計算機アーキテクチャ　62－1
（１９８６．７．２１）

グラフ操作に基づく
Ｐｒｏｌｏｇ並列実行マシンのアーキテクチャ

片岡　良治　，伊藤　秀男　，新島　裕幸　，根元　晋一　，中道　松郎

千　葉　大　学　工　学　部

論理型言語Prologの所有する推論機構は，単一化処理の繰り返しにより実現されており，単一化処理の高速化がProlog高速処理には不可欠となる．本稿では，Prologプログラムの構成単位であるホーン節をホーン節グラフと名付けるグラフ構造データに変換し，これに対して単純なアルゴリズムを再帰的に適用することで，効率の良い高速な単一化処理を実行する方式と，グラフ操作を基調にPrologをＯＲ並列・ＡＮＤストリーム並列に処理するProlog高並列・高速実行マシンＧＲＩＭ(Graph Reduction Inference Machine)を提案する．最初に，ホーン節のグラフ表現法と単一化アルゴリズムが示される．次いで，組込み述語に対するグラフ操作方式とＡＮＤストリーム並列処理のためのグラフ分割方式が示され，最後にＧＲＩＭのハードウェア第一次モデルが示される．

ARCHITECTURE OF
THE PROLOG PARALLEL EXECUTION MACHINE
BASED ON GRAPH OPERATION

Ryoji KATAOKA , Hideo ITOH , Hiroyuki NIIJIMA , Shinichi NEMOTO , Matsurou MAKAMICHI

Faculty of Engineering, Chiba University, 1-33 Yayoichou, Chiba-shi, 260, Japan

Inference mechanism in the logical language Prolog is comprised in iteration of unification. So, fast unification is needed for fast processing of Prolog. In this paper, the method of efficient and fast unification is proposed. A horn clause, a unit of Prolog program, is interpretted on the graph structure named horn clause graph, and simple algorithm is applied to the graph recursively. Moreover, architecture of the Prolog parallel execution machine GRIM (Graph Reduction Inference Machine) is presented . In GRIM ,OR paralelism and AND stream paralelism based on graph operation are used.

First, graph representation of a horn clause and unification algorithm are described. Next, graph operation for built-in predicate processing and graph division for AND stream paralell operation are described, and last of all, prototype hardware model of GRIM is described.

１．まえがき

論理型言語Prologは第五世代計算機を始め，近年急速に発展しつつある知識工学の分野に於いても適合性を認められたプログラミング言語であり，各種実用システムへの応用のためにProlog高速処理系の開発が急務となっている．Prologの高速処理手法としては，それに内在する各種並列性の利用が有効であり[1][2]，現在各所で並列推論マシンの研究開発が行なわれている[2][3]．しかし，並列処理を行なう上でも，Prolog処理の基本操作である単一化処理の高効率化，高速化は重要な課題である．

単一化の概念は，Prologに限らず広汎な分野で利用されており，現在までに項の単一化について様々なアルゴリズムが提案されている[4]-[8]．中でもRobinsonのアルゴリズム[4]の改良版としてCorbin, Bidoitにより提案されたものは[6]，複雑な構造を持ち計算機での取り扱いが容易ではない項を項グラフと呼ばれる比較的単純なグラフ構造データに変換することで，単一化を単純なアルゴリズムの再帰的な繰り返しにより効率良く実現しており，計算機処理に向いたアルゴリズムであると言える．我々は，このアルゴリズムに着目し，Prologのホーン節を項グラフの拡張形式であるホーン節グラフに変換することで，グラフ操作によりPrologの単一化処理を効率良く実現する方式を考案した．本稿は，このようなホーン節グラフを用いた単一化処理方式を基調として現在我々が検討を進めているProlog並列実行マシンＧＲＩＭ(Graph Reduction Inference Machine)のアーキテクチャについて述べるものである．

以下，２章でホーン節グラフを定義し，３章でホーン節グラフを用いたPrologの単一化アルゴリズムを示す．４章では，ＧＲＩＭに実装する組込み述語について述べ，５章では，ＧＲＩＭに於けるPrologの並列実行方式を示す．６章では，ＧＲＩＭのハードウェア第一次モデルについて概説する．

２．ホーン節のグラフ表現

ＧＲＩＭは，Prologのホーン節をグラフ表現し，グラフ操作によりPrologを処理する所に特徴を持つ．この章では，ＧＲＩＭに於けるホーン節グラフの定義について述べる．

２．１　項グラフ

ホーン節グラフは，項グラフの概念をホーン節へ拡張したものである．そこでホーン節グラフについて述べる前に，項及び項グラフを次のように定義する[4][6][8]．

【定義１】　Ｖを変数の集合，Ｆを関数記号の集合（但し，$V \cap F = \phi$）とする．$f \in F$がｎ個の引数を持つ時，ｆをｎ次関数記号と呼ぶ．特にｎ＝０の時，ｆを定数と呼ぶ．$V \cup F$上で，

(1) 全ての変数$x \in V$は項である．
(2) 全ての定数$a \in F$は項である．
(3) $t_1, t_2, \cdots, t_n$ を項，$f \in F$をｎ次関数記号($n \geq 1$)とする時，$f(t_1, t_2, \cdots, t_n)$は項である．
(4) (1)から(3)によって得られるもののみが項である．

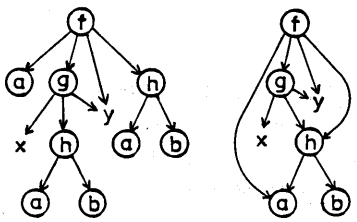
【定義２】　Ｎをノードの集合，Ｅを有向アークの集合とする時，以下の条件を満たす有向非巡回グラフＧ＝（Ｎ，Ｅ）を項グラフと呼ぶ．

