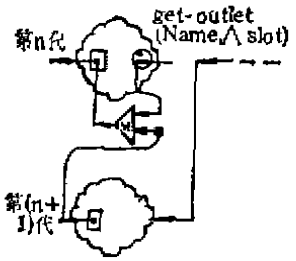

图13 输出端槽的访问

更新：向对象传送输出端槽更新消息set-outlet(Name,slot)，则用nil闭锁当前输出端。存取结果的输出端(消息的第二参数)与新的槽连接。

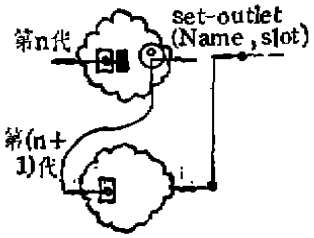

图14 输出端槽的更新

后期：对象终结时，闭锁各输出端槽。

2.9 原始对象

在AIUM中，Stream是基础，所有的对象都是以Stream为接口相互传递消息的。

一般，称原始数据的处理，如整数（例如，5），原子数据（例如，a），布尔值（例如，true），字符串（例如，"hi"）等为原始对象。它们也和其它抽象对象一样，是以Stream为接口的对象。类是最原始的对象。

例如，程序里指定整数5，生成整数对象5，并指示该对象的输出端。当向该输出端传送加法消息add (1∧sum)，则对象与适时接受该加法消息，生成加法结果6的整数对象，而且该对象的输出端为∧sum。对于被加数1和加法结果6，情况相同。可以为原始对象的操作提供简易描述，并且，它的操作也可以不使用Stream。

表及向量是由原始对象合成的抽象对象，同样也可提供简易描述。

2.10 Stream的完整化和计算终结

半序集合能分解为链。如果半序集合中，整体存在顺序关系下的最小元，各链存在最小上界（在顺序关系中，比链中所有元素都大的集合的最小元），则称为完整化的半序集合（CPO）。并且，我们称给出集合整体的最小元和给出各链的最小上界为半序集合的完整化。

当我们视最小元为集合的开始，集合本身为计算过程，最小上界为计算的结束时，可以给出如下解释：

不决定最小元，不开始计算；

不决定最小上界，不结束计算。

当Stream只出现在输入端或者输出端之一的情况下，会引起如下弊端。

输出端的闲置——因为某对象可能在继续等待接收其输入端的消息或闭锁检验，所以有可能死锁。

输入端的闲置——流入进来的消息群，谁也不接受，成为永久的垃圾（称为GC对象）。

同时，这些消息中，作为参数的Stream的输入端、输出端也被闲置。消息的发送者期待着消息的接收者将传递这些参数的输入端与某一对象相连，或期待着消息的接受者闭锁输出端。因而，仍可能发生死锁。

Stream计算中，确定各Stream的输入端和输出端的连接端点，称为Stream的完整化。没有输入端或者输出端的不完整的Stream的最上界和最下界可以按如下办法决定。

最上界空：被闲置的输出端闭锁。

最下界为sink对象：被闲置的输入端与sink对象连接。

2.10.1 **Sink对象** 所谓sink对象是具有解释消息的能力和完成回收消息作用的对象。sink对象的各世代为：

·接受消息时，对于Stream端点参数，在进行如下动作时，世代下降。

——若是输入端，则生成新的sink对象，并与之连接。称为输入端（或Stream）的回收。

——若是输出端，则闭锁。

·检查出闭锁时，结束。

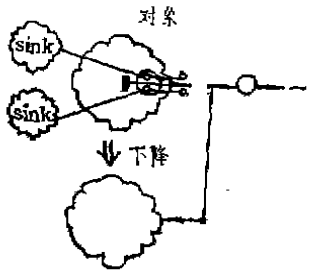

图15 sink对象

2.11 类

类是定义实例的类型对象，它由以下组成：

·应继承类群的定义，

·应保持输入端槽及输出端槽群的定义，

·定义对象世代的方法群的定义。

类是以Stream为接口的原始对象。接收实例生成消息，则根据该类型定义生成实例。

不存在类之类的元类概念。

2.12 继承

AIUM以程序代码量最小化为目的，支持多重继承。一个类可从任意多个类继承方法群。这种类继承扩大了对象能适用的方法群和能存取的槽群空间，但并不为对应继承的上层类分别生成实例。类继承不因为Stream计算而直接受影响。

