ICOT Technical Memorandum: TM-0176

TM-0176

# 並列推論マシンにおける
# 分散型ユニファイア構成法の検討

村上健一郎(NTT)
滝　和男，宮崎敏彦

June, 1986



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

# 並列推論マシンにおける分散型ユニファイア構成法の検討

## A DISTRIBUTED UNIFICATION SCHEME ON THE PARALELL INFERENCE MACHINE

村上健一郎 *　　瀧 和男**　　宮崎敏彦**

Ken-ichiro MURAKAMI　　Kazuo TAKI　　Toshihiko MIYAZAKI

*NTT 電気通信研究所　　**新世代コンピュータ技術開発機構

NTT Electrical Communications Laboratories　　Institute for New Generation Computer Technology

### 1.はじめに

並列推論用カーネルソフトウェアは、協調型問題解決システムを構築する上での基礎となるものである。これまで、論理型言語PROLOG[1]に基づくいくつかの並列処理用論理型言語が提案され、逐次型マシンの上でシミュレータを使った実験が行われている。例えば、PARLOG[2], Concurrent PROLOG[3], GHC(Guarded Horn Clause)[4] などがその例である。これらの言語の実験を並列マシン環境で行ったものとして、ICOTのPIM (Parallel Inference Machine)[5] や、東大のPIE[6] がある。これらのマシンは、データフローモデルやリダクションモデル等を基本的計算モデルとしてマシンアーキテクチャへ反映させたものである。

ここでは、これらのマシンと異なったアプローチをとる。即ち、従来の逐次型推論マシンを基本として、これらを比較的遅いチャネルで結合し、その上で並列処理機構を構築する。本論文では、特に、GHCを計算モデルとした、並列マシン環境下における分散型ユニファイア構成法について述べる。本ユニファイアの特徴は、(1)特別な排他制御機構を用いず、ユニファイ処理のtestフェーズを繰り返すことにより矛盾の発生をなくしていること、(2) PE(Processing Element) 間の通信量が最小となるように考慮していること、および(3)リダクションの木構造を、リダクションIDとして持たせることにより、後追いフェイルを実現していることである。

本論文の後半では、分散型ユニファイアの完全性を検証するためにDEC-2060上にインプリメントした並列推論マシンのシミュレータについて説明する。これは、GHCを縮退させたflat GHCを計算モデルとするものであるが、このシミュレータ自身もGHCで記述されている。

### 2.プロセッサモデル

ユニファイアについて考察する場合のベースとして、プログラムの変更なしに、容易にシステムの拡張が可能なアーキテクチャを想定する。各PEは全体のアーキテクチャに関する情報をもたず、隣接するPEしか意識しない。また、個々のPEは、絶対位置を意識するようなPE番号も持たない。結合方式は、共有メモリを持たず、隣接するPE間をチャネルで結合した方式を想定する。すべてのPE間のコミュニケーションは、このチャネルを通じて行う。ソースプログラムや変数は、PEそれぞれの固有メモリに保持する。一例として、図1

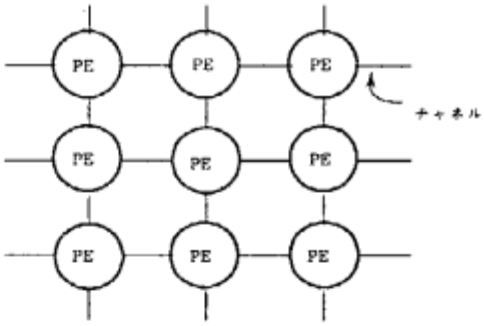
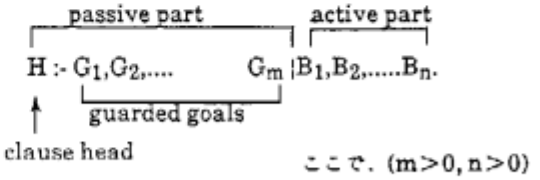



図1. 二次元構成アーキテクチャの例

に2次元メッシュ状構成の場合を示す。図1におけるPE当たりのチャネル数は4本であるが、チャネルの数に制限はない。各PE上では、並列処理用論理型言語GHCによって記述されたマルチプロセスが走行する。

### 3.計算モデルGHCの概要

GHCプログラムは、次の形をしたguarded Horn clauseの有限集合である。



この形式は、DECsystem-10 Prolog[6]の形式に従っている。guarded Horn clauseは、右辺に"|"(guardと呼ばれる。)をもつ。H,G,Bはatomic formulaである。Hはクローズヘッド、$G_i$はガーディドゴール、そして$B_i$はボディゴールと呼ばれる。"|"の前の部分は、パッシブパートとよばれ、その後の部分は、アクティブパートと呼ばれる。

GHCでは、ANDで結ばれた複数のゴールを並列に解く。(AND-parallel) また、1つのゴールに対しては、それを解くいくつかのクローズを並列に処理する。(OR-parallel) 並列に実行されているクローズのうち、さきにガーディドゴールが成功したクローズは、それと並列に実行されている同じヘッドを持つクローズの実行を中止させる。(このことをトラストと呼ぶ。)即ち、パッシブパートの実行が最初に成功したクローズだけが、アクティブパートを実行する。

GHCでは、ユニフィケーションにおいて、以下のような状態が発生した場合に、サスペンドが発生する。

(a) ユニファイする側(呼ぶ側)が変数であり、ユイファイされる側が非変数項の場合

(b) 2つの変数のユニフィケーションで、ユニファイする側の変数のレベルとユイファイされる側の変数が含まれているゴールのレベルが異なっている場合

即ち、呼ばれた側からクローズの呼び出し側への、非変数項の"持ち出し"は、アクティブパートにおいてのみ可能である。(b)の判別を行うため、各変数は、それがどのクローズで生成されたかを示す識別子GSN(Guard System Number) が必要となる。GSNは、プロセス固有のものである。

## 4. 分散型ユニファイアの基本メカニズム

逐次推論マシン上での論理型プログラム実行と、並列推論マシン上での論理型プログラム実行との違いは、異なったPEへのプロセス実行依頼や、異なったPE上にあるオブジェクトどうしのユニフィケーションが存在することである。このため、複数のPEが協調してユニフィケーション処理を行う必要がある。このことから、このユニファイアを分散型ユニファイアと呼ぶ。分散型ユニファイアでは、

(1) 複数のPEに分散したユニファイアの同期やユニフィケーションに伴うサスペンド/リジュームを実現する同期メカニズム。

(2) PE間でオブジェクト(データや命令)を転送するためのルーティングメカニズム。

(3) ユニフィケーション処理を構成する2つのフェーズであるtestとassignの不可分操作が保証されない並列処理環境でも矛盾をおこさない排他制御メカニズム。

が問題となる。

ここでは、これらの問題を明らかにするため、まず、並列推論マシンにおけるユニファイアの処理イメージについて説明する。次に、分散型ユニファイアの基本となるメカニズムについて考察する。

### 4.1. 処理イメージ

#### 4.1.1. 他PEへのタスクディスパッチ

あるPEが、隣接するPEへプロセス実行を依頼する場合には、チャネルを通じてゴールを渡す。ゴールを送ることをexportすると言い、受け取ることを、importすると言う。変数は、それ単独ではexport出来ず、必ずゴールに含んでexportしなければならない。ゴールを受け取ったPEは、それに含まれている変数を、ゴールの通ってきたチャネル番号(その変数を送ったPEを指すポインタ)を含む特別な変数に置き換える。これをチャネル変数と呼ぶ。チャネル変数は、ゴールがexportされた向きとは逆向きのポインタを持つ。