(1) Ｎ中のノードは$V \cup F$中の要素をラベルとして持つ．
(2) Ｖ中のラベルが付いたノードから出るアーク数は０であり，同一ラベルを持つノードは２つ以上存在しない．このノードを変数ノードと呼ぶ．
(3) Ｆ中のｎ次関数記号がラベルとして付いたノードから出るアーク数はｎである．このノードを関数ノードと呼ぶ．

【定義３】　項グラフＧの各ノードは次のような項を表わす．

(1) ｘをラベルに持つ変数ノードは項ｘを表わす．
(2) ｎ次関数記号ｆをラベルに持つ関数ノードは，このノードから出るアークの先にあるノードが項$t_1, t_2, \cdots, t_n$を表わす時，項$f(t_1, t_2, \cdots, t_n)$を表わす．

項グラフの例を図2.1に示す．変数ノードはそのラベルのみで表わし，関数ノードはそのラベルを○で囲むことで表わす．関数ノードの共有，非共有により，１つの項に対して項グラフは，複数考えられる場合がある．



f(a,g(x,h(a,b),y),y,h(a,b))
f,g,h,a,b∈F　x,y∈V

図2.1 項グラフ

２．２　ホーン節グラフ

ホーン節は，ゴール節，規則節，事実節に大別されるが，これらはどれも１つ以上の項（Prologに於いては一般にリテラルと呼ばれる）の並びと見なせる．そこで，項グラフの拡張としてホーン節グラフを以下のように定義する．

【定義４】　ホーン節$B_1, \cdots, B_m :- A_1, \cdots, A_n.$($0 \leq m \leq 1, n \geq 0$)に対して，リテラル$A_i, B_i$に対する項グラフを$G_{Ai}, G_{Bi}$とする時，以下の条件を満たすグラフＨ＝（Ｎ，Ｅ）をホーン節グラフと呼ぶ．

(1) $\forall G_{Ai}$及び$\forall G_{Bi}$はＨの部分グラフである．但し，各G_{Ai}, G_{Bi}に対応するＨ中の部分グラフは唯一である．
(2) Ｎ中の入力アークを持たないノードが表わす項に等しいリテラルA_iまたはB_iが存在する．但し，各項に対応するA_iまたはB_iは唯一である．
(3) Ｎ中には同一ラベルを持つ変数ノードは２つ以上存在しない．
(4) Ｎ中の変数ノードの内，大域変数（初期ゴール節中の変数）に対応する変数ノードのみ出力アークを１つ持つ．

【定義５】　ホーン節グラフは次のように分類される．

(1) ゴール節（ｍ＝0）に対するホーン節グラフをゴール節グラフと呼ぶ．
(2) 定義節（規則節＋事実節，ｍ＝1）に対するホーン節グラフを定義節グラフと呼ぶ．

項グラフの場合と同様にホーン節グラフも１つのホーン節に対して複数考えられる場合がある．ＧＲＩＭではホーン節グラフをメモリ上に表現する場合，グラフを構成する各ノードをメモリセルに対応付けているので，グラフに含まれるノード数は出来る限り少ない方が望ましい．そこで以降，ホーン節グラフを表現する時は，関数ノードを出来る限り共有させたものを用いることにする．

また，大域変数ノードに与える出力アークは，処理終了後に得られるグラフ中から解（大域変数に対する束縛項）を容易に抽出するためのものであり，初期状態では自ノードへのループアークとなってい

る．つまり，ホーン節グラフでは，項グラフの定義に含まれない巡回性がこのアークのみ許されるものとする．詳しくは３章で述べる．

例として，appendプログラムのホーン節に対するホーン節グラフを図2.2(a)に，以降の説明で用いるホーン節グラフを構成するノードの分類と図2.2(a)に対応した例を図2.2(b)に挙げる．

３．単一化アルゴリズム

この章では，ホーン節グラフを用いたグラフ操作に基づくPrologの単一化処理方式を示す．

Prologに於ける単一化処理は次の２つの処理から成る．

①ゴール節中の１つのリテラルと定義節頭部リテラルの単一化

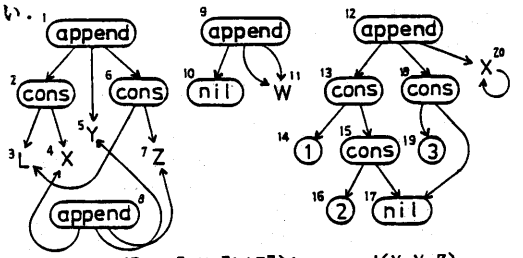
②①の単一化により生成された束縛環境に基づく，定義節本体部及び①の単一化の対象とならなかったゴール節リテラルからの新ゴール節の導出

②の処理が①の処理に依存するものであるという性質上，一般に①，②の処理は，この順序で逐次的に実行される場合が多い[9][10]．ＧＲＩＭに於いては，ホーン節グラフの特徴を生かし，これら２つの処理を同時に実行している．

３．１　単一化処理とグラフ操作

単一化処理の手順をホーン節グラフの操作へ対応付けて見る．一般に単一化の成否を決定する第一条件としては，単一化を行なう２つのリテラルの述語及びその引数の個数の一致が挙げられる．これはゴール節グラフ，定義節グラフが与えられた時，単一化を行なうリテラルの述語をラベルとして持つそれぞれのグラフ中の述語ノード（図2.2(b)参照）について，そのラベルと出力アーク数の一致を調べることに相当する．

条件を満たしたものについては，次に引数項の単一化を行なうことになる．グラフ上では，各グラフの述語ノードから対応出力アークをたぐり，その先にあるノードが定義３に基づいて表わす項を単一化することに相当する．従って，次の操作を行なえば良い．




```
append([L|X],Y,[L|Z]):-append(X,Y,Z).
append([],W,W).
?-append([1,2],[3],X).
```