2.12.1 **继承树** 类定义的类继承关系构成该类为根的树，称为类继承树。继承树以根为起点，按深度优先和从左到右的左优先的继承顺序，生成继承串。

例如，对于下述关系：1.类a继承b、c；2.类b继承p、q；3.类c继承r、s。继承串为a→b→p→q→c→r→s→。

这样，由用户明确定义生成的继承串，称为用户定义的继承串。该串两端分别加上上界类、下界类，即：下界类——用户定义继承串——上界类，便形成了完全继承串。即使对于明显不继承的类，也能生成这样的继承串：下界类——用户定义类——上界类。

2.12.2 **方法搜索** 对象根据受动期中观测的事件，从该完全继承串出发，搜索并确定能适用的方法后，进入能动期。可以指定从哪个类开始搜索；如不指定，则以下界类开始搜索。

上界类及下界类是定义"能适用于所有类的对象的公共方法群"的类，依据用户定义类，按可否再定义划分为相异的两类。

2.12.3 **上界类** 上界类是加在用户定义继承串最末尾的类。公共方法群中，一部分方法可根据用户定义类再定义。这些方法有：

·对应于初始化消息（initiate）的方法；

·定义由检测闭锁而结束的方法；

·定义处理"方法未定义错"的方法。

2.12.4 **下界类** 下界类是加在用户定义继承串最前面的类。公共方法群中，一部分方法不可由用户定义类再定义。这些方法有：

·槽存取方法；

·终结消息（$ terminate）的方法。

3. 基本方法

并行系统是事件的半序集合。所谓并行程序设计可以用描述事件的图表示。从而，所谓语言，其研究的要点在于"如何能容易

地描图”。图示也是一种语言。这里，我们定义一种符号语言。

对于Stream程序设计，与最大限度地实时回收无用单元和避免死锁问题联系在一起，Stream的完整化问题得必须加以考虑。AIUM为了能简单、安全、有效地编写程序，在定义函数式语法的同时，提供了各种支持。

3.1 类定义

类定义包括类名的定义，继承类的定义、输入端槽及输出端槽的定义和描画一世代的动作的方法的定义。

```
‹Class Definition› ::=
    Class ‹Class Name› "."
    [ ‹Super Class Definition› "." ]
    [ ‹Inlet Slot Defintion› "." ]
    [ ‹Outlet Slot Definition› "." ]
{ ‹Method› "." }
    end "."
‹Super Class Definition› ::=
    Super ‹Super Class Name› {", " ‹Super Class Name›}
    ‹Inlet Slot Definition› ::=
      In ‹Slot Name› {", " ‹Slot Name›}
    ‹Outlet Slot Definition› ::=
      Out ‹Slot Name› {", " ‹Slot Name›}
```

3.2 方法定义

方法由受动和能动两部分定义组成。受动部分定义应观测事件，能动部分定义对应于受动部分的应取动作。

```
‹Method› ::= ‹Event› "|" ‹Actions› {"," ‹Actions›}
‹Actions› ::= ‹Nil Expression›
```

3.3 Stream变量

Stream的方向可用下列Stream变量指定。

输入端变量：头部出现“∧”的变量名（如∧x）表示输入端。

输出端变量：仅仅出现变量（如x），表示输出端。

3.4 函数表达式

方法的受动与能动部分分别用函数表达式定义（表1）。

表1 基本表达式

关系	表达式	结果
receive(∧x,m,y)	':' ‹Message› ':=' ‹Out›:m=y	‹Nil›
is_closed(∧x)	::	‹Nil›
send(x,m,∧y)	‹Out›':' ‹Message› x:m	‹Out› Y
close(x)	‹Out›'::' x::	‹Nil›
merge(∧x,∧y,z)	‹Out›'=' ‹In› z=∧x	‹Out› Y
append (∧x,∧y,z)	‹Out›'\' ‹In› z\∧x	‹Out› ∧y
descend(∧x,s)	'⟸' ‹In› ⟸∧x	‹Nil›