例えば、図2に示すように、$PE_0$のプロセッサの負荷が重いために、g(X)を$PE_1$へexportして実行を依頼すると仮定する。この際、$PE_1$ではXをチャネル変数$X_0$に置き換える。$PE_1$でもプロセッサの負荷が重いために、$g(X_0)$のリダクションを行う際にそのサブゴール$f(X_0)$の実行を$PE_2$へexportして実行を依頼する。$PE_2$では$X_0$をチャネル変数$X_1$に置き換える。このように、チャネル変数のリンクはゴールを渡してゆく際にダイナミックに生成される。このため、プログラムはシステム中のPEの絶対位値を示すPE番号を意識せずにすみ、PEの数が増加しても、もとのプログラムに手を加えることなし

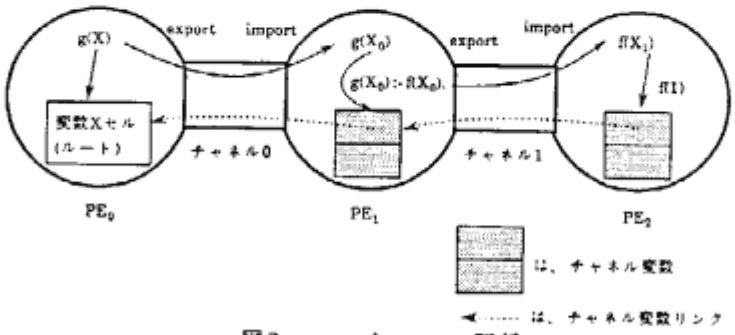



図2. export,importの関係

に走行させることが出来る。ここでは、チャネル変数の実体であるexport元のセルをルートと呼ぶ。

4.1.2.ルーティング

チャネル番号は、異なったPE上にあるチャネル変数をユニファイするためのルーティング情報として使用される。チャネル変数をユニファイする場合、チャネル番号をそのルートへのびるポインタとして使用し、次々と隣接するPEにユニフィケーション要求等を渡す。それに対する応答も、チャネル番号をたどって返される。即ち、PE間にまたがるゴールおよびチャネル変数の関係は、チャネル変数のチャネル番号のみによって定められる。

例えば、図2の$PE_2$において、f($X_1$)とf(1)がユニファイされるとする。$PE_2$のユニファイアは、$X_1$がチャネル変数であることがわかると、ルーティング情報1を見て、最初にチャネル1の方向にチャネル変数Xと1とのユニフィケーション要求命令を発行する。この際、ユニフィケーション要求命令を発生したプロセスは、命令に対する応答が返ってくるまでサスペンドする。要求および応答が速度の遅いチャネルを通り、しかもこの要求の処理がいつ完了するか不明なためである。これを受けた$PE_1$では、チャネル変数$X_0$のルーティング情報0を見て、チャネル0の方向へ、この要求命令を中継する。$PE_0$へユニフィケーション要求命令が到着すると、変数Xがinstantiateされているかどうかのチェックを行う。もし、instantiateされていれば、その値と1との比較を行い、その結果(success/fail)を$PE_2$へ返す。instantiateされていないならば、ユニファイする側が変数でユニファイされる側が非変数項なので、GHCのサスペンドルールに従って、Xがinstantiateされるまで処理をサスペンドする。サスペンドが解けた時に、改めて値の比較を行う。

$PE_2$では、ユニフィケーション要求命令を発行する際に、その中に、変数をうめこむ。この変数は、$PE_0$に到着した時点で、$PE_2$にルートを持つチャネル変数となる。このルーティング情報を使用して、応答も要求命令と同様にして$PE_0$から$PE_2$へ転送される。

4.2.チャネル変数

4.2.1 チャネル変数のリンク

並列処理環境では、複数のユニファイアが同時に動く。このため、同じ変数を対象とする複数のユニフィケーションが同時に発生する可能性がある。対象がチャネル変数である場合、チャネル変数セルは複数のPE上にあるため、複数のユニファイアが異なったPE上のセルに対してアクセスを行うと矛盾が発生する。従って、ユニフィケーション時にアクセスするセルを、一定の場所に決めておく必要がある。これは、チャネル変数リンクの両端のうちどちらかにするのがルーティング上簡単であるが、リンクが更に伸びる可能性があるので、それを最初にexportしたPE内のセル(ルート)とする。

リンク方法には、図3に示す2つの方法が考えられる。方法1では、図2に示したように、各PE上のチャネル変数は、それをexportした1つ手前のPE上のチャネル変数を指すルーティング情報だけをもつ。一方、方式2では、全てのPE上のチャネル変数が、export先からそのルートに至るまでの全てのルーティング情報を持つ。ここでは、方式1を想定して検討を進める。

4.2.2.変数の付加情報

変数は、以下の3つの情報を持つ必要がある。

1) 変数とチャネル変数とを区別するタグ。
2) この変数がinstantiateされていないためにサスペンドしているプロセスをつなぐポインタ。これにより、バインドフック(後述)を実現する。
3) 変数どうしのユニフィーションを行う場合に、サスペンドするかどうかを判別するためのGSN。GSNは、その変数が属するガードをさすポインタである。

チャネル変数は、これらの情報に加えて、2つの特別な情報を持つ。以下にその情報を示す。

4) ルーティング情報。このチャネル変数がどのPEからexportされたかを示す。PE間通信のルーティングに使用する。
5) 問合せ中表示フラグ。あるユニファイアがチャネル変数の値の問合せ中のためにサスペンドしている時、同一PE内の他のユニファイアが再び同じチャネル変数に対して問合せを行わないようにするための情報である。

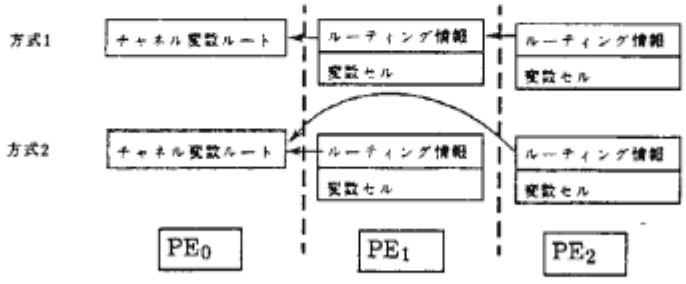


図3.チャネル変数のリンク

これにより、チャネルの負荷および、問合せのオーバヘッドを抑制する。

### 4.3. 排他制御

チャネル変数を対象とするユニフィケーションでは、testのフェーズで、そのチャネル変数の値をチェックするための、チャネルを渡るアクセスが発生する。チャネルを渡るアクセスを行う場合には、アクセス要求に対する応答が帰るまでの時間が長くしかも不定であるため、実行中のプロセスをサスペンドする。従って、testのフェーズとassignのフェーズが分割される。この時、2つ以上のプロセスで変数を共有していた場合、testのフェーズではundefであったものが、別のプロセスによってinstantiateされ、assignを行う時点で矛盾が発生することも有り得る。一般には、このような場合、test&setによる排他制御を行うが、このための制御機構が必要となり、制御も複雑になる。本方式では、test&setによる排他制御を行わず、矛盾の発生する恐れのある場合にはassignフェーズの直前に再testを行う。再testにおいてtestのフェーズとの矛盾を検出した場合には、ユニフィケーションを失敗させたり、再testの結果に従ってユニフィケーションをやり直す。

