26-30 推理和人工智能 Horn子句逻辑 Horn子句

计算机科学1992 Vol.19 No.6

基于Horn子句逻辑的并行推理机 TP18

吴 陈（镇江船舶学院计算机系，镇江210003）

摘 要

The paper describes in detail a parallel inferenec machine based on Horn Clause Logic. The structure of this parallel inference machine is constructed by processes and process networks, and the way of computation of it is performed by process unifications, or process communications. Meanwhile, the paper points out the computing similarity between such a parallel inference machine and a neural network.

顺序推理中遵循先左后右——子句在前者先选择（即由上到下）的原则[3]，而人们发现 Horn 子句逻辑具有下列基本的并行性：

1) AND 并行性，即并行计算规则，指一个目标中的各个子句可并行约化；

2) OR 并行性，即并行搜索规则，指一个子目标可同时与多个可能匹配的子句进行匹配；

3) 搜索并行性，指同时搜索库中所有子句；

4) 合一并行性，指同时匹配单一目标中的各个项；

5) 流并行性，指具有共享变量的子目标间能形成产生者和消耗者关系。

到目前为止，对 Horn 子句逻辑的并行性的开发已有很多研究工作，设计了不少的并行程序设计语言，如Concurrent Prolog, Parlog, Guarded Horn Clauses, Epilog, Flat Concurrent Prolog, Flat GHC, P-Prolog, Flat PARLOG, FCP 等等[1]。它们的目的都是使用逻辑来进行程序设计，改进性能和表达能力等，但是离真正的逻辑还比较远，而且语义有待更进一步明确。

本文旨在设计一个基于Horn 子句逻辑，由进程和进程网络构成的并行推理机，并指出它同目前较热门的神经网络模型计算的相似性。

一、基于 Horn子句逻辑的并行推理机

对于基于 Horn 子句逻辑的演绎推理，如果消除那些语义上无关紧要的顺序性，那么它就更接近逻辑，具有完备性。如果仅关心演绎推理机制，而不关心逻辑如何应用于程序设计，那么可以沿着一个形式化方法定义一个从逻辑到并行机的严格描述。也就是说，有一个完全适合于以并行方式描述的推理机制的形式化，并且可以抽取机器体系结构的需要，离开冯·诺依曼体系结构。

下面通过一个例子来引出一般形式的并行机定义。

例1 一个简单的 Horn 子句例子。

P(x,y) ←Q (f(x)), R(x,y)

Q(f(z)) ←Q(z)

Q(a).

R(t,b).

P(a,u)?

我们约定，每个原子有一个进程相对应；当每个子目标与可能的子句头相合一时有一个连接相对应；而每个子句中的共享变

量的集合也有一个连接相对应。这样，对于目标P(a,u)？的并行机如图1所示。

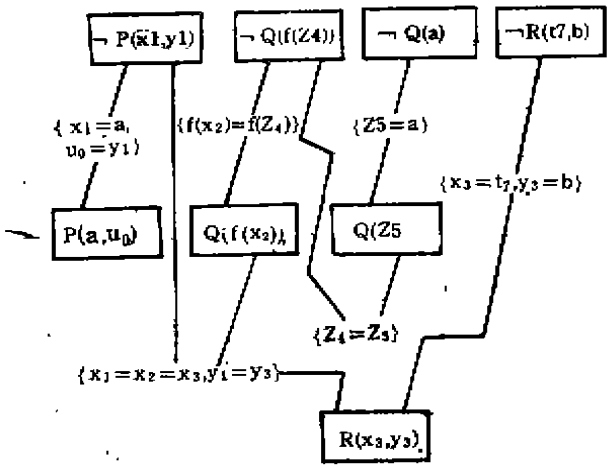


图1　例1的进程网络结构

由于并行机实际上被定义成互连通讯进程的集合，因此，合一就应由进程间的通讯完成。

图2中进程用方框表示，一进程与它进程之间的连接用线表示，在这里进程是状态转移自动机，而状态就是项，由某个有序字母表上的字母构造而得。例如 R(S(S(0)), S(0))，其中R是原子名，S为自然序后继函数。状态空间可以无限，而状态转移是项的重写。例如 R(m,S(n))⟹R(S(m),n)。

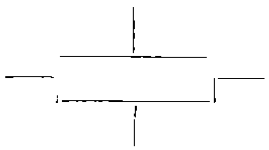


图2　进程的一般形式

规则l⟹r可用，如果当前状态q与l匹配，存在一个置换S，使得 q=S(l)，那么下一状态就是S(r)。例如，当前状态为R(S(S(0)), S(0))，且置换为〔m/S(S(0))，n/0〕时，下一个状态就为R(S(S(S(0))),0)。当几个规则都可用时，进程可能是非确定的，同样，非终止也就发生了。下面来看进程如何通讯。