(a) appendプログラム

分類		備考	(a)の対応ノード
関数ノード	述語ノード	述語をラベルに持つ	1,8,9,12
	引数関数ノード		2,6,13,15,18
	定数ノード	出力アーク数が０	10,14,16,17,19
変数ノード	大域変数ノード	出力アークを持つ（定義５）	20
	局所変数ノード		3,4,5,7,11

(b) ノードの分類

図2.2 ホーン節グラフ

(a) 共に関数ノードの場合

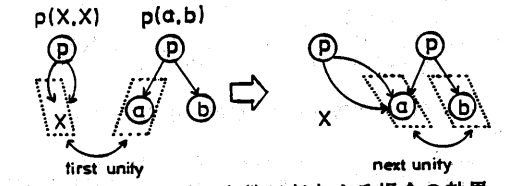
述語ノードに対して行なう処理と同様の処理を行なう．つまり，ラベルと出力アーク数の一致を調べ，対応出力アークの先にあるノードについて，それが表わす項の単一化を行なう．

(b) 少なくとも一方が変数ノードの場合

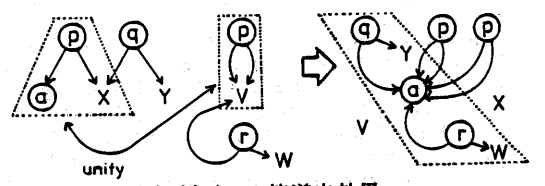
この場合は，変数に対する束縛が生じる．グラフ上では，単一化以前に変数ノードを指していたアークが，単一化以降，束縛項を表わす項グラフを指すようにすれば良い．つまり，変数ノードへの入力アークが，他方のノードの入力アークとなるようなアークの付け替え操作（これをreplace操作と呼ぶ）をすれば良い．但し，共に変数ノードの場合は，定義節側の変数を束縛項と考える．

ホーン節グラフに対するreplace操作は単一化処理に於いて次のような効果を持つ．

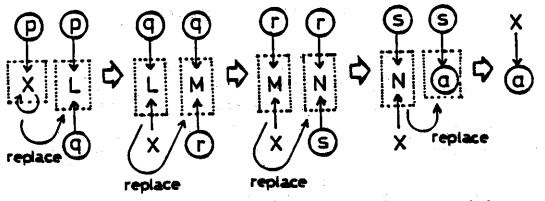
まず，引数間で共通な変数（リテラル内共有変数）に対する単一化を容易にしている．従来の方式では，引数項の単一化に於いて同一変数に対する単一化が２回以上起こる場合，先行した単一化による束縛項を参照しながら，その変数に対する以降の単一化を行なう必要がある．例えば，リテラルp(X,X),p(a,b)の単一化に於いて引数項の単一化を左側から順に行なうとすれば，第二引数の単一化は，第一引数の単一化により変数Ｘに代入された束縛項ａを参照した後，項ａ，ｂの単一化を試みるという手順になる（この単一化は失敗である）．replace操作は，このような参照の手間を省く効果を持つ．定義４により，ホーン節グラフ中には同一変数ラベルを持つ変数ノードは２つ以上存在しない．従って，引数間で共通な変数に対するホーン節グラフ中のノードは同一であるため，第一引数の単一化でreplace操作により変数ノードＸへの入力アークを全て束縛項ａを表わすノードへ付け替えておけば，第二引数の単一化では，直接，項ａ，ｂの単一化が行なえる(図3.1(a))．



(a) 引数間に共通な変数が存在する場合の効果



(b) 新ゴール節導出効果




```
?-p(X).    p(L):-q(L).    r(N):-s(N).
           q(M):-r(M).    s(a).
```

(c) 中間変数のたぐりの不要化

図3.1 replace操作による効果

また，replace操作はこれと同様な仕組みにより，単一化と同時に本章の始めに挙げた②の処理を行なっていることが分かる．replace操作は，変数ノードへの入力アークを全て一括して付け替えるため，定義節本体部に対応するグラフ中からの入力アーク及び単一化に用いられなかった未使用ゴール節リテラルに対応するグラフ中からの入力アーク（リテラル間共有変数）についても同時に付け替えが行なわれる．従って，単一化により変数に束縛項が代入されると，その変数を参照している定義節本体部及び未使用ゴール節リテラルに対しても同時に束縛項の代入が起こり，単一化と新ゴール節の導出が同時に行なわれる(図3.1(b))．

更にreplace操作は，変数に対する束縛項を求める時に必要となる中間変数のたぐり操作を不要とする効果を持つ．図3.1(c)のプログラムを実行した場合，入力ゴール節で与えた変数Xに対する束縛項を求めるには，X→L→M→N→aというたぐりを行なわなければならない．しかし，replace操作の結果，グラフ上では，定義４で与えた大域変数ノード(図2.2(b)参照)からの出力アークが，直接，束縛項を指しており，中間変数のたぐりを行なう必要はない．

３．２　単一化アルゴリズム

ホーン節グラフのグラフ操作に基づく単一化処理のためのアルゴリズムを図3.2に示す．ゴール節グラフの述語ノードV1と定義節グラフ頭部リテラルの述語ノードV2を入力とし，単一化の成否を出力とする．単一化に成功した場合は，副作用としてゴール節グラフ，定義節グラフから新ゴール節に対するゴール節グラフが導出される．但し，アルゴリズム中に用いられている各関数は，次のような意味を持つ．