### 4.4. 同期メカニズム

分散型のユニファイアでは、以下の2つの場合に、サスペンドが発生する。

(1) GHCのサスペンド条件に該当する場合

(2) チャネルを渡るアクセスが発生した場合

処理系の簡素化を計るため、これら両方の場合に、バインドフックを使用する。バインドフックとは、特定の変数がinstantiateされた時に、予め登録していたプロセスが実行されるというものである。ユニフィケーションの対象の組み合わせによってサスペンドした場合には、サスペンドしたプロセスを、それがinstantiateされた時にサスペンドが解けることになる変数のバインドフックとして登録する。

他PEに対するチャネル変数のユニフィケーション要求やアクセス要求が発生した場合には、ユニファイアは、操作要求を相手のPEに送る。この際、応答を受けるための変数を用意し、操作要求に含めてexportする。この変数がinstantiateされたときに実行されるプロセスとして自分自身を登録し、サスペンドする。相手のPEは、この変数に結果をバインドすることによって応答を返すので、登録して置いたユニファイアがリジュームされる。

## 5. ユニフィケーション処理方式各論

第4章では、分散型ユニファイアの基本となるメカニズムについて説明した。本章では、それらのメカニズムを使用して分散型ユニファイアを実現するための処理方式について述べる。

### 5.1 分散型ユニファイアのtestフェーズ

逐次推論マシンにおけるユニフィケーションは、変数がundefかどうかを見るtestのフェーズとundefであった場合に値を書き込むassignのフェーズとに分けられる。assignのフェーズでは、testの結果によって、ユニフィケーションの対象の組み合わせを以下の3つの場合に分け、それぞれで異なった処理を行う。

(1) どちらも非変数項の場合

(2) 片方が変数でもう片方が非変数項の場合

(3) 両方が変数の場合

並列推論マシンにおけるユニフィケーションも逐次推論マシンと同様に、testのフェーズとassignのフェーズからなる。ただし、上記の3つの場合に加えて、

(4) 片方がチャネル変数でもう片方が非変数項の場合

(5) 片方がチャネル変数でもう片方が変数の場合

(6) 両方がチャネル変数の場合

の3つを考慮しなければならない。チャネル変数のtestを行う際には、そのチャネル変数のルートをアクセスするのではなく、ユニフィケーションの発生したPEにあるローカルなチャネル変数用セルをアクセスする。このtestにより、(1)から(6)のうち、どの場合にあてはまるのかを決定し、それぞれの処理を行う。以後のセクションでは、並列推論マシン固有の問題である(4)から(6)までのそれぞれの場合にわけて、再testとassignフェーズにおける処理方式を検討する。

### 5.2 チャネル変数と非変数項の場合

#### 5.2.1. PE間の通信量に関する考察

testの結果、ユニフィケーションの発生したPE上にあるチャネル変数セルがinstantiateされていないとわかっても、そのチャネル変数のルートがinstantiateされていないとは限らない。従って、チャネル変数のルートを持つPEに対して値の問合せを行った後に、どのような処理を行うかを決定しなければならない。これを行うには、2つの方法が考えられる。

(1) ユニフィケーションを発生したPEが主体となり、チャネル変数のルートのあるPEへ値の問合せを行った後に、どのような処理を行うかを決定する。

(2) ユニフィケーションを発生したPEは、チャネル変数のルートのあるPEへユニフィケーションを依頼し、依頼されたPEが主体となり、ユニフィケーションの処理を行う。依頼されたPEは、結果のみを返す。

(1)の方法では、①値の問合せ、②それに対する応答、③代入が発生した場合の処理の依頼、④それに対する応答の4回の通信が必要となる。これに対

して、(2)では、全てのユニフィケーションの処理をチャネル変数のルートのあるPEが行うため、ユニフィケーションを発生したPEとの通信は、ユニフィケーションの依頼と結果の報告の2回に収まる。従って、チャネル変数と非変数項のユニフィケーションは、そのチャネル変数のルートがあるPEへ依頼する。

5.2.2. 処理シーケンス

例を示して処理シーケンスを説明する。例えば、チャネル変数Xと1とのユニフィケーションを考える。(図4) ユニフィケーションの発生したPEは$PE_1$であり、Xのルートは、$PE_0$上にある。ユニフィケーションに際して行われるPE間の通信および処理シーケンスを図5に示す。

[CASE 1] ユニファイする側が非変数項で、ユニファイされる側がチャネル変数の場合

1) まず、$PE_1$は$PE_0$に対して、Xと1とのユニフィケーションを依頼する。(図5の①) それと同時に、応答を受けるために使用する変数Rのバインドフックの対象としてユニフィケーションプロセス自身を登録し、相手のPEから応答が返されてくるまでサスペンドする。

2) ユニフィケーションの依頼を受けた$PE_0$では、自PEでユニフィケーションが発生した場合と全く同じ様にtestのフェーズからユニフィケーションを開始する。即ち、Xのテストを行い、Xがundefであれば、Xに1を代入し、$PE_1$に対してsuccessを通知する。(図5の②') Xが既にinstantiateされていた場合には、その値と1との比較を行う。一致すれば$PE_1$に対しsuccessを返し(図5の②")、そうでなければinstantiateされていた値を通知する。(図5の②"")

3) 応答を受けた$PE_1$では、Rに結果が代入されると同時に、サスペンドしていたプロセスがリジュームされる。リジュームされたプロセスは、Rをチェックし、successであれば$PE_1$上のXのセルへ1を代入し、ユニフィケーションは成功する。それ以外の場合には、Rの値をチャネル変数セルへ代入して、ユニフィケーションは失敗する。代入を行うのは、再度チャネル変数を対象とするユニフィケーションが発生した場合でも、そのルートを持つPEへユニフィケーション要求等を送らずに処理を行えるようにするためである。

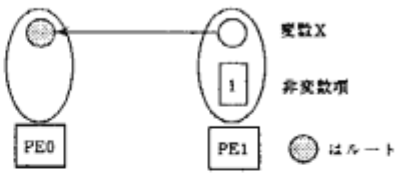



図4. チャネル変数Xと1とのユニフィケーション

[CASE 2] ユニファイする側がチャネル変数で、ユニファイされる側が非変数項の場合

1) 最初の処理は、CASE1の1)と同じである。

2) $PE_0$ではXのテストを行い、Xがundefであれば、Xがinstantiateされるまで GHCのルールに従ってサスペンドする。Xが既にinstantiateされていた場合、および、Xがinstantiateされてサスペンドが解けた場合には、その値と1との比較を行う。一致すれば$PE_1$に対しsuccessを返し(図5の②"")、そうでなければinstantiateされていた値を通知する。(図5の②"")

3) 応答を受けた$PE_1$では、Rに結果が代入されると同時に、サスペンドしていたプロセスがリジュームされる。リジュームされたプロセスは、Rをチェックし、successであればそのユニフィケーションは成功する。それ以外の場合に、ユニフィケーションは失敗する。

ここでは、チャネル変数を対象とするユニフィケーションは、必ず、そのルートをもつPEへ依頼するため、チャネル変数のtestとassignのフェーズが不可分操作として実行でき、再testは必要ない。