二、进程之间通讯

先定义几个简单的通讯进程，作为了解进程间通讯的例子。

例2　作为简单通讯进程的加法器（图3(a)）。

ADDER　⇒Q
Q: A(m) B(n)⇒R(m,n)　用到通讯A和B。
R(m,S(n))⇒R(S(m),n)　无通讯。
R(m,0):C(m)⇒Q　用到通讯C。

其中，Q指建立加法器进程之前和之后的状态。从建立进程R(m,n)开始，这时由连接器A、B分别传入值m，n。在加法器进程的转移过程中可以用到不同数目的通讯，从1个到多个，也可以是0个通讯。

例3　作为简单通讯进程的缓冲区（图3(b)）。

BUFFER　⇒E
E: U(x)⇒F(x)　用到通讯U
F(x):V(x)⇒E　用到通讯V

例4　作为简单通讯进程的栈（图3(c)）。

STACK　⇒S(NIL)
S(1):I(e)⇒S(CONS(e,1))　用到通讯I
S((CONS(e,1):O(e)⇒S(1)　用到通讯O

栈是非确定的，依赖于在I或者O上通讯的环境。

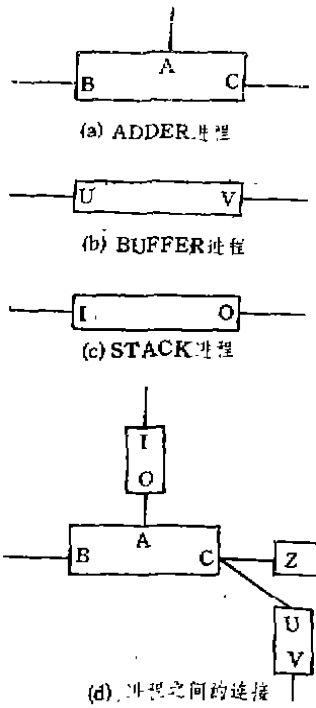
综上所述，进程通过连接器来通讯值，进程间的通讯连接也是靠连接器，连接器在图中用一些标号来表示（图3(d)），如简写成A，B,C,…，而值是指在某个有序字母表上的项，例如：0，S(0)，S(S(0))，…；NIL，CONS(S(0)，NIL)，…。通讯指在一个连接器上的一个值的临时发送，如A (S (S (0)))，B(S(0))，…。状态转移可以触发0、1或更多的通讯。如：

R(m,S(n))⇒R(S(m),n)　0个通讯
R(m,0):C(m)⇒Q　1个通讯
Q:A(m)B(m)⇒R(m,n)　2个通讯

我们所定义的并行机是一个通讯进程网络，也指进程使用某个进程代数的操作符所能进行的进程复合。例如，加法器和缓冲区的复合如图4(a)，可以直观地表示为：

MACHINE＝ADDER‖BUFFER
+{C, U}。

其意义在于，



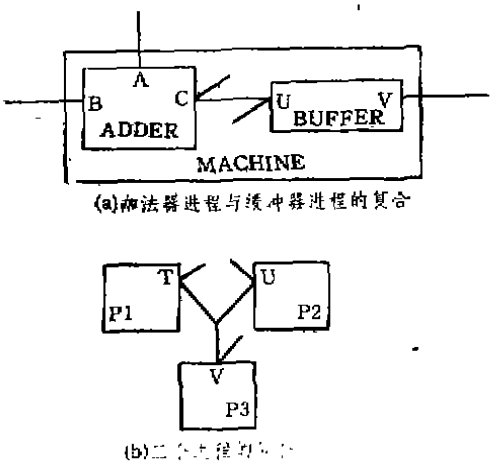
(a) ADDER进程

(b) BUFFER进程

(c) STACK进程

(d) 进程之间的连接

图3 进程及其连接



(a)加法器进程与缓冲器进程的复合

(b)三个进程的复合

图4 进程的复合

(1) 构成网络的进程结构；

(2) 形成一个与构成网络等价的新进程的转移规则。例如，加法器和缓冲区进程的并行复合具有如下的形式：

Q, E:A(m)B(n)U(x)⇒R(m, n), F(x)

R(m, S(n)), E:U(x)⇒R(S(m), n), F(x)

R(m, o), E:C(m)U(x)⇒Q, F(x)

...