- replace(V,W): 前節で述べたreplace操作を行なう関数である．変数ノードVへの入力アークを全てノードWへ付け替える．
- indegree(V): ノードVへの入力アーク数を与える．
- outdegree(V): ノードVからの出力アーク数を与える．
- arity(V): 関数ノードVが持つ引数の個数を与える．
- label(V): ノードVに付けられたラベルを与える．
- term(V): 定義３によりノードVが表わす項に対応するホーン節グラフ中の項グラフを与える．
- arc(V,k): 関数ノードVが持つ左からk番目の出力アークを与える．
- succ(V,k): 関数ノードVが持つ左からk番目の出力アークの先にあるノードを与える．
- delete(X): Xが示すものをグラフ中から削除する．但し，X=term(V)の場合はterm(V)が与える項グラフの内,他から参照のない部分のみを削除する(図3.3)．

前節で述べたノードのラベル及び引数個数の比較が第9,10行で行なわれ，第13～26行のループで引数ノードに対する単一化が行なわれる．ループ中，第24行で単一化アルゴリズムが再帰呼び出しされている．

また，このアルゴリズムでは前節で述べたグラフ操作に加えて，単一化後，グラフ中で不要となるガーベッジ・ノードを単一化操作に伴い削除している(第8,16～23,28～33行)．従って，処理後に得られるグラフは，導出された新ゴール節及び束縛環境に関するノードのみを含む．

図3.4に図2.2(a)のプログラムの実行例を示す．この例からも分かるように，実行過程で生成されるグラフには，ゴール節グラフに加えて以下に定義する束縛環境グラフが含まれる．そして，全ての単一化終了時には束縛環境グラフのみとなり，これが解

```
ALGORITHM UNIFY: unify(V1,V2)

INPUT: A pair(V1,V2) of nodes such that term(V1)
is one of literals in a goal clause and term(V2)
is a head literal of a definition clause.

OUTPUT: Bool=true if and only if unification
has succeeded,and in this case,as side-effect,
two graphs are reduced to a new goal clause
graph,including binding environment.

 1 BEGIN
 2 IF both nodes are variable nodes,let V be one in
 3    a goal clause graph and let W be the other in
 4    a definition clause graph
 5 IF either node, V ,is a variable node, let W be
 6    the other
 7 THEN replace(V,W); bool:=true;
 8      IF outdegree(V)=0 THEN delete(V) FI.
 9 ELSE IF label(V)≠label(V2)
10         OR arity(V1) ≠arity(V2)
11      THEN bool:=false.
12      ELSE k:=0; bool:=true;
13           WHILE k<arity(V1) AND bool
14           DO k:=k+1; W1:=succ(V1,k);
15                      W2:=succ(V2,k);
16              IF indegree(V1)=0
17              THEN delete(arc(V1,k))
18              FI;
19              IF indegree(V2)=0
20              THEN delete(arc(V2,k))
21              FI;
22              IF W1=W2
23              THEN delete(term(W1)).
24              ELSE bool:=unify(W1,W2)
25              FI
26           OD;
27           IF bool
28           THEN IF indegree(V1)=0
29                THEN delete(V1)
30                FI;
31                IF indegree(V2)=0
32                THEN delete(V2)
33                FI
34           FI
35      FI
36 FI
37 END
```

