論理型プログラムのＯＲ並列処理における冗長計算の除去方式

松原　健二　，　相田　仁　，　後藤　厚宏　，　田中　英彦　，　元岡　達

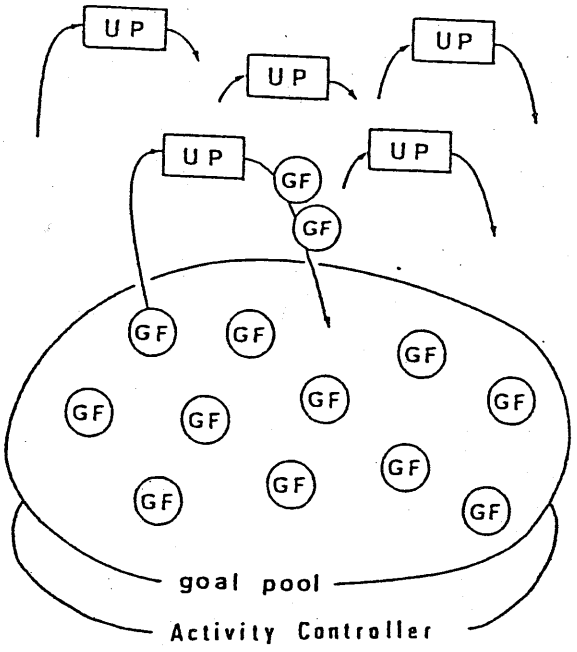
（　東京大学　工学部　）

１．　はじめに

論理型プログラムでは、問題解決のアルゴリズムが明確でないために、従来のプログラム言語では記述が困難であった問題に対しても、宣言的に解りやすい単純な形に記述することができる。この場合、論理型プログラムを実行するシステムはプログラムによって記述された事実関係（定義節）により推論を行ないながらそのプログラムの実行を進める。

この推論として現在のProlog のとっている方法は試行錯誤であり、この方法では冗長な計算が生じることがある。この冗長性は、本来問題解決のアルゴリズムが明確でないことにより生ずるものであるため、ユーザがこの冗長性を排除すべくプログラムを記述することは非常に困難である。そこでシステムがプログラムの実行において冗長性を自動的に除去することが望まれる［Per８２］。

本稿では論理型プログラムのＯＲ並列処理における冗長計算の除去方式を示し、シミュレーションによりその評価、検討を行なう。



[Fig. 2.1] Parallel Processing Feature of PIE

２．　高並列推論エンジンＰＩＥについて

論理型プログラムのＯＲ並列処理モデルとして、我々が開発を進めている高並列推論エンジンＰＩＥ（Parallel Inference Engine ）を用いる。

ＰＩＥでは論理式の導出に忠実なゴール書き変え操作に基づくＯＲ並列処理を採用している［ＰＩＥ１］。ＰＩＥの基本処理単位は、“導出過程の中間ゴール節とそれに付随した単一化の環境”であり、これを特にゴールフレーム（Goal Frame：以下ＧＦと略す）と呼ぶ。ＰＩＥにおける論理型プログラムの実行は、ＯＲ並列単一化プロセス間で独立性の高いＧＦを次々と受け渡しながら並列に進む。ＯＲ並列単一化プロセスではＧＦの導出において定義節のすべての選択肢を適用する。この結果プログラムに内在する並列性を活かした高並列処理が実現できる。

ＰＩＥの処理イメージを図２．１に示す。ＰＩＥの基本処理要素であるユニファイプロセッサ（ＵＰ：Unify Processor）はＧＦを入力するとその中のひとつのゴールリテラルについて対応するすべての定義節テンプレートとの単一化を行ない、新たなＧＦを（複数）導出する。ＧＦはゴールプールに蓄えられ、ＵＰに対する新たな入力となる。このようにして生成された多数のＧＦはゴールプールに付随したアクティビティコントローラ（ＡＣ：Activity Controller ）によって管理される。ＡＣは探索ストラテジに従ったＧＦの選択、資源管理および負荷分散を行なう。

３．　冗長計算除去による効率化

３．１　ゴール書き換えモデルに基づく論理型プログラムの実行における冗長性とその除去方法

ゴール書き換えモデルに基づく論理型プログラムの実行における冗長性とその除去方法は以下のように分けることができる。

Ⅰ．探索木上に同じＧＦが現われる場合

同じＧＦは、単一化されるゴールリテラルの順番が異なっていても最終的な結果は等しい。従っ

てそのうち一つのＧＦについて実行すればよく、残りのＧＦの実行は冗長である。

この冗長性を除去するために、既に実行されたＧＦを記憶しておき、新しく導出されたＧＦと比較を行なう。一致していれば、それは重複するＧＦであるから、その新しく導出されたＧＦを実行せずに削除する。

また実行待ちのＧＦと、新しく導出されたＧＦとの比較も行ない、一致していれば重複するＧＦを削除する。

Ⅱ．ひとつのＧＦ内に同じゴールリテラルが現われる場合

これは同じ単一化が繰り返されることになるから、そのうち一つのゴールリテラルに対して単一化を行なえばよく、残りのゴールリテラルの実行は冗長である。

この冗長性を除去するために、新しく導出されたＧＦの中に含まれるゴールリテラル同士を比較する。もし一致していれば、それは重複するゴールリテラルであるから、どちらか一方のゴールリテラルを除去する。

Ⅲ．異なるＧＦ内で同じゴールリテラルの単一化が繰り返される場合

この冗長性を除去するために、既に単一化を実行したゴールリテラルを記憶しておき、新しく導出されたＧＦ中のゴールリテラルと比較を行なう。この比較は記憶しておくゴールリテラルにより、次の二つに分けられる。

