3Y-2

モード情報を用いたシャローバックトラックの高速化方式

阿部重夫　桐山薫　黒沢憲一
(日立製作所　日立研究所)

1. はじめに

Prologの高速化のためには、クローズの選択、実行及びバックトラックの最適化を行なうことが必要である。Warrenの命令セット[1]は、これらの最適化に適しており、文献2)においてWarrenの命令セットをベースにしたクローズの選択、実行の最適化方式について述べた。

バックトラックには、シャローバックトラックとデープバックトラックの2つがあるが、Warrenの命令セットは、デープバックトラックに基づいた命令セットであり、シャローバックトラックのときの効率が悪い。本論文では、シャローバックトラックを高速化する方式について述べる。

2. 高速化の考え方

プロセジャをバックトラックの形態により分類すると次の3つに分かれる。

(1) シャローバックトラックしか生じないプロセジャ
(2) デープバックトラックしか生じないプロセジャ
(3) シャロー、デープバックトラックとも生じるプロセジャ

ここでプロセジャの一般形を次のように表わす。

```
h(H11,··, H1n):-B_IN1, !, b1(B11,··, B1m1),····
h(H21,··, H2n):-B_IN2, !, b2(B21,··, B2m2),····
                    :
h(Hk-1,1,··, Hk-1,n):-B_INk-1, !,
  bk-1(Bk-1,1,··, Bk-1,m(k-1)),····
h(Hk1, ··, Hkn):-B_INk, bk(Bk1, ··, Bk,mk), ····
```

ここでB_INi(i=1, ··, k)は、決定的な組込述語命令である。B_INi(i=1, ··, k-1)の直後にカットがあれば、シャローバックトラックしか生じないプロセジャとなる。これをシャローバックトラックプロセジャと呼ぶことにする。get命令によりユニファイされるアーギュメントレジスタの内容は、undo処理により回復できるから、シャローバックトラックプロセジャでは、put及びunify命令で破壊されるアーギュメントレジスタのみ退避すれば良く、基本的にはチョイスポイントの生成は不要である。

3. スタックの管理

シャローバックトラック中は、ローカル、ヒープ、トレールの各スタックの管理はできるだけなくするようにすることが必要である。

ローカルスタック

シャローバックトラックプロセジャに対しては、チョイスポイントの生成は不要である。環境の生成の要否は、クローズに依存する。また、それらを生成すれば、ユニフィケーションに失敗したときにオーバヘッドとなる。このため、環境の生成はユニフィケーションの失敗の可能性のあるコードを全て終了してから行なうようにする。

ヒープ

モード情報を用いれば、ヒープへの書き込みがあるか否かを判定できる。シャローバックトラック中にもしヒープへの書き込みがないときは、top-of-heapレジスタのリセットは不要となる。

トレイルスタック

モード情報を用いれば、変数の拘束があったか否かの判定は可能である。もし拘束がないときは、シャローバックトラック時のundo処理は不要となる。

4. コードの最適化

シャローバックトラックを高速化するために、失敗する可能性のあるコード(これをチェックコードと呼ぶ)を、代入となるユニフィケーション及びゴール生成のコード(ロードコードと呼ぶ)の前に生成するようにする。チェックコード及びロードコード内でのレジスタ競合は実行する順序を変更することにより解消する。[2] 但し両者にまたがるレジスタ競合は、ワークレジスタを割りあてることにより解消する。これはシャローバックトラックの前に不要な代入をさけるためである。

もしプロセジャの中で同一のユニフィケーションが存在すれば、[3] チェックコードのリスト、構造体の要素のユニフィケーションでアーギメントレジスタの内容が破壊されても良い。そのときは、シャローバックトラックでは、最初にそのユニフィケーションを実行しただけで、後のクローズでは省略できる。そうでないときは、それらの内容をレジスタに退避する必要がある。

有向グラフ化

クローズiに対して、HUiをユニフィケーションに失敗する可能性のある引数番号の集合とする。従ってHUi

Shallow Backtraking Optimization Using Mode Information
Shigeo ABE, Kaoru KIRIYAMA, Ken-ichi KUROSAWA
HITACHI, Ltd.