図3.2 単一化アルゴリズム

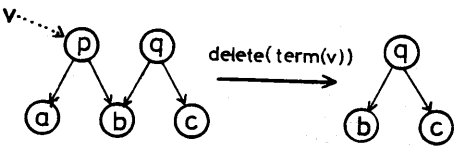


図3.3 delete(term(X))によるグラフ操作

を表わす．（以降の説明では便宜上，束縛環境グラフを含めてゴール節グラフと呼ぶ．）

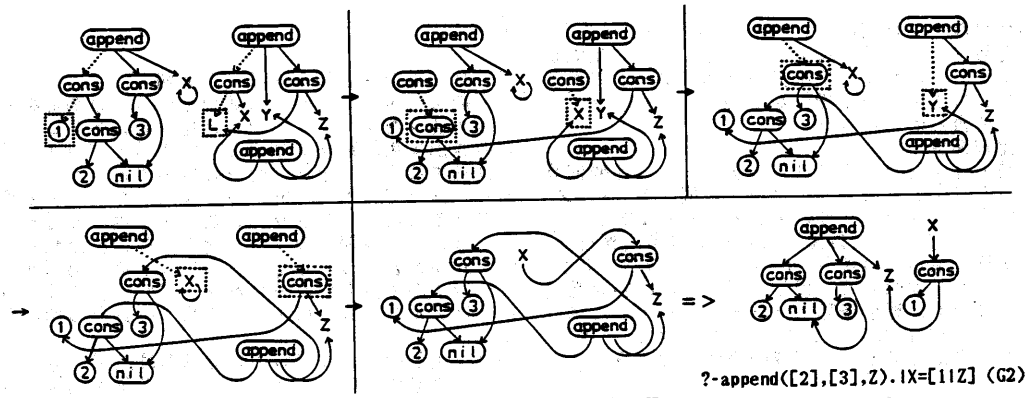
【定義６】　大域変数ノードは次のような束縛環境を表わす．

(1) ラベルがXであり出力アークが自ノードへのループアークである時，変数Xは未束縛である．

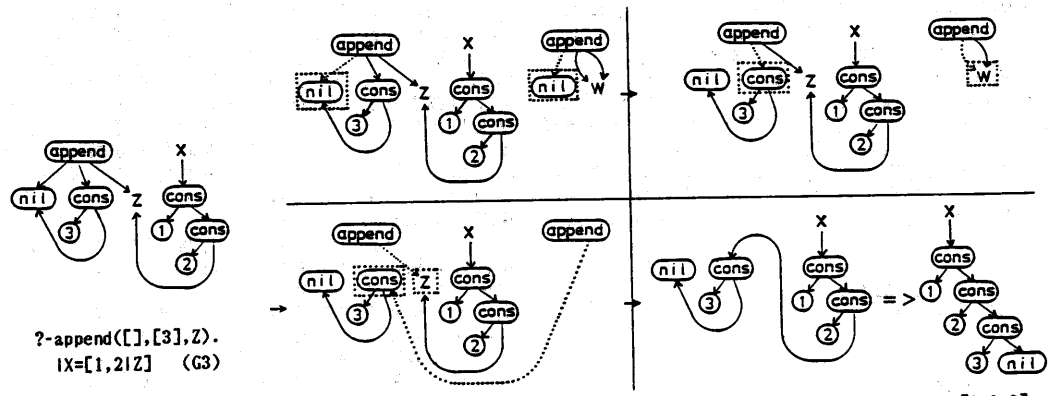
(2) ラベルがXであり出力アークが自ノード以外のノードｖを指す時，X＝term（ｖ）である．

【定義７】　大域変数とその束縛項に関するグラフを束縛環境グラフと呼ぶ．

なお，変数名の与え方（例えば，?-append([1,2],[3],Z).）によっては，実行過程で生成されるゴール節グラフ中に見かけ上，同一ラベルを持つ変数ノードが２つ以上存在し,定義４に反する場合が起こり得る.しかし，これらの変数ノードは，同一ラベ



(a) (D1)と(G1)の単一化過程



(b) (D1)と(G2)の単一化により得られるゴール節グラフ

(c) (D2)と(G3)の単一化過程

図3.4 appendプログラムの実行例

ルの異なる変数に対するものであり，グラフ中で同一ノードとすることは出来ない.従って,このようなゴール節グラフも定義4で与えるグラフの本質性には反していない.

4．組込み述語とグラフ操作

組込み述語を含むリテラルに対する処理は一般のリテラルに対する単一化処理とは異なるため，前章で述べた単一化アルゴリズムを組込み述語を含むリテラルに対して適用することは出来ない．この章では，ＧＲＩＭに搭載する組込み述語とそれに対するグラフ操作について述べる．

4．1　組込み述語のグラフ表現

ＧＲＩＭの初期モデル搭載言語はpure Prologとし，次のような組込み述語を含むものとする．更に多くの組込み述語を含ませる場合も，以降で述べるグラフ操作により，ほぼ同様に処理することができる．

①算術演算子・・・・ ＋，－，＊，／，mod
②比較演算子・・・・ ＜，≦，＞，≧，＝，≠
③数式評価演算子・・ is

ホーン節中で組込み述語は一般のリテラルの述語とは異なりinfix形式で記述されるが，ホーン節グラフ上では一般のリテラルと全く同様の表現方式を採る．図4.1,4.2に例を示す．このように算術演算子は，直接，リテラルの述語として用いられることはなく，比較演算子及び数式評価演算子に対する引数項中に用いられる．

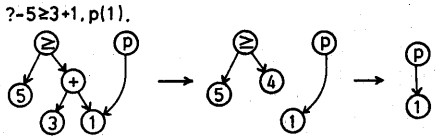


図4.1 比較演算子に対するグラフ操作

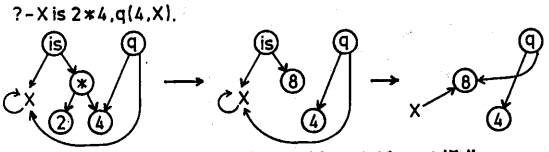


図4.2 数式評価演算子に対するグラフ操作

4．2　組込み述語に対するグラフ操作

ホーン節グラフを用いたProlog処理に於いて組込み述語は次のように処理される．

(1) 算術演算子

算術演算子をラベルに持つ引数関数ノード（算術演算子ノードは図2.2(b)の引数関数ノードに属する）Vの２つの出力アークをたぐり，その先にある定数ノード（図2.2(b)参照）W1,W2が持つラベル（数値）に対して該当演算を施す．W1,W2は他のリテラルと

の共有がない場合削除し，Vは演算結果をラベルとした定数ノードに置き換える．但し，W1またはW2が定数ノードではなく，更に算術演算子ノードである場合は，以上のグラフ操作をその算術演算子ノードに対して再帰的に行なう．

(2) 比較演算子

比較演算子をラベルに持つ述語ノード（比較演算子ノードは図2.2(b)の述語ノードに属する）Vの２つの出力アークをたぐり，その先にある定数ノードW1,W2が持つラベル（数値）に対して該当演算を施す．演算結果が真の場合，delete(term(V))により不要ノードを削除する．但し，W1またはW2が定数ノードではなく，算術演算子ノードとなっている場合は(1)の操作を行ない，それにより得られた定数ノードをW1またはW2として同様の操作を行なう．（図4.1）

(3) 数式評価演算子

数式評価演算子isをラベルに持つ述語ノード（数式評価演算子ノードは図2.2(b)の述語ノードに属する）Vの２つの出力アークをたぐり，左側の出力アークの先にある変数ノードW1と右側の出力アークの先にある定数ノードW2に対してreplace(W1,W2)を行なう．Vは削除，W1は出力アークを持たない場合削除する．但し，W2が定数ノードではなく，算術演算子ノードである場合は(1)の操作を行ない，それにより得られた定数ノードをW2として同様の操作を行なう．また，W1が定数ノードである場合は，isを比較演算子＝と見なし(2)の操作を行なえば良い．（図4.2）

５．並列処理方式

この章では，ＧＲＩＭで用いるPrologの並列処理方式について述べる．

５．１　基本方針

Prolog処理の並列性には，ＯＲ並列，ＡＮＤ並列，引数間並列，ＡＮＤストリーム並列が存在するが[2]ＧＲＩＭに於いては次のような理由からＯＲ並列と擬似的なＡＮＤストリーム並列を採用する．