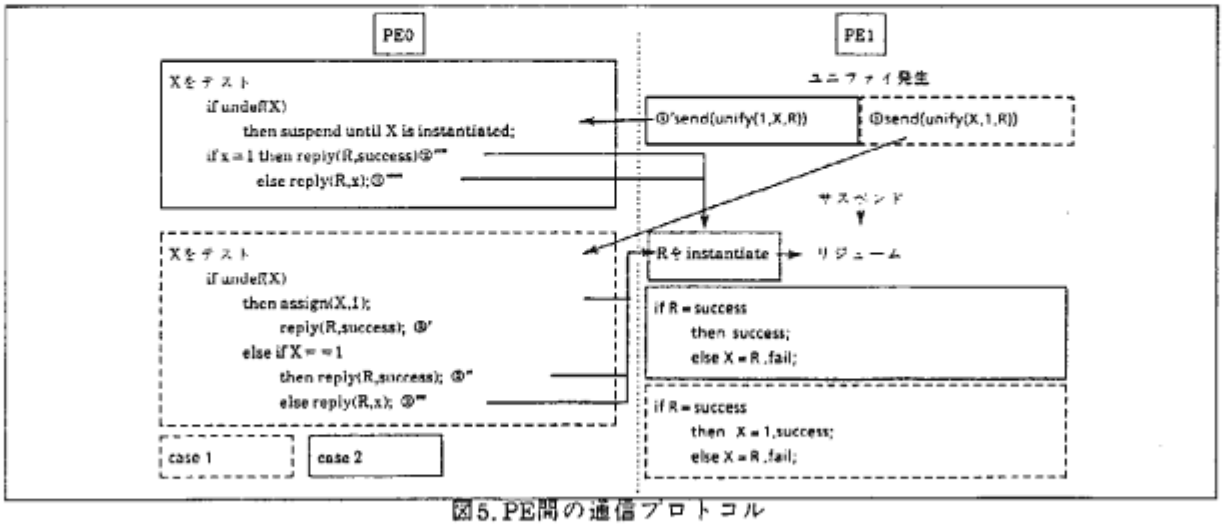



図5. PE間の通信プロトコル

### 5.3 チャネル変数と非チャネル変数の場合

#### 5.3.1. PE間の通信量に関する考察

ここでは、サスペンドとリジュームに着目する。GHCのルールに従ってサスペンドが発生する場合には、その原因となったinstantiateされていない変数のバインドフックとして、リジュームさせるべきプロセスを登録しなければならない。バインドフック対象の変数が、サスペンドが発生したPEと別のPE上にあれば、登録のためのメッセージとリジュームを通知するメッセージがチャネルを通ることになる。これを防止するため、サスペンド原因となった変数と同じPE(即ち、ユニファイする側の(チャネル)変数をもつPE)上でプロセスが起こるように考慮する。

#### 5.3.2. 処理シーケンス

$PE_0$上にルートを持つチャネル変数Xと$PE_1$上の変数Yとのユニフィケーションを例にとって処理シーケンスを説明する。(図6) ユニフィケーションの発生したPEは$PE_1$である。ユニフィケーションに際して行われるPE間の通信および処理シーケンスを図7に示す。ユニフィケーションの処理は2つの場合に分けられる。

**[CASE1]** ユニファイするのがXで、ユニファイされるのがYの場合(unify[X,Y])

1) まず、$PE_1$は$PE_0$に対して、XとYとのユニフィケーションを要求する。(図7の①) このユニフィケーション命令によって変数Yは$PE_0$にexportされ、$PE_1$にルートを持つ新たなチャネル変数となる。また、応答を受けるために使用する変数Rのバインドフックの対象として、リジュームすべきプロセスをつなぐ。

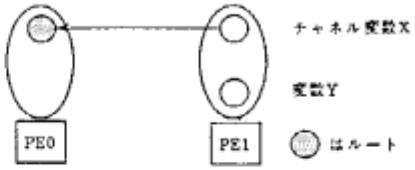


図6. チャネル変数Xと非変数チャネルY とのユニフィケーション

2) $PE_0$では、Xのテストを行う。Xが既にinstantiateされていれば、$PE_1$に対してその値を通知する。(図7の②') Xがundefの場合には、Xの変数レベルとYを含んでいるゴールレベルの比較を行う。一致すれば$PE_1$に対しXがundefであるという情報を返す。(図7の②") そうでなければXがinstantiateされるまでサスペンドする。リジューム後は、instantiateされた値を$PE_1$に対して通知する。(図7の②"')
3) 応答を受けた$PE_1$では、変数Rに結果が代入されると同時に、サスペンドしていたプロセスがリジュームされる。リジュームされたプロセスは、変数Rをチェックし、undefであれば$PE_1$上のXのセルとYのセルとをユニファイする。それ以外の場合には、まずXに値を代入する。その後、Yとこの値とのユニファイを行う。この時、Yが別のユニファイアによって、異なった値にinstantiateされているためにfailすることもある。

**[CASE2]** ユニファイするのがYで、ユニファイされるのがXの場合(unify[Y,X])

1) $PE_1$はXの転送命令だけを$PE_0$に送る。(図7の①') 応答を受けるために使用する変数Rのバインドフック対象として、リジュームすべきプロセス

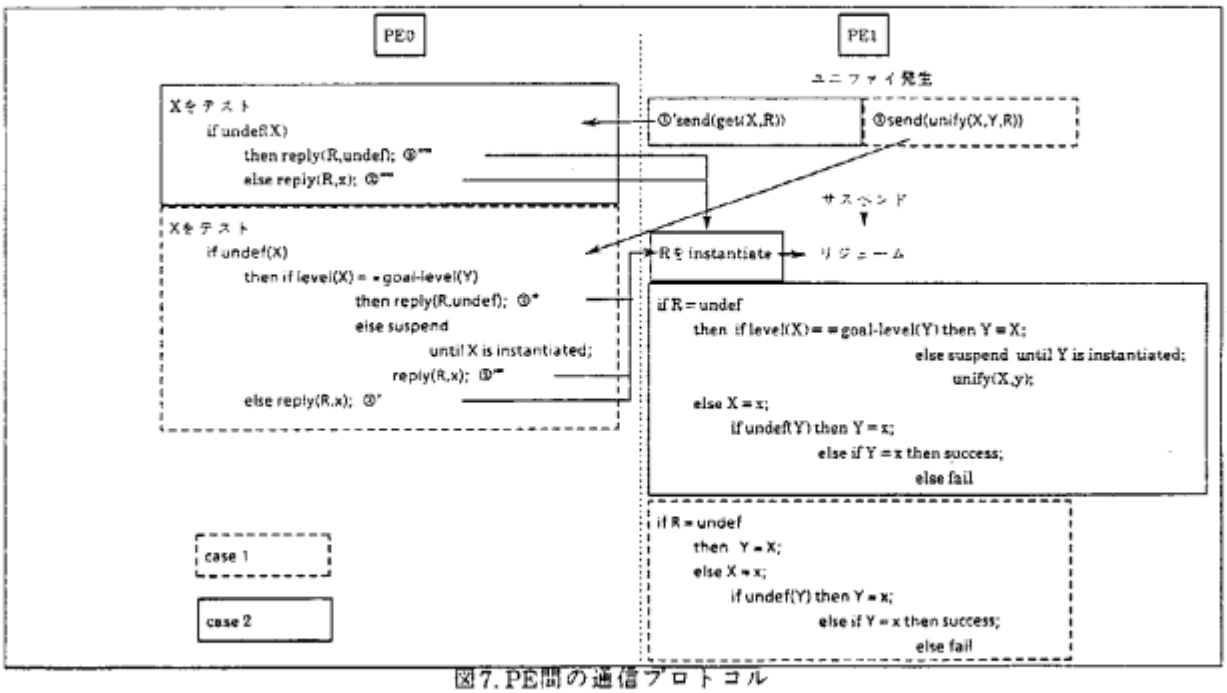


図7. PE間の通信プロトコル

を登録する。このサスペンドが解けるのは、Rに値が返された時である。

2) Xのルートをチェックして、その結果を$PE_1$へ返す。(図7の②"")