① 単一化に成功したゴールリテラルとの比較
一致すれば、記憶した結果を用いて単一化を行なわずに新ＧＦの導出ができる。

② 単一化に失敗したゴールリテラルとの比較
一致すれば、単一化を実行せずにＧＦの実行の失敗が解る。

３．２ ＰＩＥにおける冗長計算除去の方法

２で述べたようにＰＩＥでは単一化を行なうときに定義節のすべての選択肢を適用するＯＲ並列処理を採用している。ＯＲ並列処理では一般に一つのＧＦから複数のＧＦが導出され、導出されたＧＦはゴールプールに蓄えられる。従ってＰＩＥの処理モデルでは、冗長計算の除去はゴールプール中のＧＦの削除という形で実現される。ＧＦを削除することによって処理速度の向上を期待することができ、またゴールプールの記憶領域も節約することができる。

現在のところＰＩＥでは３．１で示した論理型プログラムの冗長計算の除去方式のうち、そのオーバーヘッドが小さいと考えられる、ＧＦの比較による冗長性の除去と単一化に失敗するゴールリテラルによる冗長性の除去を行なう。

４． 冗長計算除去のシミュレーション

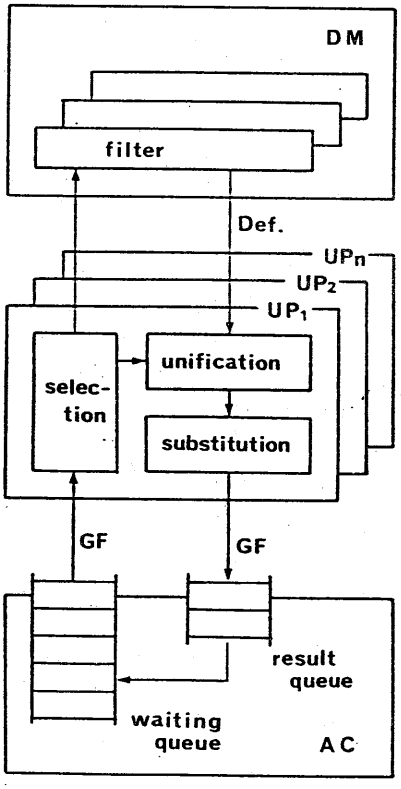
４．１ シミュレーションの目的

３．２で示した冗長性の除去方法のシミュレーションにより、論理型プログラムの冗長性の特性評価および冗長計算除去による効果を具体的に調べることを目的とする。

ＰＩＥには基本処理シミュレータ［ＰＩＥ２］および並列処理シミュレータ［ＰＩＥ３］の２種類のシミュレータがある。そこでまず基本処理シミュレータに冗長計算の除去方法を実装し、論理型プログラムの実行における冗長性を測定した。さらに並列処理シミュレータに冗長計算の除去方法を実装し、実際の並列処理の環境下でどれだけ冗長性を除去できるかを調べた。

４．２ 基本処理シミュレータ

基本処理シミュレータは、Unix 上のＣ言語で記述されており（約3600行）、図４．１で示すようにＵＰ部とＡＣ部から成る。



[Fig. 4.1] Preliminary Simulation Model

UP部は、PIEにおけるUPと定義節を蓄えているDM（Definition Memory ）に相当し、任意の台数のUPをシミュレートできる。UPはGF内の選択されたゴールリテラルについて定義節テンプレート間並列処理を行なう。

AC部はPIEにおけるACとMM（ゴールプール）に相当する。シミュレートするACは1台であり、ゴールプールの管理を行なう。ゴールプールはwaiting-queue とresult-queueからなる。waiting-queue は優先度に従ってソートされた実行待ちGFの queueである。ACはwaiting-queue の先頭からUP台数分のGFを取り出しUPへ送る。UPによって生成されたGFは result-queue につながれる。ACは result-queue からGFを取り出すとその優先度を評価して waiting-queueに挿入する。

基本処理シミュレータでは、ACおよびMMが一台であるのでGFはすべてそのMM内に記憶される。従って冗長性を最大限除去することができ、論理型プログラムの実行における冗長性の特性評価を行なうことができる。

GFはタグ付きの固定長セルが連続した構造になっており［PIE4］、次の3つの部分から成る（図4．2）。

① ヘッダ： ゴールフレームの属性や区分を示す。

② リテラル部： ベクタ型のリテラルの領域。

引数は各々1つのセルで表現され、1セルでは表現できないものは構造部へのポインタとなる。

③ 構造部： 1セルで表わせないデータ構造を収納する。

4．3 基本処理シミュレータへの冗長性除去方法の実装

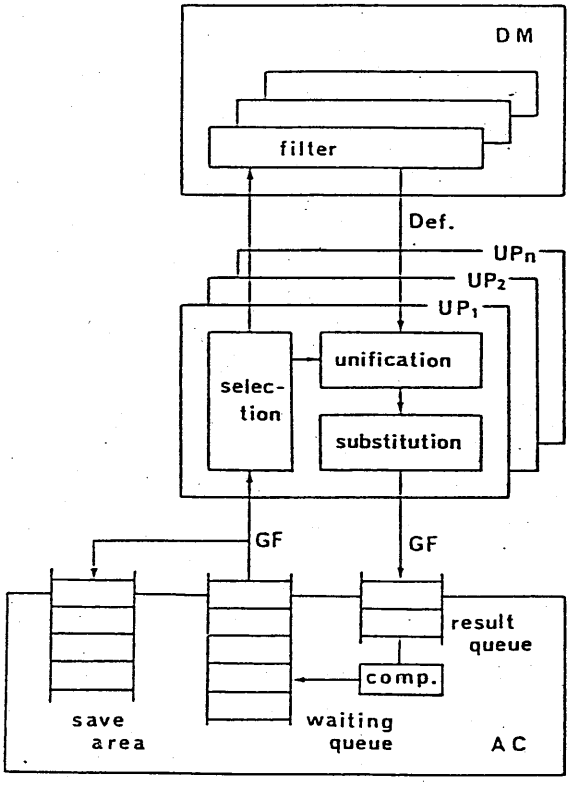
基本処理シミュレータにおける冗長性除去の様子を図4．3に示す。新しく導出されたGFはresult-queueに蓄えられ、冗長性除去のための比較を行なった後、waiting-queue で実行を待つ。すべてのUPの処理が終了すると実行待ちのGFは waiting-queue からUP台数分だけUPに送られ、このとき同時にsave-area に格納される。