(1) ３章で述べた単一化アルゴリズムによる単一化処理は，処理過程で与えられたゴール節グラフ，定義節グラフの構造を書き替えているためバックトラック制御に適していないが，バックトラックを必要としないＯＲ並列には有効である．

(2) ＡＮＤ並列処理を行なうためには，ゴール節グラフをリテラル単位に分割する必要がある．この時，リテラル間に共通な変数が存在するならば，この変数に対するゴール節グラフ中の変数ノードは唯一であるため，変数ノードのコピーを行なう必要が生じる．しかし，変数ノードをコピーしたのでは，replace操作による効果が全て損われてしまい，単一化アルゴリズムを採用するメリットが失われる．

(3) 単一化アルゴリズムでは，引数間の単一化を逐次的に行なっている．現段階では，このアルゴリズムをそのままハードウェア的に実現する方針で検討を進めているので引数間並列は考慮しない．

(4) マルチプロセッサ上でＯＲ並列のみを行なう場合，１つのゴール節に含まれるリテラル数が膨大となり，これがプロセッサ間通信のオーバーヘッドを招く可能性がある．これを解消するためにＯＲ並列に加えてＡＮＤストリーム並列を採用するという方法が考えられるが，ＡＮＤストリーム並列を忠実に実現したのでは(2)と同様の問題が起きてしまう．そこで，ゴール節をリテラル単位に分割するのではなく，replace操作の効果を最大限に生かせるようにゴール節グラフを分割した上で擬似的なＡＮＤストリーム並列を行なうことにする．詳しくは次節で述べる．

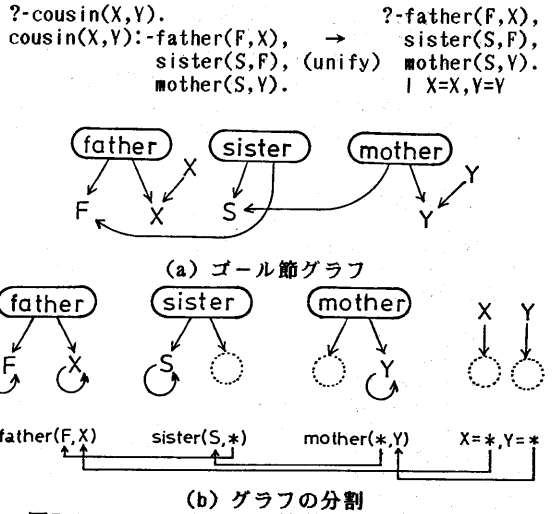
５．２　擬似ＡＮＤストリーム並列

一般にＡＮＤストリーム並列とはＡＮＤ関係のリテラル各々を１つのプロセスと見なし，各プロセスが導き出す束縛環境をプロセス間でパイプライン的に授受することで全体の解を導き出すというものである．従って，ホーン節グラフを用いたProlog処理に於いてもＡＮＤストリーム並列を忠実に実行するならば，ゴール節グラフをリテラル単位に分割するのが適切と言える．しかし，このような分割方式はリテラル間で共有される変数ノードが存在する場合，単一化アルゴリズムによる次の２つの効果の低減を招く．

①単一化と新ゴール節導出の同時処理効果の低減
②単一化過程でのガーベッジ・ノード削除効果の低減

図5.1(a)のゴール節グラフを通常のＡＮＤストリーム並列のためにリテラル単位に分割すると図5.1(b)のようになる．このようにリテラル間で共有される変数ノードは，そのノードを共有するリテラルの内，リテラルの並びの中で最も左側に位置するものに所属させ，大域変数ノードと同様のループアークを持たせる．このループアークは，そのノードが属するリテラルのプロセス処理過程で変数への束縛項を指すようになり，そのノードを共有する他のリテラルはプロセス起動時にこの出力アークをたぐり，束縛項を求めることになる．しかし，このようなアークのたぐり操作は，単一化処理に於けるreplace操作の効果により本来全く不要なものであり，①の原因となる．また，共有される変数ノード及びその出力アークが指す束縛項に対する項グラフは，この変数ノードが大域変数ノードでない限り，ＡＮＤストリーム並列処理終了後ガーベッジとなるが，これらがどの段階でガーベッジになるかを判別するのは非常に困難である.これが②の原因となる．

そこで，ＧＲＩＭに於いては①，②の解消可能なゴール節グラフの分割方式によりＡＮＤストリーム並列と同様の実行制御を行なう．これを忠実なＡＮＤストリーム並列に対して擬似ＡＮＤストリーム並列と呼ぶことにする．擬似ＡＮＤストリーム並列のために図5.1(a)のゴール節グラフを分割すると図5.2のようになる．図中の各段階では，次のような操



(a) ゴール節グラフ

(b) グラフの分割

図5.1 ＡＮＤストリーム並列のためのグラフ分割

作が行なわれている．

Step 1　ゴール節グラフをリテラル単位に分割する．各分割グラフに含まれるノードの集合をそれぞれ $N_1 \sim N_{n+1}$（n＝リテラル数）とする．ここで，N_{n+1} は束縛環境グラフに対応するものである．

Step 2　N_i に属するノードvの出力アークが N_j (j<i)に属するノードを指す時，$N_i' - \{v\} \rightarrow N_i'$，$N_{min\ j}'' \cup \{v\} \rightarrow N_{min\ j}''$ とする．但し，初期状態で $N_i' = N_i$，$N_i'' = \phi\ (1 \leqq i \leqq n+1)$ である．