3) Rに値が返され、サスペンドが解けるとRの値をチェックする。変数Rの値がundefの場合、Xのレベルと Yのゴールレベルとを比較する。その結果、レベルが同じであれば、XとYとのユニフィケーションを行う。レベルが異なれば、Yがinstantiateされるまでサスペンドし、instantiateされたあと、Xとその値とをユニファイする。"undef"以外であれば、その非変数項と変数とのユニフィケーションをtestのフェーズから始める。testのフェーズからユニフィケーションを始めるのは、チャネル変数の値を尋ねている間に変数が別のユニファイアによってinstantiateされていることが有り得るからである。

5.4 チャネル変数どうしの場合

5.4.1. 処理概要

チャネル変数どうしのユニフィケーションで最も問題となるのは、バインド後の新ルートの位置である。ルートを一定の場所に決めなければ、複数のユニファイアが並列に動くため、それらの間で矛盾が発生する。ユニフィケーション処理では、3つのPE、即ち、ユニフィケーションを発生したPE、ルートを持つ2つのPEの間で通信を行わなければならない。この通信量を少なくし、通信距離を短くするために、ルートを持つ2つのPEの中間にあるユニフィケーションを発生したPEを主体としてユニフィケーションを行う。この方式では、2つのチャネル変数に対するバインドフックがすでに存在していた場合、新たなルートがinstantiateされた時に、それらのバインドフックの対象として登録されていたプロセスをどのようにして起動するかが問題となる。この問題は、新ルートと旧ルートの間で、ルートのinstantiateが発生した時に、互いにその値を知らせることで解決する。

ユニフィケーションは、先ず2つのチャネル変数のルートをもつPEへ、その値を尋ねることから始める。その結果、一方がinstantiateされていれば、5.3.に述べたチャネル変数と非変数項とのユニフィケーションを行う。両方がinstantiateされている場合には、値の比較を行う。両方がundefであった場合、新たなルートにするためのセルをつくり、そのバインドフックとして、それがinstantiateされた場合に2つの旧ルートへ値を知らせるためのプロセスを登録する。逆に、2つの旧ルートがinstantiateされた場合を考え、それから伸びているチャネル変数のパス上にある全てのPEへブロードキャスト命令を送るプロセスを2つの旧ルートのバインドフックとして登録する。

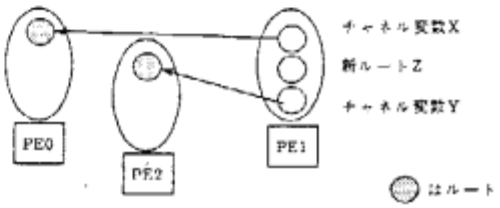



図8. チャネル変数X,Yのユニファイ

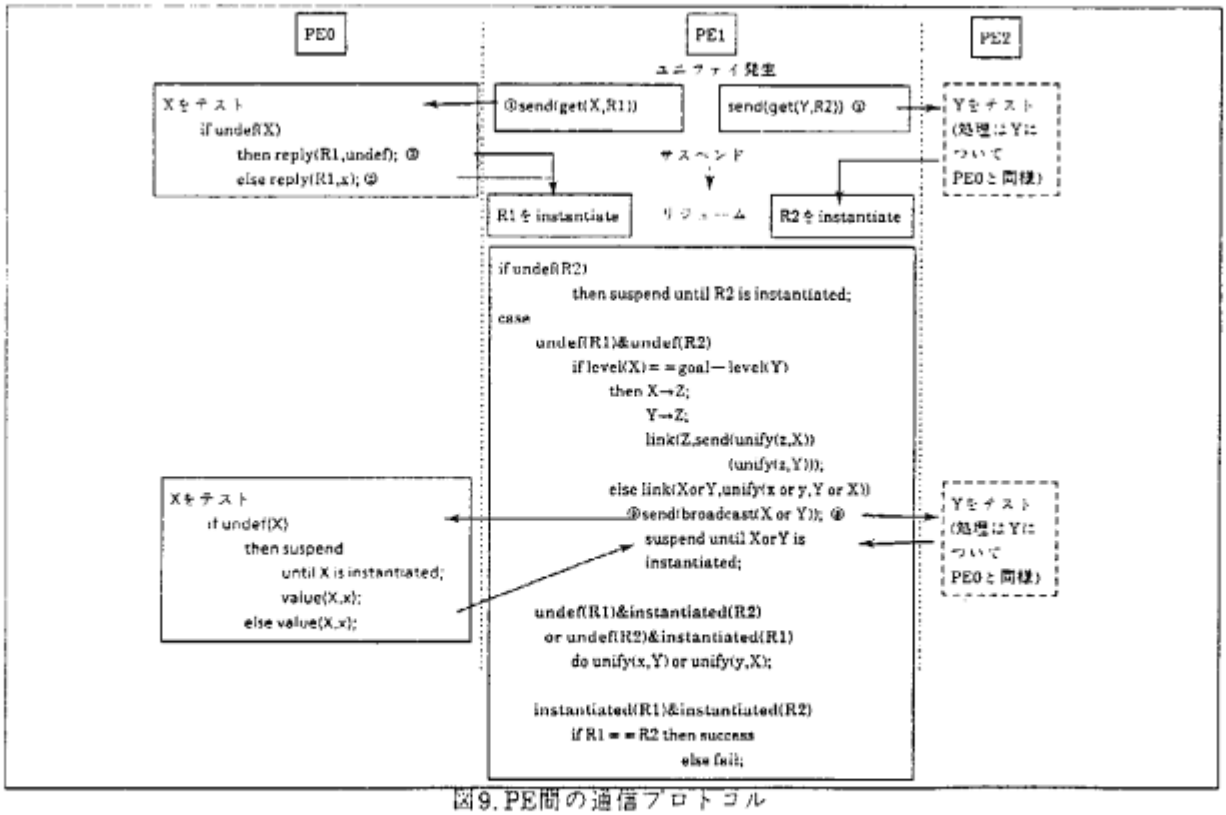



図9. PE間の通信プロトコル

| | Xがundef | Xがinstantiated |
|---|---|---|
| Yがundef | 2つのチャネル変数Xのレベル,Yを含んでいるゴールレベルとを比較する。<br><br>1)一致すれば、新たなルートZをPE1上につくり、PE1上のX,Yの変数セルからポイントする。Zのサスペンドリストには、PE0のチャネル変数XとPE2上のチャネル変数Yそれぞれのノードへのbroadcast命令をつなぐ。Zがinstantiateされてときにこのプロセスは起動される。<br><br>2)変数レベルが一致しない場合、サスペンドの原因となったチャネル変数のPE1上ある変数セルのサスペンドリストへ、サスペンドが解けた時に実行するプロセスをつなぐ。同時に、そのチャネル変数のルートをもつPEへ、broadcastの要求命令を出す。(③)従って、このルートがinstantioateされた時に、その値がPE1にも転送され、チャネル変数とinstantiateされた変数とのユニファイが行われる。 | Xに値を代入して、チャネル変数Yとinstantiateされた変数とのユニファイを行う。 |
| Yがinstantiated | Yに値を代入して、チャネル変数Xとinstantiateされた変数とのユニファイを行う。 | 2つの値を比較する。 |

