論理型プログラムのOR並列処理における冗長計算除去の効果

4H-6

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

論理型プログラムの実行では、一般に探索木上には同一計算を行なうノードが多数存在する。そこで、一度実行された計算を記憶し、冗長な計算を取り除くことにより、実行の効率化が図れると考えられる。

本報告では、pure-Prolog をOR並列に処理するモデルのシミュレータを用いて、その具体的な効果を測定した結果を報告する。

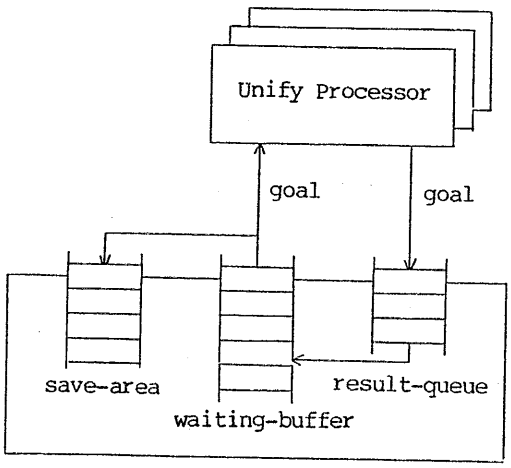
2. 冗長計算の検出方法

冗長な計算を検出する方法としては次の二つの方式が考えられる。

I. ゴール節(以下ゴールと略)全体の比較

II. ゴールリテラル(以下リテラルと略)ごとの比較

シミュレーションモデルを図1に示す。探索木上で同じ深さにあるゴールは、十分多数用意されたUnify Processorによって並列に処理される。一方導出されたゴールは result-queue に入り、冗長性除去のための比較を行った後、waiting-buffer で実行を待つ。1サイクルの処理が終ると、実行待ちのゴールは waiting-buffer からUnify Processorに送られ、同時に save-area に格納される。



[Fig.1]

Simulation model

2.1. ゴールの冗長性の除去

探索木上で全く同じゴールが複数出現する場合、そのうち一つのゴールだけを実行すればよい。重複するゴールを削除する方法として、次の二つの方法の効果を調べた。

① 新たに導出されたゴール(result-queueのゴール)と1サイクル前までに導出された全ゴール(save-area のゴール)との比較(save-cmp)による重複ゴールの除去

② 新たに導出されたゴールと実行待ちのゴールとの比較(wait-cmp)による探索木の同一レベル上での重複の除去

2.2. リテラルの冗長性の除去

既に単一化を実行したリテラルを記憶しておき、新しく導出されたゴール中のリテラルと比較を行う。この比較は記憶しておくリテラルにより、次の二つに分けられる。

① 単一化に成功したリテラルとの比較
(success-cmp)

一致すれば、記憶した結果を用いて新ゴールの導出ができる。

② 単一化に失敗したリテラルとの比較
(fail-cmp)

一致すれば、単一化を実行せずにゴールの失敗が解る。

今回は②のみについて測定した。

失敗リテラルとの比較において、リテラルの引数が変数である場合、変数は定数を包含するとみなすとより一般的な冗長性の検出ができると考えられる。しかし、図2に示すようにリテラル1と定義の単一化は失敗し、リテラル2と定義の単一化は成功する。このときリテラル1を失敗リテラルとして記憶し、変数は定数を包含すると考えると、リテラル2は失敗リテラルとみなされてしまう。

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

[Fig. 2]

このような誤りを避けるためには、比較ではなくリテラル2の変数を定数として単一化を行わなくてはならないが、それではオーバーヘッドが大きすぎる。そこで今回は完全に一致する場合についてのみ取り扱った。

3. シミュレーション結果

シミュレータはUNIX上のC言語で記述されている（約4000行）。評価プログラム［equiv 2］（式の簡単化）、［ds］（微分プログラム）の実行結果を表1，2に示す。

実行時間はVAX11／730 上でのシミュレータの動作時間であり、括弧内はそのうち比較に要した時間である。今回の方式ではゴール又はリテラルの比較は、リテラルごとにリストとして蓄えた。セル数とは、リテラルを記憶している領域の大きさである。各比較方式のもとで冗長性の除去によるメモリ及び実行時間の節約量と、比較に要するコストの比が評価の基準になる。

4. 検討

① プログラムにより効果の差はあるが、ゴールの比較、失敗リテラルの比較、またそれらの組合せにより、単一化の実行回数を大幅に減らすことが可能である。

② ゴールの比較の場合、save-area はメモリの空き領域を用いている。今回のプログラムは小さいので、save-area に全てのゴールを記憶している。しかし、一般に全てのゴールを記憶するには大きなメモリが必要であり、メモリの余裕が無くなればゴールを捨てなくてはならない。この場合どのゴールを残すかは今後の課題である。但しwait-cmpを用いた探索木上の同一レベルの冗長性除去の効果が予想以上であったことは興味深く、save-area に残すゴールの選択手法の参考になろう。

③ リテラルの比較は、それ程大きな記憶領域は必要とせず、小さい領域での効果が期待できる。これはリテラルの失敗パターン数が少ないためと考えられる。

④ シミュレータの実行時間は冗長性の除去により短縮されている。今後はより具体的な機構を検討し、時間の評価も含め、シミュレーションを行う予定である。

《 参考文献 》

［1］ 後藤 他 ：“高並列推論エンジンPIEについて”，Logic Prog. Conf. '83，ICOT.

［2］ 後藤 他 ：“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”，信学技報 EC83-9，1983.

［ Table 1 ］

[equiv2]	Num. of unifications			execution time (cmp-time)	Num. of Literal Patterns
comparison	success	failure	total (%)		
normal	524	7431	7955 (100%)	43s	success 43 failure 21
save-cmp	317	3703	4020 (51%)	29s(3.4s)	
wait-cmp	419	5940	6359 (80%)	33s(1.0s)	Num. of Cell for fail-cmp 542
fail-cmp	524	3531	4055 (51%)	31s(1.2s)	

［ Table 2 ］

[ds]	Num. of unifications			execution time (cmp-time)	Num. of Literal Patterns
comparison	success	failure	total (%)		
normal	1514	18820	20334 (100%)	84s	success 50 failure 30
save-cmp	327	3041	3368 (16%)	28s(3.2s)	
wait-cmp	892	10146	11038 (54%)	53s(2.0s)	Num. of Cell for fail-cmp 447
fail-cmp	1514	7482	8996 (44%)	68s(4.4s)	