R(m, o), F(x):C(m)V(x)⇒Q, B

注意，进程连接后，会产生新的规则，如R(m, 0), E⇒Q, F(m)。新规则由变量置换表示通讯所得。

由于通讯是项的合一，因此通讯可以是"双向的"。两个相连接进程经过通讯后，分别转移至不同的下步状态。另外，通讯也可以包含于两个以上进程，如图4(b)所示。进程P_1，P_2和P_3为三个不同的进程，两两之间存在通讯连接，可简写成：

$P_1 \| P_2 \| P_3 + \{T, U, V\}$。

其中，T，U，V分别是各进程的连接器(图4(b))。

上面已较全面地定义了进程及其间的通讯，现在看看如何将Horn子句程序编译成进程和进程网络。

三、Horn子句程序到进程网络的编译

再考察一下例1中的Horn子句程序，我们将每个进程所对应的原子联系起来，如下所示：

A_1 A_2 A_3

$P(x, y) \leftarrow Q(f(x)), R(x, y)$ C_1

A_4 A_5

$Q(f(z)) \leftarrow Q(z)$ C_2

A_6

$Q(a)$ C_3

A_7

$R(t, b)$ C_4

B_1

$P(a, u)?$ G

对于每个A_i，编译成下面形式的一个进程(图5)：

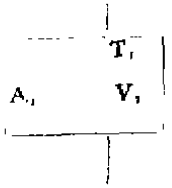


图5 原子A_i进程

A_i ⇒S_i

S_i:T_i(原子A_i的子项)

V_i(A_i所在子句的共享变量)