冗長性の除去方法を実装するためにシミュレータを変更した部分および新たに書いたプログラムは約1000行である。

	type	data
HEADER AREA	GF ID.	
	GF length	
	number of literals	
	depth in Search Tree	
LITERAL AREA	INT	3 (argn)
	SYM	sym. id of 'P'
	SYM	sym. id of 'a'
	VAR	var. id of '*x'
	INT	3 (argn)
	SYM	sym. id of 'Q'
	VECT	0 (index)
	VAR	var. id of '*y'
STRUC-TURE AREA	INT	4 (argn)
	SYM	sym. id of 'f'
	VAR	var. id of '*x'
	INT	321
	VAR	var. id of '*y'

?- P(a,*x),Q(f(*x,321,*y),*y).

[Fig. 4.2] Internal Representation of Goal-Frame



[Fig. 4.3] Elimination of Redundant Calculation

4．3．1　ゴールフレームの冗長性の除去

重複するＧＦを除去する方法として次の二つの方法について調べた。

①　新たに導出されたＧＦと、そのときまでに実行されたＧＦのすべてとの比較（これをsave-cmpと呼ぶ）による重複ＧＦの除去

②　新たに導出されたＧＦと、実行待ちのＧＦとの比較（これをwait-cmpと呼ぶ）による重複ＧＦの除去

▶ ＧＦの比較と削除

ＧＦはMM中の配列ＧＦmemoryに記憶されている。ＧＦの比較をするとき、単に順探索するのでは、明らかに異なるＧＦとも比較してしまい無駄が大きい。そこでＧＦの最初のゴールリテラル名および第一引き数の属性と値により、ハッシュ関数を用いてＧＦを区別する。これにより、比較の無駄を省き、速度向上が図れる。第一引数の値と属性を用いた理由は、定義節の動的な絞り込みのシミュレーション結果等［ＰＩＥ2］［War７７］における有効性に基づいている。

ＰＩＥにおいてＧＦはＵＰにおける縮退操作により常に定まった形式に整形されているので、同じＧＦかどうかは１セルずつ順番に比較し、最後のセルまで一致しているかどうかにより判定できる。比較の途中で一つでもセルが一致していなければ違うＧＦであるので、比較を止めてかまわない。但しヘッダ部のＧＦ固有の情報については比較を行なわない。同じＧＦと解った場合にはそのＧＦを削除する。

4．3．2　ゴールリテラルの冗長性の除去

単一化に失敗するゴールリテラルの比較（これをfail-cmpと呼ぶ）による冗長性の除去を行なう。

▶ ゴールリテラルの記憶

ゴールリテラルの単一化が失敗したら、そのゴールリテラルはMM中の配列 literalmem にコピーされる。このとき、ＧＦの場合と同様に比較の速度向上を図るために、ゴールリテラル名および第一引き数の属性と値によりハッシュ関数を用いてゴールリテラルを区別し、同じハッシュ関数値を持つゴールリテラルをリストでつなぐ。

literalmem はハッシュ関数値をindex とするポインタの配列により管理される。ゴールリテラルをコピーするとき、ＧＦの構造部については再配置する。

▶ ゴールリテラルの比較およびＧＦの削除

比較もＧＦの場合と同様にして１セルずつ行なう。構造部へのポインタはゴールリテラルをコピーした際の再配置を考慮しなければならない。同じゴールリテラルであると解った場合、そのゴールリテラルを含むＧＦを削除する。

ゴールリテラルの引数が変数であるとき、変数は定数を包含すると考えるとより一般的な冗長性の検出ができる。しかし、図４．４に示すように、ゴールリテラル１と定義との単一化は失敗し、ゴールリテラル２と定義の単一化は成功する。このとき変数は定数を包含すると考えると、ゴールリテラル２は失敗するゴールリテラルであるとされてしまう。このような誤りを避けるためには、比較ではなく、ゴールリテラル２の変数を定数として単一化を行なわなくてはならないが、それではオーバーヘッドが大きくなる。そこでここではゴールリテラルが完全に一致する場合に限って冗長性の除去を行なった。

```
definition    example( a, b).
literal-1     example(*X,*X).
literal-2     example(*X, b).
```

[Fig. 4.4]

4．4　並列処理シミュレータ

並列処理シミュレータも基本処理シミュレータと同様に、ＵＮＩＸ上のＣ言語で記述されている（約10000行）。プロセスのコンカレントな実行がシミュレートでき、実際の並列処理環境下での冗長計算除去の評価が行なえる。

ＰＩＥの１ユニット分のシミュレーションモデルを図４．５に示す。

4．5　並列処理シミュレータへの冗長性除去方法の実装

並列処理シミュレータにおいても基本処理シミュレータと同じ冗長性の除去方法を実装した。

新しく導出されたＧＦはネットワークを通じて運ばれ、ＧＦ input queueに蓄えられる。ＧＦはＧＦ managing unitにおいて冗長性除去のための比較を行なったあと、ＧＦ memory で実行を待つ。ＧＦはＵＰ Controller に送られるとき同時にＧＦ memory 内のsave-area に蓄えられる。