表2. 二つのチャネル変数のユニフィケーション処理

5.4.2. 処理シーケンス

$PE_0$上にルートを持つチャネル変数Xと$PE_2$上にルートを持つチャネル変数Yとのユニフィケーションを考える。(図8) ユニフィケーションの発生したPEは$PE_1$である。ユニフィケーションに際して行われるPE間の通信および処理シーケンスを図9に示す。チャネル変数どうしのユニフィケーションでは、$PE_1$上に新たなセルZを用意する。Zがinstantiateされた場合に、$PE_0$および$PE_2$のルートのプロセスリストにつながっているプロセスも起動しなければならない。そこで、Zには、broadcast命令をリンクしておき、Zがinstantiateされた時点で、$PE_0$,$PE_2$にある2つのチャネル変数のルートもinstantiateする。

1)まず、$PE_0$,$PE_2$に対して、各変数の値の転送命令を出す。(図9の①) プロセスは、この命令に対する応答が帰ってくるまでに、サスペンドする。また、応答を受けるための変数$R_1$または$R_2$のどちらかのサスペンドリストへ、応答が帰ってきたあとに起動したいプロセスをつないでおく。

2)$PE_0$,$PE_1$は、それぞれチャネル変数X,チャネル変数Yをテストし、その値を$PE_0$へ返す。undefの場合には、"undef"の情報を返す。(図9の②)

3)$PE_0$上の変数$R_1$または$R_2$がinstantiateされると、1)でサスペンドしていたプロセスが起動され、$R_1$にプロセスがつながれていたら、$R_2$を見にゆき、undefであれば、再びプロセスを$R_2$のサスペンドリストへつなぐ。既にinstantiateされていれば、$R_1$と$R_2$の値の組み合わせによって表2の処理を行う。

5.4必要なプリミティブ

以上5.1から5.4までの処理で必要となるプリミティブをまとめると、以下のようになる。

1)チャネル変数の値を要求するコマンド
get(Channel-Variable,Return-Value)
Return-Valueは、応答を返す際のルーティング情報としても使用する。

2)チャネル変数のbroadcastを要求するコマンド
broadcast(Channel-Variable)
Channel-Variableがinstansiateされた時に、その値をbroadcastする。

3)チャネル変数の値を転送するコマンド
reply(Return-Value,Value)
1)の要求に対する応答である。

4)ゴールを転送するコマンド
send-goal(Goal,Result)
ResultはGoalの実行結果を通知する際、ルーティングにも使用される。

5)ゴールの実行結果を転送するコマンド
reply-result(Result)

6)チャネル変数の値をbroadcastするコマンド
value(Old-Channel-Variable,Value)
この命令は、2)に対する応答である。

7)ユニフィケーションを要求するコマンド
unify(Channel-Variable,Variable)

6. 後追いフェイルの実現方式

ゴールが他のPEへ分散リダクションした場合、そのリダクション先でfailが起これば、リダクション元へfailを伝えると共に、必ずその祖先及びANDで結ばれた祖先とその子孫のゴールに対して、実行中止を伝えなければならない。このようなfailの伝達を後追いfailとよぶ。(図10) failを起こしたゴールの祖先及びANDで結ばれた祖先とその子孫の

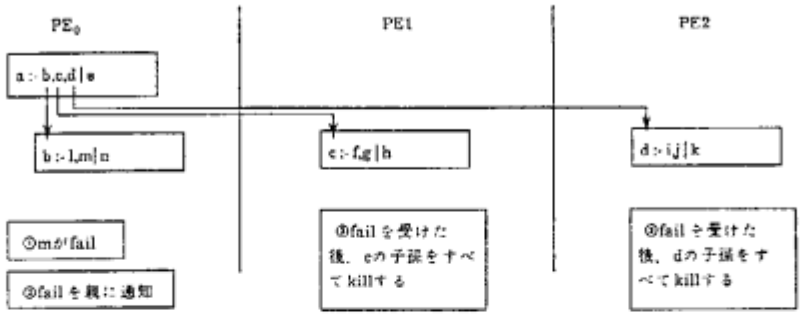


図10. 後追いfail

ゴールを、すべてのPE上で積極的に探すことはオーバヘッドが大きい。しかも、failが発生して実行中止命令が目的のゴールに到達するまで、時間がかかるため、停止までに、かなりの実行が進む恐れがある。そこで、中止すべきgoalを各PEへ知らせ、各PE上のディスパッチャが、それに該当するgoalをプロセスキューから取り出した時点で、それを破棄する。

並列実行には、AND並列とOR並列とがある。AND並列では、並列に実行しているゴールのうち、1つでもfailすれば、外のゴールの実行も中止しなければならない。また、OR並列の場合、GHCでは、どれかのゴールがガードを越えたならば、ほかのゴールの実行を中止しなければならない。したがって、GHCにおいては、次のような並列実行を管理するモジュールが必要となる。

1)guard manager

OR並列の管理を行う。OR並列で実行する1つのゴール群につき、1つ存在する。特にガードの管理を行う。

2)and goals manager

AND並列管理を行う。ゴールをリダクションする場合、1つのAND並列群につき、1つ存在する。

これらの関係を図11に示す。各AND並列は、それ

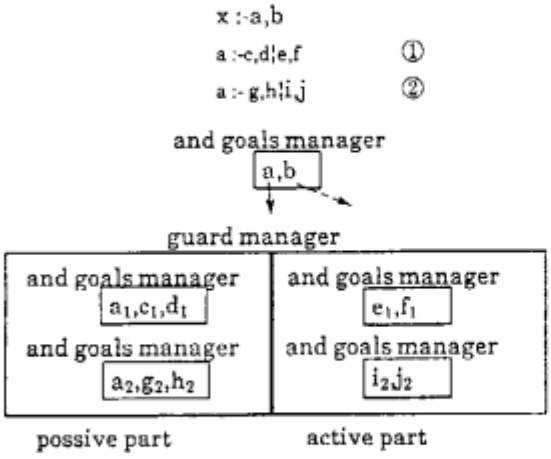


図11. マネージャとゴールの関係

ぞれのand goals managerによって管理される。それらのand goals managerは、更にその上位のguard managerによって管理される。guard managerは、その上位のand goals managerによって管理される。また、guard managerのactive partでは、実際には、1つのANDゴールしか実行されない。

これらのマネージャおよびゴールは、次のように処理を進める。

1) 同じhead clauseを持つ各clause(OR並列ゴール)へ、それぞれ異なったリダクションIDを付与する。従って、同じand goals managerで管理されているゴールは、同じIDを持つ。IDには、リダクションを行ったPE番号を含ませ、システムで一意に識別可能とする。例えば、

a :- b, c, d

では、b,c,dは、同じリダクションIDを持つ。図11の例では、①、②のclauseは、それぞれID1, ID2を持つ。

2) 各ゴールは、実行結果をそのゴールを管理するand goals managerへ通知する。

3) and goals managerは、通知された結果がfailであれば、AND並列で実行しているゴールをkillするため、それらがexportされたPEに対してリダクションIDを通知する。各PEのプロセスディスパッチャでは、プロセスをディスパッチするごとにそのリダクションIDをチェックし、該当するゴールは、そのまま捨てる。またその上位にあるguard managerにも結果を通知する。possive part実行の結果は、and goals managerからguard managerへ通知される。