Step 3　$N_i' \cup N_i'' \rightarrow N_i$ とする時，各 N_i の要素により構成されるグラフ G_i が擬似ＡＮＤストリーム並列のためのゴール節分割グラフである．

擬似ＡＮＤストリーム並列では，以上の分割により得られた $G_1, G_2, \cdots, G_n$ の各々を１つのプロセスと見なし，プロセス間で次のようなパイプライン処理を行なう．

［Ⅰ］G_1 を第１プロセスとして起動する．ゴール節グラフが ?-$A_1, A_2, \cdots, A_n$ なるゴール節に対するものである場合，G_1 にはリテラル A_1 に対する項グラフが含まれ，この起動により A_1 に関する単一化処理が開始される．

［Ⅱ］第(i-1)プロセス(2≦ i ≦n)に於ける単一化処理の結果としてグラフ G_{i-1}' が得られる毎に G_i と G_{i-1}' を併合した上で第 i プロセスを起動する．この時，G_i と G_{i-1}' の併合グラフにはリテラル A_i 中の変数に第(i-1)プロセス以前の処理で得られた束縛項を代入したものに対する項グラフが含まれる．

［Ⅲ］第 n プロセスにより得られるグラフ G_n' を G_{n+1} と併合し，これを現在のプロセス系列を呼び出した１つ高いレベルにあるプロセスへ返す．

実行例を図5.3に示す．このように擬似ＡＮＤストリーム並列では，プロセス処理時にそのプロセス中の変数ノードが自身への全ての入力アークを保持しているため，replace操作の効果が完全に生かされ①が解消される．また，リテラル間で共有される変数ノードに出力アークを与える必要がないことから，本来通り単一化処理過程で全てのガーベッジ・ノードが削除され②が解消される．

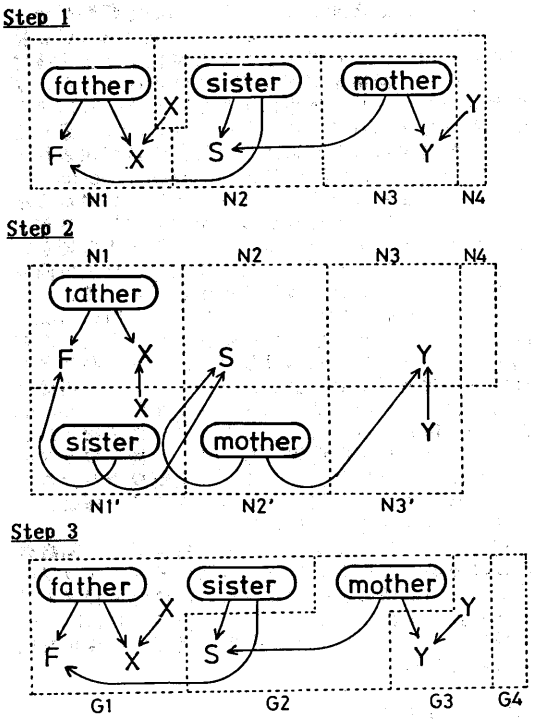


図5.2 擬似ＡＮＤストリーム並列のためのグラフ分割

６．　ハードウェア第一次モデル

この章では，現在我々が検討を行なっているＧＲＩＭのハードウェア第一次モデルについて述べる．

６．１　ホーン節グラフの内部表現

ホーン節グラフは，マシン内部で図6.1のようなセルを構成単位として表現される．このセルはホーン節グラフの各ノードに対応するものであり，各フィールドは次のような意味を持つ．

Node#：ノードの識別番号．
Tag：ノードの種類（変数ノード，関数ノード等）を示すタグ．
Label：ノードの持つラベル．

CAR：左側の出力アークの先にあるノードのNode#．
CDR：右側の出力アークの先にあるノードのNode#．

フィールド構成から分かるように，マシン内部では，１つの関数ノードからの出力アークは２つ以下に制限される．出力アークを３つ以上持つ関数ノードは，図6.2に示すようにlinkをラベルに持つダミーノードを用いて表現する．

６．２　ハードウェア構成

前章までに述べた各種グラフ操作及び並列処理方式を実現するためのハードウェア構成として，現在我々が検討を進めているものが図6.3である．図中の各ブロックは次のような機能を持つ．

(a)マッチングユニット（MU:Matching Unit）：Network2からプロセスを受け取り，そのプロセスに含まれるリテラルとの単一化処理の対象となる定義節グラフの格納アドレスをＭＵ内の定義節グラフテーブル(DGT: Definition clause Graph Table)によ