各表达式的值只能为输入端或输出端或空，相应地可称之为输入表达式、输出表达式或空表达式。用这些表达式的组合可以描述复杂的图形。

例如，c:up:up:up:show(∧u)，是组合的的消息发送表达式。最前面的c:up是向输出端c传送消息up，接着，以Stream的输出端（称之为c_1）为其估价值。从而，该表达式改为c_1:up:up:show(∧u)。如此反复，整个表达式传送了消息show(∧u)之后，最后表示Stream的输出端。

3.4.1 “右→左”的原则 对于以前所示各图，消息都是从右到左流入，最后流到最左边的对象。也就是说，各消息都希望比它右边的消息更快为对象所接受。从对象接收消息的顺序来看，可以说时间上是从左到右。表达式遵从这个原则。因此，可以与描图类似地书写程序。

3.4.2 空的导出 为了防止由于输入端或输出端闲置的不完整Stream所导致的弊病，促使计算终止，基本文法给出以下规则：

规则1 （Stream变量成对出现）各个Stream变量必须以输入端变量和输出端变量

的变量对的形式出现。

规则2 （仅空表达式的出现）对于方法能动部分顶层的动作串（〈Actions〉），基本上只能指定空表达式。

如后所述，在扩充包含自动闭锁及自动回收支持的文法后，可以排除这些规则。

3.4.3 **原始对象** 原始对象的文字出现，意味着生成原始对象并具有到该对象的输出端。

3.4.4 **公共消息** 包括原始对象在内的所有对象都有几个应该接受的消息。对一般对象而言，要定义下界类或对应于下界类的方法。这里，我们介绍其中一个较有特征的消息。

"谁消息"：Who-are-youc (Who) 是找寻对象"所示image"的消息。

例如，向整数1传送消息：Who-are-you (Who)，则对于该输出端，整数1送出自己本身是1的消息后，闭锁。

所谓"所示image"，对对象来说，只要是能识别的，什么都行。

一般对象在接受"谁消息"时，返回以流向自身的Stream为参数的返回消息。即，问"你是谁？"，回答为"我是我"。

```
:Who-are-you (∧Who) =Rest|⇐∧Self
    Who:i-am (Selfz)::,
    Self=∧Selfz=∧Rest::
```

下面，我们用基本语法描述一个简单的例子——计数器程序。

应用例-1（计数器）

```
class counter
      out n
    :up=Rest  |⇐∧self,
      self:get-outlet(n, ∧N)
         :set-outlet(, n,N1) = ∧Rest::,
        N:add(1, ∧N1)::.
    :doun=Rest |⇐∧self,
      self:get-outlet(n, ∧N)
         :set-outlet(n, N1)=∧Rest::,
        N:sub(1, ∧N1)::.
    :set(∧N)=Rest| ⇐∧self,
    self:set-outlet (n, N)=∧Rest::.
    :show(N)=Rest | ⇐∧ self,
    self:get-outlet (n, ∧N )=∧Rest::.
    ::     | ⇐∧self,
        self:' $ terminate' ::
    end.
    class test
    :test=Rest=  | ⇐∧self,
      self:testM(∧Um,∧Dm):testA(∧Ua,
            ∧Da)
         :nop(∧Result)=∧Rest::
  Result\∧WUm\∧WDm\∧WUa\∧WDa::,
    Um:Who-are-you(WUm)::,
    Dm:Who-are-you(WDm)::,
    Ua:Who-are-you(WUa)::,
    Da:Who-are-you(WDa)::.
    :test M(U, D)=Rest | ⇐∧Rest,
    ##Counter:new(∧Counter)::,
    counter:set(5)=∧C::,
    C1:up:up:show(∧u)::,
    C2:down:down:show(∧D)::,
    C=∧C1=∧C2::.
    :testA (U, D)=Rest | ⇐∧Rest,
    ##counter:new(∧counter)::,
    counter:set(5)=∧C::,
    C1:up:up:show(∧U)::,
    C2:down:down:show(∧D)::,
    C\∧C1\∧C2::.
    :nop(Result)=Rest | ⇐∧Rest
      ##sink:new(∧Result)::
end.
```