4) and goals managerからguard managerへの結果の通知によって、最初にガードを越えたguarded clauseが出来ると、その他のOR並列ゴールのリダクションIDをOR並列ゴールをexportしたPEに対して送る。プロセスディスパッチャでは、3)と同じ処理がなされる。guard managerへのすべての通知がfailであれば、さらにひとつ上の、and goals managerにfailを通知する。

尚、各ゴールは、どのゴールから生成されたかを示すIDリストを持っていなければならない。このリストの中に一つでも中止するIDをもっているゴールは、破棄される。

## 7.並列推論マシンシミュレータ

分散型ユニファイア構成法を検討/検証するため、DEC-2060上に並列推論マシンシミュレータを作成した。シミュレータは、TOPS-20上のGHC(Guarded Horn Clause)で記述されている。現在、最小の機能を持ったシミュレータ上で、2台のPEがシミュレートされている。

### 7.1シミュレータの構成

図12に示すように、各プロセッサをGHCの1プロセスによってシミュレートする。シミュレートするプロセッサ上では、次の節で述べる、Flat-GHCで記述されたプログラムが、マルチプロセスで走行する。

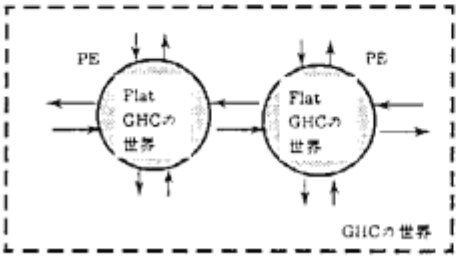


図12.対応するソフトウェアモジュール

本シミュレータでは、2つの変数をPE間のチャネルとして使用する。2本のチャネルは、それぞれ上り、下り専用のチャネルである。これは、チャネルの管理を容易にするためである。送り手のPEからは、受け手の所望の情報だけではなく、次にチャネルとして使用すべき変数を送る。チャネル変数のリンクは、第4章で述べたようにルートへ向かう単方向リンクであり、各PE上で変数バッファを持つ。シミュレータを簡単にするため、第5章で述べた分散型ユニファイアとは、以下の部分が異なっている。

(1) チャネル変数をinstantiateするPEは、それをexportしたPEのみである。

(2) チャネル変数同志のユニフィケーションをインプリメントしていない。

(3) サスペンドしたプロセスのリジュームには、バインドフックではなく、常に変数を監視するビジーウェイト方式を使用する。

### 7.2 チャネル変数テーブル

既存の逐次推論マシンを接続することを考えた場合、セルアドレスを直接渡すためには、PE番号を識別するためのアドレスを拡張することが必要である。しかし、アドレスの拡張は、システムの基本部分を大きく変更しなければならない上、PEの台数が多い場合には、アドレスの拡張が困難である。本シミュレータでは、既存の逐次推論マシンを接続することを想定し、拡張アドレスを使用しない。従って、論理の世界にあるチャネル変数を、非論理の世界であるチャネルを通して、再び別PE上の論理世界へ持ちこまなければならない。シミュレータでは、チャネルを通る変数は、すべて変数を一意に識別する変数ID(IDentifier)に変換して転送し、受け側のPEで再び変数に変換する。チャネル変数テーブルはこれらのチャネル変数と変数IDとの変換、およびチャネル変数の管理を行うためのものである。

PEへ入ってくるチャネル変数やPEの外へ出ていくチャネル変数のリンクはチャネル変数テーブルですべて管理される。

チャネル変数テーブルは、チャネルの接続されている両方のPE上で、チャネルマネージャによって管理されている。チャネルマネージャは、senderとreceiverの2つのモジュールから成る。senderは、ゴールを転送する場合だけではなく、チャネル変数にユニファイされた値を転送する際にもリスト等に含まれている変数を変換する。

### 7.3 チャネル変数

シミュレータ内部では、チャネル変数は($cha(Table_number, Value), Access_flag)のように表現される。$chaは、チャネル変数を表すIDであり、Table_numberはチャネルごとに持っている、チャネル変数と変数IDとを変換するテーブルの番号(即ち、チャネル番号)である。また、Valueは変数セルであり、Access_flagは、チャネル変数の問合せ中を示すフラグである。

一度importした変数を再度exportした場合には、チャネル変数の値が更にチャネル変数となる。たとえば、チャネル変数($cha(Table_number0, Value), Access_flag)を再度exportした場合には、それをimportしたPE上で、($cha(Table_number1, ($cha(Table_number0, Value)), Access_flag)の形になる。このようにして、チャネル変数のリンクを張る。リンクの向きは、ゴールがexportされた向きと逆である。

### 7.4 計算モデルFlat-GHC

#### 7.4.1Flat-GHCとGHC

GHCに比較してFlat-GHCは以下の点が異なっている。

(1) guarded goalとして他のuser definedのゴールを呼ぶようなゴールは書けない。(即ち、他ゴールとの変数のユニフィケーションを許さない。) GHCでは、guarded goal内でユニフィケーションを行う場合、実行中のゴールをサスペンドするかどうかを決定するために、GSNをチェックする必要があった。Flat-GHCでは、guarded goalが他のhead goalにユニファイされることはなく、system definedだけである。このため、guarded goalにundefである変数が含まれれば、そのゴールをサスペンドするという簡単な処理だけを行えばよい。

(2) (1)のため、guarded goalが更にリダクションされて、他のPEへ制御は分散しない。従って、

GHCで問題となる後追いfail問題を考える必要がない。

### 7.4.2 Flat-GHCのマルチプロセッササポート

Flat-GHCでは、マルチプロセッサ環境を提供するために、GHCに比較して以下のメカニズムが拡張されている。

(1) 他のPEへゴールを送る機能

負荷制御モジュール(Flat-GHCの一部ではなく、その上に構築されるべきポリシーの部分)の指示により、指示されたPEへゴールを転送する。現在のシミュレータでは、一度に一つのゴールを転送する。複数のゴールを転送したければ、複数回に分けて転送する。

(1)は、陽にFlat-GHCユーザが意識しなければならないが、以下(2)から(3)の機能は、implicitにFlat-GHC内部で行われる機能である。

(2) (1)で送ったゴールの実行結果を戻す機能

本シミュレータのFlat-GHCでは、top level以外でのゴールの転送を許していないため、この機能はインプリメントしていない。

(3) exportされたチャネル変数の値をimport側PEへ持ち出す機能

import側のPEからチャネル変数の値の要求があれば、その値をexport側から転送する。

## 7.5 Flat-GHCシステムの構造

### 7.5.1. 実行方式

本シミュレータでは、図13に示すように、各

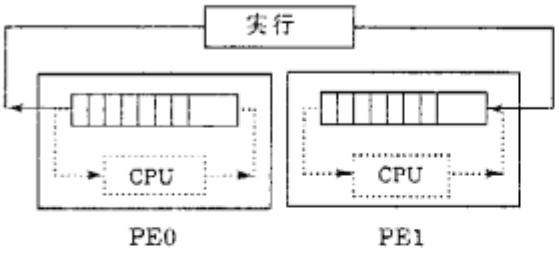



図13. 実行方式

PEごとに実行すべきゴールのキューを持っている。あるPEのキュー内のゴールを一回りすると、別のPEのキュー内のゴールの実行に移る。キューから取り出す際には、サスペンド中のゴールであれば、その原因となった変数をチェックし、すでにinstantiateされていれば実行を再開する。instantiateされていない場合には、そのままキューの最後につける。また、取り出したゴールと並列に実行されている他のORゴールがすでにトラストしていれば、そのまま捨てる。ゴールの実行中にサスペンドが発生した場合には、そのゴールをキューの最後につける。