5．シミュレーション結果および評価・検討

5．1　測定内容

評価用プログラム［equiv 2］（式の簡単化），［ ds ］（微分プログラム）を用いてシミュレーションを行なった。実行は全解を探索し、その過程における各種の特性値を測定した。

基本処理シミュレーションでは冗長性の特性評価を行なうために、それぞれの冗長性の除去方法による、

・単一化の回数の変化
・実行待ちのＧＦ数の変化
・探索木の深さと削除されるＧＦ数の関係

を調べた。また、冗長性の除去による記憶領域のオーバーヘッドを調べるために

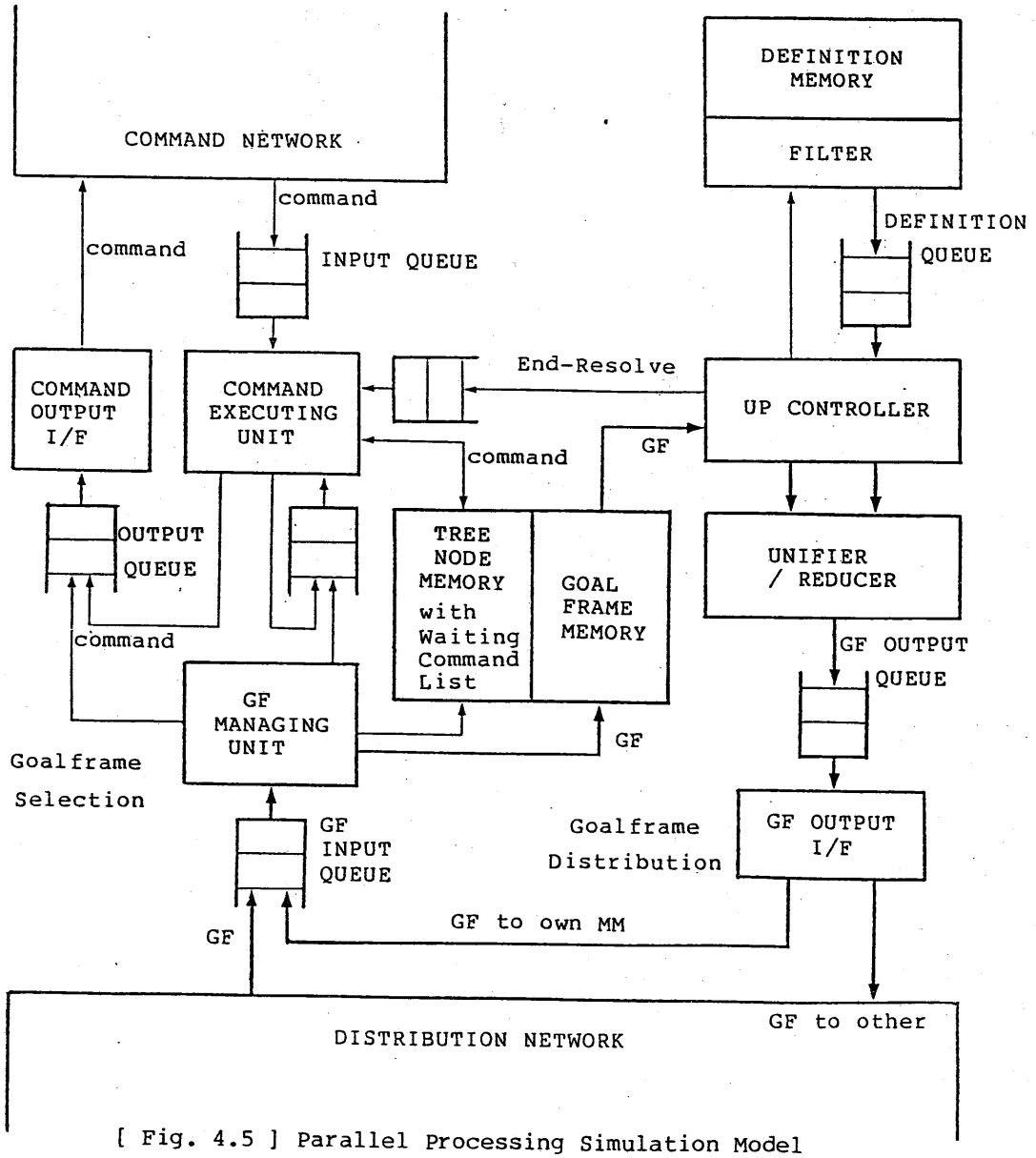
・save-area のＧＦ数と冗長性の除去を行なわない場合の総ＧＦ数
・fail-cmpに要するセル数と単一化されるゴールリテラルのパターン

を調べた。

並列処理シミュレーションでは、実際の並列処理の状況下でどれだけ冗長性が除去できるかを調べるため、ＵＰ台数を変化させてそれぞれの除去方法による、

・単一化の回数の変化
・シミュレーション・タイム

を調べた。



［ Fig. 4.5 ］ Parallel Processing Simulation Model

5．2　基本処理シミュレーションの結果

5．2．1　冗長性除去の効果

表5．1，表5．2は冗長計算の除去を行なったときの単一化の実行回数である。UP台数は無限大を仮定した。GFの比較、単一化に失敗するゴールリテラルの比較、またそれらの組み合わせにより、単一化の実行回数を大幅に減らすことができる。効果がsave-cmp, fail-cmp, wait-cmpの順になっているのは、比較の方法から見て妥当である。