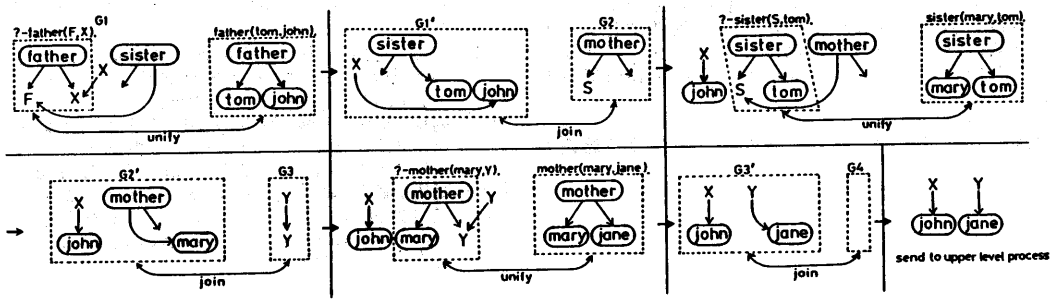


図5.3 擬似ＡＮＤストリーム並列の実行例

り選択する．定義節グラフはＵＰ内の定義節グラフメモリ(DGM:Definition clause Graph Memory)に格納されており，ＤＧＴによりＯＲ関係にある定義節グラフのＤＧＭ格納アドレスが検索可能である．プロセスをコピーし，検索により得られたＤＧＭアドレス各々をそれらに付加した上でNetwork3を通してＵＰに転送する．これによりＯＲ分岐が実現される．
(b)単一化プロセッサ(UP:Unification Processor)：ＭＵで付加されたアドレスをもとにＤＧＭから単一化処理に用いる定義節グラフを取り出し，３章で示した単一化アルゴリズムに従って単一化処理を行なう．ＯＲ分岐したプロセスを複数のＵＰ上で同時に処理することでＯＲ並列処理が実現される．
(c)グラフ分割プロセッサ (GDP: Graph Division Processor)：５章で示した擬似ＡＮＤストリーム並列のためのグラフ分割処理及びプロセス起動制御を行なう．起動プロセスはNetwork2に出力し，待機プロセスはＧＤＰ内のプロセステーブル(PT: Process Table)に格納する．ＭＵからのＯＲ分岐情報及びＵＰからの単一化処理情報に基づく，待機プロセスのガーベッジ・コレクションにより全解探索を行なう．また，４章で示した組込み述語に対する処理は，このプロセッサ内で行なわれる．

現在，単一化アルゴリズム中の各種組込み関数処理及び擬似ＡＮＤストリーム並列に於けるグラフの分割処理を高速に実行するための高機能連想メモリの基本設計を終了しており，これを各ＵＰ,ＧＤＰに用いることでプロセッサ内での処理の高速化を図る方針である．

７．むすび

本稿では，現在我々が検討を進めているProlog並列実行マシンＧＲＩＭのアーキテクチャについて述べた．ＧＲＩＭは，Prologのホーン節をホーン節グラフと呼ぶグラフ構造データに変換し，これを用いてグラフ操作により単一化処理を実行する点に特徴を持つ．本稿で提案した単一化アルゴリズムによれば，
(1)引数間で共通な変数に対する同一束縛項の代入処理
(2)単一化操作とゴール節導出操作の同時処理
(3)変数に対する束縛項探索時の中間変数のたぐりの不要化
(4)単一化処理過程でのガーベッジ・データの即時消去
が容易に可能である．

また，アルゴリズム自身が単純な再帰的繰り返しにより構成されているためハードウェアへの実装も容易と考えられ，これにより高効率かつ高速な単一化処理が実現可能である．

更にＧＲＩＭでは，ＯＲ並列及び擬似ＡＮＤストリーム並列を採用し，Prologを高並列に処理することでProlog高速処理系の実現を目指している．

今後，各プロセッサに搭載する高機能連想メモリの詳細設計，プロセッサ間ネットワークに関する検討，各種シミュレーション評価，言語の拡張等を行なう予定である．

謝辞　研究に御協力頂いた塩原隆一氏に深く感謝致します．

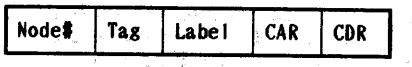


図6.1 ノードセルのフィールド構成

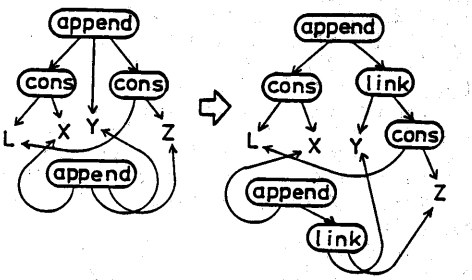


図6.2 linkノードの使用例

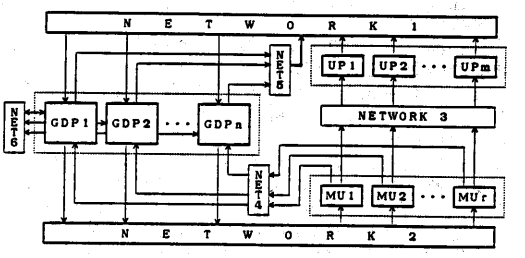


図6.3 ハードウェア第一次モデル

参考文献
[1] "特集：プログラミング言語Prolog", 情報処理, Vol.25, No.12, 1984.
[2] 田中："並列推論マシン", 情処研報, 計算機ｱｰｷﾃｸﾁｬ, 57-3, pp.1-8, 1985.
[3] 片岡,川北,萩原,伊藤："Prolog並列実行マシンの一構成", 昭和60年信学全大, 1720.
[4] J.A.Robinson:"A Machine-Oriented Logic Based on the Resolution Principle", Journal of the ACM, Vol.12, No.1, pp.23-41,1965.
[5] M.S.Paterson, M.N.Wegman:" Linear Unification", J.CSS 16, pp.158-167, 1978.
[6] J.Corbin,M.Bidoit："A REHABILITATION OF ROBINSON'S UNIFICATION ALGOLITHM", Proc. of the IFIP 9th Congress, pp.909-914, 1983.
[7] A.Martelli,G.Rossi:"EFFICIENT UNIFICATION WITH INFINITE TERMS IN LOGIC PROGRAMMING", Proc. of Int. Conf. on FGCS 1984, pp.202-209.
[8] 安浦,大久保,矢島："論理型言語の単一化操作のためのハードウェアアルゴリズム", 信学技報, EC84-67, pp.9-20, 1985.
[9] 湯原,相田,後藤,田中,元岡："高並列推論エンジンPIEの単一化プロセッサと縮退アルゴリズム", 信学技報, EC83-30, pp.47-56, 1983.
[10] 尾内,清水,麻生,益田,松本："リダクション方式並列推論マシンPIM-Rのアーキテクチャ", Proc. of the Logic Programming Conference '85, 2.1, 1985.