このようにして、与えられたゴールが解けるかもしくは、failが発生するまで実行を続ける。

### 7.5.2 チャネルを通るパケットのフォーマット

パケットの形式を以下に示す。

{ Packet_Kind, Information, Next_Channel_Variable}

Packet_Kindは送られるパケットの種類をあらわす。Informationは真のデータ、Next_Channel_Variableは次にチャネルとして使用する変数である。本シミュレータでは、同一変数の再利用が不可能であるため、3番目のアーギュメントが必要である。表3に各フィールドの内容を示す。

| Packet_Kind | パケットの意味 | Information |
|---|---|---|
| value_request | 変数値の要求 | 変数ID |
| reply_value | value_requestに対する応答 | (変数ID, variable/instance,変数値) |
| send_goal | goalの実行を要求 | goal |
| reply_result | send_goalの実行結果 | success/fail |

表3. 各フィールドの内容

### 7.5.3. シミュレータのモジュール構成

シミュレータは、5つのモジュールから構成される。これらモジュールの機能を表4に示す。

| モジュール名 | 機能 |
|---|---|
| SOLVER | 1)ゴールの実行<br>2)あるORプロセスがcommitした場合の他のORプロセスの刈り取り<br>3)ORプロセスの実行結果のチェック<br>4)他のPEへ転送命令を出してサスペンドしたゴールが実行可能かどうかのチェック |
| UNIFIER | 1)変数をチェックし、実行中のゴールをサスペンドするかどうか決定<br>2)変数をバインドする |
| SCHDULER | 1)UNIFIERのチェック結果により、ゴールのスケジュールを行う。チャネル変数とのUNIFIERでサスペンドする場合には、SENDERに対し、変数をexportしたPEから、値を取り出してくることを依頼する。 |
| SENDER | 1)送り側チャネルを使用して、コマンドパケットを、依頼元のゴールから指定されたPEへ転送する。<br>2)チャネルを渡る変数のテーブル管理及び変数IDへの変換 |
| RECEIVER | 1)受け側チャネルを監視し、コマンドパケットを受信する。<br>2)チャネルを渡ってくる変数IDのテーブル管理および、変数への変換 |

表4. モジュールの機能

## 7.6. 簡単な実行例

付録1にストリームをマージし、それを出力するプログラムを示す。$PE_0$で2つのストリームをマージする。$PE_1$では、マージが行われるごとに、ストリームをCRT上に出力する。実行時のCRT出力を付録2に示す。

## 8. おわりに

従来の逐次推論マシンを比較的遅いチャネルで結合した並列推論マシン上での分散型ユニファイアについて検討した。PE間通信の少ないユニフィケーションアルゴリズムを示し、陽に排他制御を行うことなく複数のtestフェーズによってユニフィケーション処理の矛盾を防止するアルゴリズムを説明した。また、リダクションIDを用いて、後追いfailを実現する方式についても説明した。今後、このユニファイア上に構築される負荷制御方式について検討をすすめるつもりである。


## 謝辞

本論文をまとめるにあたり、この研究の機会を与えていただいたNTT基礎研究所情報通信基礎研究部第一研究室雨宮真室長、新世代コンピュータ技術開発機構研究所第二研究室横井俊夫室長、第四研究室内田俊一室長、ほか関係各位に感謝いたします。


## [参考文献]


[1]David H D Warren: Impementing PROLOG - compiling predicate logic programs Volume 1,2 ,, DAI Research Report No.39(1982)

[2]Clark, K.L. and Gregory, S: PARLOG: Parallel Programming in Logic, Research Report DOC 84/4, Dept. of Computing, Imperial College, London, (1984)

[3]Kazunori Ueda: Concurrent Prolog Re-examined, ICOT Technical Report TR-102 (1985)

[4] Kazunori Ueda: Guarded Horn Clauses, ICOT Technical Report TR-103 (1985)

[5]伊藤他: データフロー方式並列推論マシンのアーキテクチャ, Proceedings of the Logic Programming Conference (1984)

[6] M.Yuhara; A Unify Processor Pilot Machine for PIE, Proceedings of the Logic Programming Conference (1984)

[7]Toshihiko Miyazaki: A Sequential Impemetation of Concurrent Prolog Interpreter,(ICOT internal memo)(1985)

[8]村上;マルチプロセッサにおけるユニファイア構成法の検討、ICOT内部メモ(1985)


## 付録1 ソースプログラム

```
merge([A|X], Y, Z) :- true | Z=[A|W], merge(Y, X, W).
merge(X, [A|Y], Z) :- true | Z=[A|W], merge(Y, X, W).
merge([], Y, Z) :- true | Z=Y.
merge(X, [], Z) :- true | Z=X.

out([]).
out([A|X]) :- write(A),out(X).

test :- merge([1,2],[a,b],Result),out(Result).
```

## 付録2 実行結果(一部)

-->は、ゴールが実行されたことを示す。

G-->は、サスペンドされていたゴールがリジュームされたことを示す。

Wt-->は、ゴールがサスペンドされたことを示す。

```
| ?- cp2.
goals(breadth-first): test.
-->merge($([1,2],[a,b],_112),_186,_187)
-->out($($cha(_1311,_1312)),_1402,_1403)
-->merge1($([1,2],[a,b],_112),_1117,_1118,_1119,_1120,
_1121)
-->out1($($cha(_1311,_1312)),_1910,_1911,_1912,_1913,_1914)
-->out2($($cha(on_asking,_1312)),_1915,_1911,_1916,_1917,
_1918)
-->ulist_m_no_susp(_112,1,_2032)
-->merge($([a,b],[2],_2032),_2035,_2036)
Wt->out1($($cha(on_asking,_1312)),_2614,_1911,_1912,_2615,
_2616)
Wt->out2($($cha(on_asking,_1312)),_3197,_1911,_1916,_3198,
_3199)
-->merge1($([a,b],[2],_2032),_4402,_4403,_4404,_4405,_4406)
G->out1($($cha(on_asking,[1|$cha(_5837,_5838)])),_2614,
_1911,_1912,_2615,_2616)
G->out2($($cha(on_asking,[1|$cha(_5837,_5838)])),_3197,
_1911,_1916,_3198,_3199)
-->ulist_m_no_susp(_2032,a,_5415)
-->merge($([2],[b],_5415),_5418,_5419)
-->writeln(1)
An answer is 1
-->out($($cha(_5837,_5838)),_6724,_6725)
                              :
                              :
                              :
                              :
-->out1($([b|$cha(_16043,_16044)]),_18534,_18535,_18536,
_18537,_18538)
-->out2($([b|$cha(_16043,_16044)]),_18539,_18535,_18540,
_18541,_18542)
-->writeln(b)
An answer is b
-->out($($cha(_16043,_16044)),_19533,_19534)
-->out1($($cha(_16043,_16044)),_20956,_20957,_20958,_20959,
_20960)
-->out2($($cha(on_asking,_16044)),_20961,_20957,_20962,
_20963,_20964)
G->out1($($cha(on_asking,[])),_21626,_20957,_20958,_21627,
_21628)

All the goals are exhausted. simulator stopped!
```