图16 使用基本文法的计数器程序

计数器是利用消息up或down作计数器值加1或减1的对象。而且依据消息set或show，设置或访问计数值。对于该计数器，可作两种测试。即类test的方法testM和testA。这两种方法共同点为：

1. 生成计数器对象；
2. 首先，传送set(5)消息，设定计数

（转封底）

（接第30页）

器值为5之后，分解该输出端；

3．一方面，传送两个up消息之后，再传送消息show（∧U），访问计数器值；

4．另一方面，传送两个down消息之后，再传送消息show(∧D)，访问计数器值。

两个方法的不同点为：

·方法testM中，两个分支Stream(C_1和C_2)合并，消息up和down谁先到达，什么时候到达，是非确定的。从而，访问结果∧U表示{5，6，7}的某个值，∧D表示{3，4,5}的某个值。

·方法testA中，两个分支Stream(C_1和C_2)连接，从C_1来的两个up消息一定比从C_2来的两个down消息先到达，从而，访问结果∧U表示7，∧D表示5。

计数器对象的第一代刚接受消息，set(5)的能动期如图17所示。

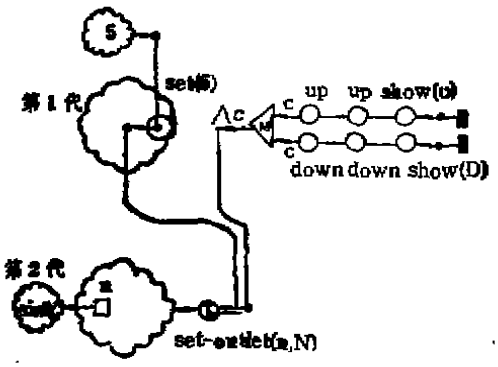

图17 计数器的第一世代

本程序可以根据下节所述的扩充文法改写为图18所示格式：

```
class  counter
        out n.
    :up→¦n+1=¦n.
        %x+y⇒s; x:add(y, ∧s)
    :down→¦n-1=¦n.
        %x-y⇒D; x:sub(y, ∧D)
    :set(∧N)→N=¦n.
    %¦n⇒$Self:set-outlet(n, N)
    :show(N)→¦n=∧N. %¦n⇒$self:
                get-outlet(n, ∧N)
end.
class test
  :test→:testM(∧Um,∧Dm):testA(∧Un,
    ∧Dn):nop(∧Result),
    Result$1=(Um? ),
      Result$2=(Dm? ),
  Result$3=(Un? ), Result$4=(Dn? ).
    :testM(U, D)→
    #counter:set(5)=∧C,
        C:up:up:show(∧U),
        C:down:down:show(∧D).
    :testA(U, D)→
    #counter:set(5)=∧C,
        C$1:up:up:show(∧U),
        C$2:down:down:show(∧D).
    :nop(Result)→.
end.
```

图18 使用扩充文法的计数器程序

（未完待续）

〔龚 红 何克清编译〕

计算机科学

第3期（双月刊）
1991年6月23日出版
国内统一刊号：CN51—1239
代号：78—68
定价：2.15元 国外定价：5美元

编辑者：中国科学技术情报研究所重庆分所
顾 问：《计算机科学》审编委员会
出版者：中国科学技术情报研究所重庆分所
重庆市市中区胜利路132号
邮政编码：630013
印刷者：中国科学技术情报研究所重庆分所印刷厂
总发行处：四川省重庆市邮政局
订购处：全国各地邮政局