図5．1，図5．2に示すように、実行待ちのGF数も相当減少させることが可能である。[ds]では比較を行なわない場合、最大１１０ものGFがwaiting-queue に蓄えられるが、save-cmpにより最大２１、wait-cmpにより最大５５、fail-cmpにより最大４９にまで減らすことができる。大規模なプログラムではGF数がUP台数をはるかに上回ることが十分予想され、そのような場合大きな効果を持つであろう。探索木上の深さ（depth ）と、削除されたGF数の関係を図5．3，図5．4に示す。fail-cmpにより削除されるGF数が、save-cmp、wait-cmpに比べてかなり多いが、単一化の実行回数はそれ程減少していない。それに対し save-cmp, wait-cmp は削除されるGFは少ないが、効果が大きい。これはfail-cmpでは単一化に失敗するGFしか削除されないからである。

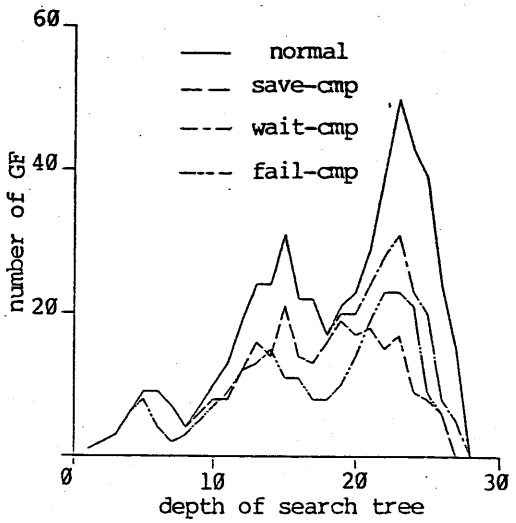
また図には表われていないが、探索木上において浅い所でのGFの削除ほど、削除によって後の単一化の回数の減少に与える効果が大きいことが観察された。

[Table 5.1]

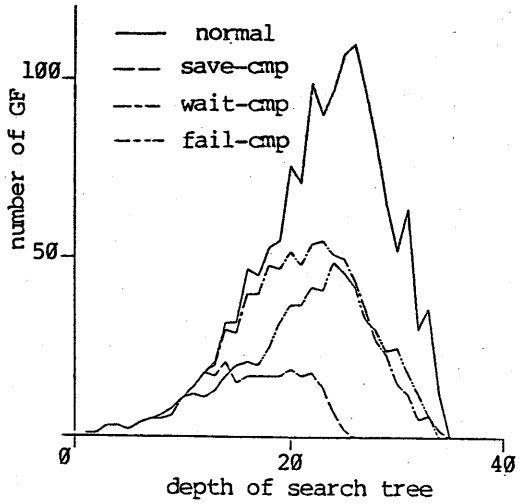
[equiv2]	Number of unifications		
comparison	success	failure	total (%)
normal	524	7431	7955 (100%)
save-cmp	317	3703	4020 (51%)
wait-cmp	419	5940	6359 (80%)
fail-cmp	524	3531	4055 (51%)
save wait-cmp	286	3458	3744 (47%)
save wait fail-cmp	286	1792	2078 (26%)

[Table 5.2]

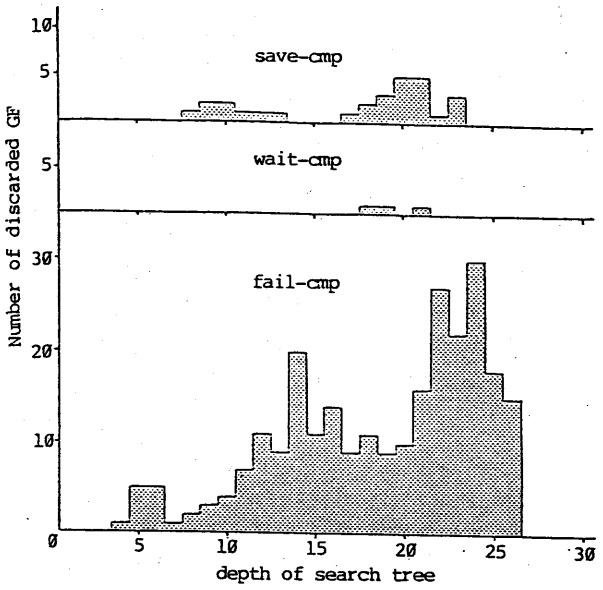
[ds]	Number of unifications		
comparison	success	failure	total (%)
normal	1514	18820	20334 (100%)
save-cmp	327	3041	3368 (16%)
wait-cmp	892	10146	11038 (53%)
fail-cmp	1514	7482	8996 (44%)
save wait-cmp	320	3000	3320 (16%)
save wait fail-cmp	320	1452	1772 (9%)



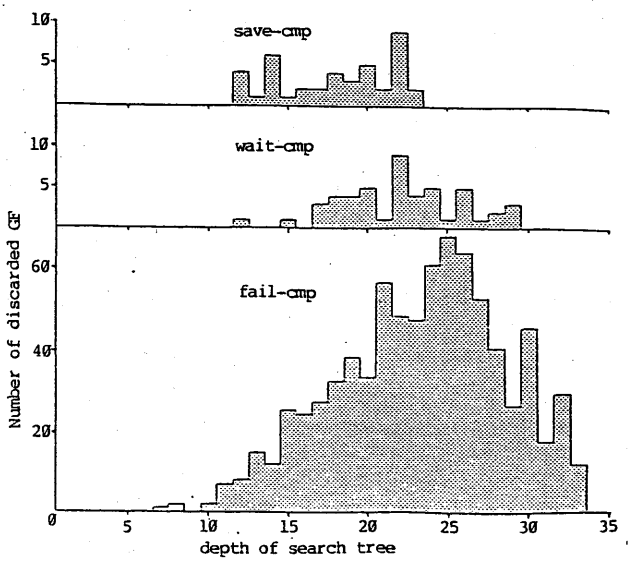
[Fig. 5.1] Num. of GF in waiting-queue (equiv2)