⇒S_i

其中如A_2可编译成下列形式：

A_2 $\Rightarrow S_2$

$S_2 : T_2(F(x_2))\ \ V_2(x_2, y_2) \Rightarrow S_2$

对于初始目标中的每个原子，也编译成一个进程，每个B_i与下列形式的一个进程对应（图6）：

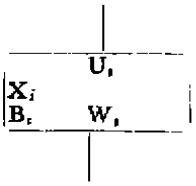


图6 子目标B_i进程

B_i $\Rightarrow S_i$

$S_i : U_i$（原子B_i的子项）

W_i（初始目标的共享变量）

$\Rightarrow R_i$（回答变量）

R_i（回答变量）$: X_i$（回答变量）$\Rightarrow S_i$

其中如B_1可编译成下述形式：

B_1 $\Rightarrow S_1$

$S_1 : U_1(A, U_1)\ W_1(\ \ ) \Rightarrow R_1(U_1)$

$R_1(U_1) : X_1(U_1) \Rightarrow S_1$

而对于每个子句则编译成一个网络表达式：

$C_1 = A_1 || A_2 || A_3 + \{V_1, V_2, V_3\}$

$C_2 = A_4 || A_5 + \{V_4, V_5\}$

$C_3 = A_6 + \{V_6\}$

$C_4 = A_7 + \{V_7\}$

$G = B_1 + \{W_1\}$

其中子句C_1的子网络如图7所示。

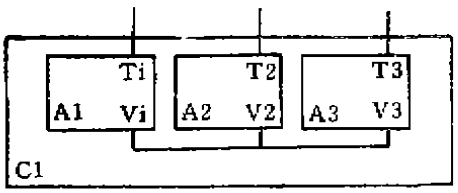


图7 子句C1的进程子网络

最后，整个Horn子句程序是一个子句网络表达式，构成该程序的并行推理机。例如上面例子的网络表达式为：

$PROG = C_1 || C_2 || C_3 || C_4 + \{T_2, T_3, T_4, T_6, T_6, T_7\} || G + \{U_1, T_1\}$

其中T_1，T_2，T_3，…，T_7为子句的通讯连接器，可以画成图8的形式：

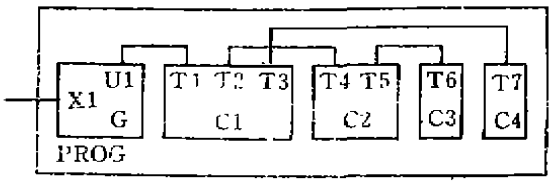


图8 例1子句间由连接器通讯的互连网络

通过连接器通讯的具体内容如图9所示：

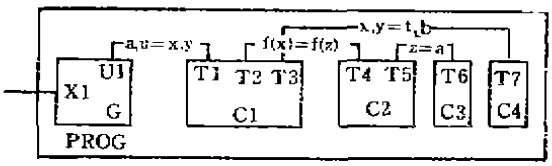


图9 例1子句间由通讯＝合一连接的互连网络

编译后的网络中，PROG这一并行推理机具有下列特征：（1）进程C_i不包括任何内存；（2）进程G有使PROG非同步及其环境的内存；（3）通讯代替计算。

四、递归子句和一般目标情形的编译

对于更一般的情形，例如递归形式的子句及任意形式的目标，我们采取下列方法编译得到相应的进程网络结构。

首先，对于递归子句，如第4节例子中的子句C_2，定义每一子句的例化为无界个数，因而形成给定进程的多份拷贝，构成无界子网络结构，如图10所示。

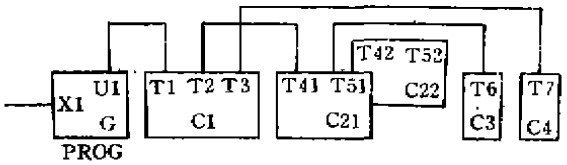


图10 例1子句间由连接器通讯的因递归子句导致的无界互连子网络

它适合于当目标为P(f(a), u)?, P(a, u)?及P(f(f(a)), u)?等形式的各种情形。子句C_2确定了不同的多份进程拷贝形式。

其次，对于多个原子组成的目标，特别是当这些原子具有同名异项时，例如P(a,

u)，P(f(a)，v)?，将目标G"塞进"子句构造PROG的网络形式，形成一般的解决方法。如图11，为下列目标的子网络：

$$G=P(a,\ u),\ P(f(a),\ v)?$$

B_1 B_2

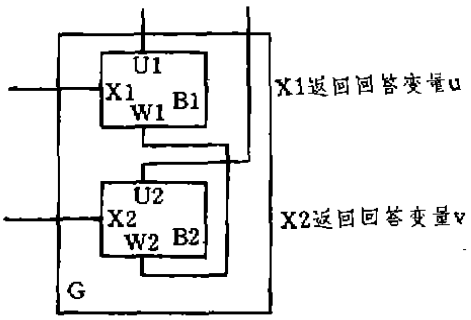


图11 多重子目标进程网络

这样，由于目标形式和子句递归以及多拷贝和例化，进程网络呈现为下面一般形式(图12)

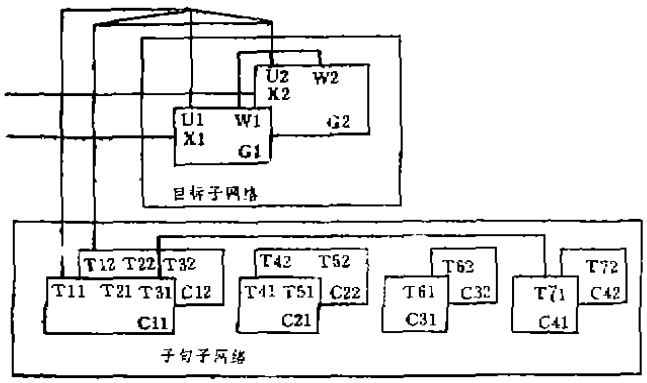


图12 进程网络的一般形式

此外，当出现多重解时，目标子网络也可以呈现为无数个进程的形式。

至此，我们定义了原子、子句和Horn子句程序编译为进程、进程子网络和进程网络的各种形式。进程之间的通讯由合一确定，进程即是一个状态转移自动机，而转换又可以定义为项的重写。多个进程的并行复合可形成新状态转移规则，"通讯=合一"是完成并行推理的最基本操作。

剩下的问题和困难是：如果存在证明时，机器会发现至少一个证明，但是，在不存在证明的情形下，机器将保持沉默；对于存在一个或多个证明，机器可能产生相同证明的几个实例。

原则上说来，我们定义的这种并行机器是"最并行"机械化的，而且对于连接机制也最好。但是，现今的计算机体系结构还不适合于这样的信息处理结构，它需要巨量而特小的进程、极高的连接性、基于合一的专门通讯原语、任意大的静态定义虚拟网络结构、静态结构的动态活动等。在目前条件下，可以在顺序机或传统并行机上模拟实现。

五、与生物神经网络的比较

本文中，我们对"计算"定义了以通讯作为信息处理的原子"活动"方式。这样一台并行推理机的操作进行如下：基本进程无"内存"，也不"计算"。当模式为几个互连进程共享时，子网络成为"相互协定"的子网络，进而触发为通讯"计算"状态。

比较一下本并行推理机同生物神经网络[4]，我们会发现，该并行机与生物神经网络十分相似：

(1) 均为巨大、密集的网络；

(2) 只有一些子网络包含在求解一给定问题中；

(3) 每个进程既是其紧邻进程的"模式发送者"，也是其紧邻进程的"模式匹配者"；

(4) 能力不受网络部分失败所影响；

(5) 对于观察者来说，高度并行的内部行为是不可及，也是无意义的，仅仅只知道它的有序输出结果；

(6) 无局部的信息表示；

(7) 因而，在某种意义下模拟了人脑的求解思路。

参 考 文 献

[1] Ehud Shapiro, The Family of Concurrent Logic Programming Languages, ACM Computing Surveys, 1989, 21(3), pp412-510

[2] [美]黄铠，布里斯格著，金兰等译，计算机结构与并行处理，科学出版社，1990