はチェックコードに対応する。

ここで、HUiに現われる変数を次のように分類する。

（1）変数X及びYが同一のリスト、構造体中に現われるときは、X，Yは同一の集合Gijに含まれる。

（2）変数Xが、X以外の変数が含まれないリスト、構造体中にあるときは、あるいはXがリスト、構造体に含まれないときは、XはXのみを要素とする集合Gijに含まれる。

集合Gij(j=1，‥，p(i))を用いて連結部分グラフDSGij=IUij→OUijを次のように定義する。

（1）変数X∈GijがHip，p∈HUiのとき、p∈IUijとする。

（2）変数Y∈GijがY=Biq，q∈Biのとき、q∈OUijとする。但し、Biはボディの引数の集合でBi={1，‥，mi}である。

ここで、$BUi=OUi_1 \cup \cdots \cup OUip(i)$とする。

レジスタ競合解消

もしHUi⊃BUiが成立するならば、チェックコードとロードコード間でレジスタ競合はない。BUi-HUiに含まれる要素jがH-HUiに含まれるときはレジスタ競合が生じる。ここでHは、ヘッドの引数の集合でH={1，‥，n}である。このレジスタ競合は、ワークレジスタを割りあてて解消するため、jはBUiから削除する。

次にk個のクローズ間でのレジスタ競合を考える。これは、i番目の節のユニファイ命令で引数レジスタが破壊されるときに生じうる。i番目の節のユニフィケーションの失敗によるi+1番目の節の実行は、i+1番目の節の最初から行なうことが必要であるが、2つのクローズでユニフィケーションが同じであれば、シャローバックトラック時は、そのユニフィケーションの省略が可能である。従ってそれ以外でワークレジスタを割りあててレジスタ競合の解消を行なえばよい。

有向グラフの順序づけ

DSGUip，DSGUiqにおいて、

$OUiq \cap IUip \neq \phi$，$OUip \cap IUiq = \phi$

であるならば、DSGUipをDSGUiqより先に実行すれば、レジスタ競合は生じない。

ロードコードについてもチェックコードと同様にしてDSGijを定義してレジスタ競合の解消を行なうことができる。

コード生成

ユニフィケーションの失敗をできるだけ早く見つけるコード生成は次の手順で行なえばよい。

（1）Hij(j∈HUi)中の定数データの生成

（2）DSGUijを決められた順序で展開

（3）命令化された組込述語B_INiの展開

（4）Hij(j∈H-HUi)中の定数データのユニフィケーション

（5）DSGijを決められた順序で展開

（6）Biにおける定数データのコード

q-sortのsplitのコード例

```
:-mode split(+, +, -, -).
split(Y, [X|L], [X|L1], L2):-X=<Y, !,
      split(Y, L, L1, L2).
split(Y, [X|L], L1, [X|L2]):-split(Y, L, L1, L2).
split(_, [ ], [ ], [ ]).
```

に対するコードは次のようになる。

```
s1: get_list A2
    unify_variable A5
    unify_variable A2
    move ACP, A6
    movea s2, ACP
    leseq  A5, A1
    get_list  A3
    unify_value  A5
    unify_variable  A3
    move A6, ACP
    execute A2, (fail, s3, s1, fail)

s2: move A6, ACP
    get_list A4
    unify_value  A5
    unify_variable  A4
    execute A2, (fail, s3, s1,fail)

s3: get_nil A2
    get_nil A3
    get_nil A4
    proceed
```

ここでmovea s2，ACPは、s2のアドレスを別解アドレスポインタにロードすることを示す。IPPでのq-sortの性能は921kLIPSでシャローバックトラックの最適化をしない場合の2倍の性能向上が得られることが確かめられた。

5．　おわりに

Warrenの命令セットをベースにしたシャローバックトラックの最適化方式について述べた。本方式により実行速度の大幅な向上とともに、効率のよいオブジェクトコードの生成が可能である。

参考文献

1）D. H. Warren, "An Abstract Prolog Instruction Set," Technical Note 309, SRI International, October, 1983.

2）S. Abe et al., "High Performance Integrated PROLOG Processor IPP," Proc of the 14th International Symposium on Computer Architecture, June 1987.

3）松本他、"PROLOGの非決定性処理を最適化するコンパイル方式の評価、" 第36回情報処理全国大会、6H-7，昭和63年3月