[Fig. 5.2] Num. of GF in waiting-queue (ds)



[Fig. 5.3] Num. of Discard (equiv2)



[Fig. 5.4] Num. of Discard (ds)

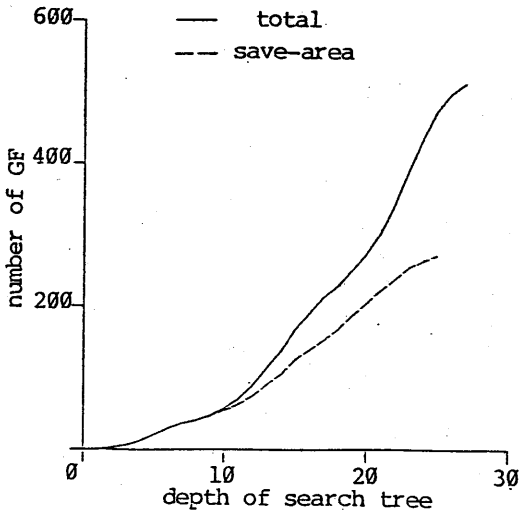
5．2．2 冗長性除去に要する記憶領域

・GFの比較の場合

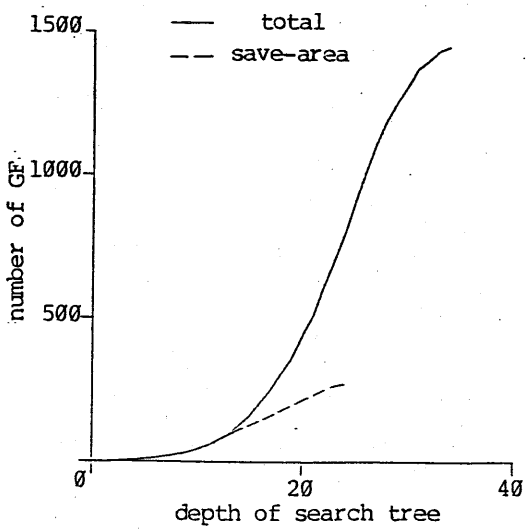
wait-cmpについては、実行待ちのGFはもともとwaiting-queue に蓄えられているわけであるから、余分な領域をほとんど必要としない。従って wait-cmp による効果が相当大きいことは非常に興味深い。

save-cmpについては、GFをUPに送り出した後もMM中に残すことにより、GFを記憶している。評価用プログラムは規模が小さいのですべてのGFを記憶することができた。しかし一般にすべてのGFを記憶するには大きな領域を必要とする。図5．5，図5．6に冗長性除去を行なわないときの総GF数と、save-cmpを行なうときにsave-area に蓄えられるGF数を示す。

save-cmpによりsave-area のGF数は大幅に減少しているが、それでも最終的には両評価用プログラムで２５０以上のGF数となる。実用的な大きなプログラムを実行する場合、すべてのGFを記憶しておくことは不可能であるから、領域に余裕がなくなればGFを捨てなくてはならない。このときどのGFを残しておくかが問題となる。save-cmpにより一致したGFの探索木上の深さの差を調べてみると、［equiv ２］では、深さの差はすべて２であった。またwait-cmpが予想以上に効果があるということより、重複するGFは探索木上において近い深さで現われることが多いと考えられる。そのような場合、save-area にはできるだけ新しいGFを残せばよいことになる。GFを残すストラテジとしては、他に次のような方法も考えられる。



[Fig. 5.5] Num. of GF in save-area (equiv2)



[Fig. 5.6] Num. of GF in save-area (ds)

・　その時点までに、比較の結果が一致して冗長性の除去を行なった回数の多いＧＦを残す。同じＧＦは幾つも現われる可能性があると考えられるからである。

・　特定のゴールリテラルを含むＧＦを残す。後に述べるように、重複するゴールリテラルはあるゴールリテラル名に偏る傾向が見られる。従って重複するＧＦも同じ傾向にあると考えられるからである。

これらの方法については今後、多くのシミュレーションを行なって検討すべき課題である。

・単一化に失敗するゴールリテラルの比較の場合

fail-cmpに要するセル数と、単一化が行なわれるゴールリテラルのパターン数を単一化の成功、失敗別に表５．３に示す。ゴールリテラルの記憶には、それほど大きな領域を必要としていない。これは記憶されるゴールリテラルのパターン数が［equiv ２］では２１通り、［ ds ］では３０通りと少ないからである。従ってゴールリテラルの冗長性の除去では、ＧＦの記憶に必要な記憶領域に比べ小さな領域により比較を行なうことができる。

［ Table 5.3 ］

	Num. of Cell for fail-cmp	Literal Patterns	
		success	failure
[equiv2]	542	43	21
[ds]	447	50	30

［ Table 5.4 ］

	execution time(cmp-time)	
comparison	[equiv2]	[ds]
normal	38 sec.	72 sec.
save-cmp	28(1.0)	24(1.9)
wait-cmp	35(2.0)	49(2.6)
fail-cmp	35(2.5)	64(5.2)
save wait-cmp	28(2.5)	25(2.5)
save wait fail-cmp	26(2.4)	24(2.7)

重複するゴールリテラルはあるゴールリテラル名に偏る傾向が見られる。この傾向が大きいとすると、記憶するゴールリテラル名をプログラマがメタ知識として記述したり、システムが自動的にそれを行なえば、より効果的な冗長性の除去が可能である。しかし単一化に失敗するゴールリテラルによる冗長性の除去では、定義節の絞り込みを行なった場合、大きな効果は望めない。

５．２．３　冗長性除去に要する時間

基本処理シミュレータの実行時間を表５．４に示す。これは、ＶＡＸ11／730 上でのシミュレータの動作時間であり、括弧内は比較に要した時間である。測定は精度の高いものではないが、冗長性の除去により実行時間は短縮されており、効果が示されている。［equiv ２］では約２０％程度の短縮で一様であるが、［ ds ］では比較の方法によってかなりばらつきが見られる。save-cmpでは実行時間が３分の１にまで短縮されている。冗長性除去のために要する時間の割合は全実行時間のほぼ一割以内であり、それほど大きなオーバーヘッドではない。

５．３　並列処理シミュレーションにおける冗長性除去効果

ＵＰ台数を１～１６まで変え、定義節の絞り込みを行なう場合と行なわない場合について、単一化の実行回数およびシミュレーション・タイムを表５．５～表５．１６に示す。シミュレーション・タイムは現在試作を進めている単一化プロセッサ［ＰＩＥ４］のマイクロプログラムのクロックを想定したシミュレーション・クロック数である。ただし、冗長性除去に要する時間は考慮していない。

[Table 5.5]
UP = 1 ; Filter

[equiv 2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	737	1261 (100%)	260957
save-cmp	317	383	700 (56%)	152285
wait-cmp	419	580	999 (79%)	212365
fail-cmp	524	737	1261 (100%)	247970
save wait-cmp	286	352	638 (51%)	141461
save wait fail-cmp	286	352	638 (51%)	136343

[Table 5.6]
UP = 1 ; No Filter

[equiv2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	7431	7955 (100%)	357648
save-cmp	317	3703	4020 (51%)	201555
wait-cmp	419	5940	6359 (80%)	290398
fail-cmp	524	3531	4055 (51%)	314462
save wait-cmp	286	3458	3744 (47%)	187807
save wait fail-cmp	286	1792	2078 (26%)	169581

[Table 5.7]
UP = 4 ; Filter

[equiv 2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	737	1261 (100%)	74599
save-cmp	411	556	967 (76%)	62094
wait-cmp	435	596	1031 (82%)	66377
fail-cmp	524	737	1261 (100%)	77490
save wait-cmp	378	507	885 (70%)	57681
save wait fail-cmp	408	540	948 (75%)	63284

[Table 5.8]
UP = 4 ; No Filter

[equiv2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	7431	7955 (100%)	106844
save-cmp	397	4775	5172 (65%)	80044
wait-cmp	467	6237	6704 (84%)	88432
fail-cmp	524	3609	4133 (52%)	92748
save wait-cmp	352	4378	4730 (60%)	69045
save wait fail-cmp	419	2743	3162 (40%)	83750

[Table 5.9]
UP = 16 ; Filter

[equiv 2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	737	1261 (100%)	42231
save-cmp	488	677	1165 (92%)	41251
wait-cmp	519	730	1249 (99%)	41944
fail-cmp	524	737	1261 (100%)	40553
save wait-cmp	488	677	1165 (92%)	40723
save wait fail-cmp	465	644	1109 (88%)	39435

[Table 5.10]
UP = 16 ; No Filter

[equiv2]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	524	7431	7955 (100%)	50411
save-cmp	485	6550	7035 (88%)	50871
wait-cmp	516	7227	7743 (97%)	50810
fail-cmp	524	3609	4133 (59%)	46598
save wait-cmp	429	5798	6227 (78%)	46381
save wait fail-cmp	474	3173	3647 (46%)	44931

[Table 5.11]
UP = 1 ; Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	1945	3459 (100%)	4985999
save-cmp	327	338	665 (19%)	105090
wait-cmp	892	946	1754 (51%)	267700
fail-cmp	1514	1945	3459 (100%)	454291
save wait-cmp	320	332	652 (19%)	103559
save wait fail-cmp	320	332	652 (19%)	99757

[Table 5.12]
UP = 1 ; No Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	18820	20334 (100%)	717224
save-cmp	327	3041	3368 (17%)	144391
wait-cmp	808	9039	9847 (48%)	377788
fail-cmp	1514	7751	9265 (46%)	590651
save wait-cmp	320	3000	3320 (16%)	142334
save wait fail-cmp	320	1486	1806 (9%)	129461

[Table 5.13]
UP = 4 ; Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	1945	3459 (100%)	131177
save-cmp	487	546	1033 (30%)	45401
wait-cmp	1057	1267	2324 (67%)	91479
fail-cmp	1514	1945	3459 (100%)	127573
save wait-cmp	505	557	1062 (31%)	48161
save wait fail-cmp	521	587	1108 (32%)	48102

[Table 5.14]
UP = 4 ; No Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	18820	20334 (100%)	188114
save-cmp	487	4726	5213 (26%)	61262
wait-cmp	1085	12149	13234 (65%)	128795
fail-cmp	1514	7856	9370 (46%)	168975
save wait-cmp	457	4543	5000 (25%)	56772
save wait fail-cmp	490	2396	2886 (14%)	54954

[Table 5.15]
UP = 16 ; Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	1945	3459 (100%)	188114
save-cmp	810	945	1755 (51%)	61262
wait-cmp	1271	1584	2855 (83%)	128795
fail-cmp	1514	1945	3459 (100%)	168953
save wait-cmp	777	903	1680 (49%)	56772
save wait fail-cmp	699	822	1521 (44%)	45954

[Table 5.16]
UP = 16 ; No Filter

[ds]	Number of unifications			simulation
comparison	success	failure	total (%)	time
normal	1514	18820	20334 (100%)	44986
save-cmp	813	8318	9131 (45%)	30738
wait-cmp	1350	15624	16974 (83%)	40152
fail-cmp	1514	7730	9244 (45%)	44575
save wait-cmp	852	8748	9600 (47%)	29409
save wait fail-cmp	805	3819	4624 (23%)	26960

冗長性の除去により、複数台のＵＰで並列処理を行なう場合についても単一化の回数を減らすことが可能である。ＵＰ台数が１台で定義節の絞り込みを行なわない場合が基本処理シミュレーションの結果に相当する。

ＵＰ台数が増すと、単一化の回数も増える。これは冗長計算を行なうＧＦがそれぞれのＵＰと対になったMM内に記憶されるために、ＵＰ台数が増加すると冗長なＧＦの削除が少なくなるからである。しかし、ＵＰ台数が１６台で定義節を絞り込んだ場合においても［equiv 2］では最高１２％、［ ds ］では最高５６％、単一化の回数は減少しており冗長性の除去による効果が十分現われている。

またＵＰ台数が多ければ当然、実行速度は上がり、シミュレーション・タイムは短縮されている。

Ｆilter による定義節の絞り込みをおこなった場合にはfail-cmpを行なっても単一化の回数は減少しない。これはfail-cmpにより削除されるＧＦに対しては、定義節を絞り込むと、対応する定義節がなくなるということであり、fail-cmpはwaiting-queueのＧＦ数を減らす効果しか持たない。定義節を絞り込んだ場合、常にfail-cmpでは単一化の回数が減少しないかどうかは、この結果だけからでは解らないが、実行に失敗するＧＦを削除するだけでは大きな効果は望めないと思われる。

ＧＦをMMからＵＰに転送するときに乱数を用いて送り先のＵＰを決めているために、測定は同じ実行を３回繰り返して行ない、その平均を採用した。それでも、表５．６などではsave-cmp, wait-cmpの場合の単一化の回数が save-cmp, wait-cmp, fail-cmp すべて行なった場合に比べて少なくなっている。これは乱数の影響がかなり大きいことを表わしている。

６．　おわりに

論理型プログラムは、問題解決のアルゴリズムが明確でないために、他の言語ではプログラムを記述することが困難であるような問題に対しても、宣言的に、簡潔明瞭な形でプログラムが記述できる。そのような場合、アルゴリズムが明確でないために、プログラムの実行において冗長性が生じる可能性がある。論理型プログラムを実行するシステムがその冗長計算を削除することによって、プログラマに負担をかけることなく大幅に処理速度を向上させることが期待できる。

本稿では高並列推論エンジンＰＩＥの実行における冗長計算の除去方式を示し、それらをＰＩＥの基本処理および並列処理の二つのシミュレータに実装し、シミュレーションにより冗長計算除去の具体的な効果を測定した。

基本処理シミュレータによる特性評価においては、冗長性の除去により単一化の実行回数を大幅に減少させ、記憶領域も大きく節約できることが解った。冗長性の除去に要する記憶領域については、記憶するＧＦ（ゴールフレーム）の選択方法は今後の課題ではあるが、単一化に失敗するゴールリテラルによる冗長性の除去に要する記憶領域は小さいことが確認された。実行時間については冗長性の除去に要する時間はシミュレータの全動作時間に比べ十分小さいことも解った。

並列処理シミュレータにおける並列処理の環境下においても、定義節の絞り込みを行なうと単一化に失敗するゴールリテラルによる冗長性の除去による効果はあまり見られないが、ＧＦによる冗長性の除去によって単一化の実行回数を減少させることができ、冗長性除去による効果が大きいことを確認した。

上記のシミュレーション結果より、本研究で示した論理型プログラムのＯＲ並列処理における冗長計算の除去方式は十分に有効であるということができる。

今後さらに、

- 単一化に成功するゴールリテラルによる冗長性の除去方法の検討
- 単一化に失敗するゴールリテラルによる冗長性の除去方法の改良
- save-area の大きさと記憶領域に余裕がなくなった場合に残すＧＦの選択方法の検討
- 並列処理シミュレータによる冗長性除去に要する時間の検討

などを行なっていく予定である。

謝辞

本研究を行なうにあたり、シミュレータを提供して頂いた丸山氏に深謝する。平田、湯原両氏には本研究に対して貴重な助言を頂いた。濱中氏にはグラフ作成の上で協力して頂いた。

参考文献

［ＰＩＥ１］後藤，相田，田中，元岡
“高並列推論エンジンＰＩＥ～ゴール書き換えモデルとアーキテクチャ～”
情報処理学会第２７回全国大会４Ｐ－９，１９８３．

［ＰＩＥ２］後藤，相田，山崎，丸山，湯原，田中，元岡，
“高並列推論エンジンＰＩＥにおける並列処理の効率化手法について”，
電子通信学会技術研究報告　ＥＣ 83- 9，１９８３．

［ＰＩＥ３］丸山，湯原，相田，後藤，田中，元岡，
“高並列推論エンジンＰＩＥ～並列度のシミュレーションとその評価～”，
電子通信学会技術研究報告　ＥＣ 83-39，１９８３．

［ＰＩＥ４］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”，
電子通信学会技術研究報告　ＥＣ 83-30，１９８３．

［Per８２］．Perira，L．M．，
“Logic Control with Logic，” Proc. of First International Logic Programming Conference，pp. 9-18（Sept．1982）．

［War７７］．Warren，D．H．D．，
“Implementing Prolog － compiling predicate logic programs，” D．A．I．Research Report 39-40，University of Edinburgh（1977）