決定的なPrologプログラムのCへの変換および最適化*

5E−2

片峯 恵一　周 能法　長澤 勲†
(九州工業大学)‡

1 はじめに

現在のPrologコンパイラの実現には2つの方法がある．1つ目の方法はプログラムを抽象マシンの命令にコンパイルするもの(方法1)である．この方法は移植性や実現性は高いもののバイトコードを解釈し実行するためオーバーヘッドを生じる．このためあまり高速な処理は望めない．2つ目の方法はプログラムを実際のマシン命令(ネイティブコード)にコンパイルするもの(方法2)である．この方法は，方法1に比べて高速であるが移植性や実現性は低い．本論文では，方法1と方法2とを融合する新しい方法について述べる．この方法では，大域決定的なプログラムは，まずCに変換され，Cコンパイラによってネイティブコードにコンパイルされる．非決定的なプログラムは，バイトコードに変換される．この方法は，方法1の利点を持っているだけではなく，方法2なみの性能を得ることができる．

2 準備

一般のProlog節は次のいずれかの形式に変換することができる．

H :- C : B　　(1)
H :- C ? B　　(2)

Hは頭部，Cは条件部，Bは本体，':' は決定的な選択子，'?' は非決定的な選択子を表す．あるゴールGに対して，もしGがHにマッチすることができ(G=Hθ)，しかも条件部Cが成功するならば，(1)は決定的にBを実行する．言い換えると，Bが失敗した場合に，次の節は実行されない．(2)は非決定的にBを実行する．言い換えるとBが失敗した場合に，次の節が実行される．ある述語に対して，もしすべての節の選択子が ':' であれば，この述語を決定的という．ある述語に対して，もしこの述語が決定的で，しかもすべての節の本体に現れる述語が大域決定的であれば，この述語を大域決定的という．節の条件部には，大域決定的な述語しか使えない．

TOAM(Tree Oriented Abstruct Machine)[4]は，照合木指向の抽象マシンでありWAM (Warren Abstruct Machine)[1]を拡張したものである．その特徴としては，プログラムを照合木に変換することにより，すべての入力変数に対してインデックスすることや，節間の共有命令を1回しか実行しないことなどがある．

TOAMには，以下の複数命令がある．

- conditional jump
  条件付きジャンプ命令
- fetch
  複合体，あるいはリストの引数を引き出す命令
- assign
  代入命令
- move
  移動命令
- unify
  単一化命令
- build
  複合体あるいはリストの引数を埋める命令
- hash
  ハッシング命令
- procedural
  述語の起動および終了に伴う制御を行なう命令

例えば，次のプログラムについて考える．

```
membchk(X,[X|_]):- true : true.
membchk(X,[_|Y]):- true : membchk(X,Y).
```

このプログラムは，1番目の引数が2番目の引数の要素であるかどうかをチェックするものである．このプログラムを図1のコードにコンパイルする．

```
membchk/2:
    jump_on_not_list A2, F          (1)
    fetch_var x(1)                  (2)
    fetch_var A2
    jump_on_not_id A1, x(1), L      (3)
    proceed.
L:
    execute membchk/2
```

図1: membchkのTOAMコード

ここで，(1)は，2番目の引数A2がリストであるかどうかをチェックし，リストでないならばF(fail)へジャンプする命令である．(2)はリストのcarをレジスタx(1)に入れ，リストのcdrをA2に入れる．(3)は，A1とx(1)とを比較し，等しくなければLへジャンプする．

3 変換

この節では，TOAMの命令からCプログラムの生成について述べる．その前に，システム構造について説明する．

コンパイラが，与えられたPrologプログラムを決定的な部分と非決定的な部分に分け，決定的なプログラムはC関数に変換し，非決定的なプログラムはTOAMのバイトコードに変換する．プログラムを実行する前に，変換されたC関数のオブジェクトとTOAMのインタプリタを結合し，新しいインタプリタを生成する．実行時に，この新しいインタプリタがバイトコードを解釈実行する．

具体的な変換方法は以下の例で示す．　図2は，図1に示されるTOAM命令から変換されたCプログラムを示す．述語membchkに対して，2つの関数が生成される．TOAMコードから述語membchkを呼ぶ時に，membchk()が使われ，Cプログラムから述語membchkを呼ぶ時に，membchk2(op1,op2)が使われる．　関数が返す値は，TRUE(述語が成功)か，あるいはFALSE(述語が失敗)である．各マクロは以下の意味を持つ．

- ARG(N)
  n番目の引数を受けとる．

---

*Translation of Deterministic Prolog Programs into C and the Optimization
†Keiichi KATAMINE, Neng-Fa ZHOU and Isao NAGASAWA
‡Kyushu Institute of Technology

- DEREF(op)
  dereference チェーンを遡り、op の束縛値を求める.
- ISLIST(op)
  op がリストであれば TRUE、そうでなければ FALSE を返す.
- UNTAGED(op)
  TAG を外す.
- FETCH_VAR(op)
  命令 fetch_var X を実行する.

図 2 に使われているマクロ以外にまだ多くのマクロがある. 一般的に制御命令 (conditional jump, hash, procedural) 以外の命令に対して、マクロが用意されている. conditional jump は if 文に、hash 命令は case 文に変換される. procedural 命令は関数呼び出しに変換される.

```
membchk()
{
    LONG op1,op2;
    op1 = ARG(1);
    op2 = ARG(2);
    return membchk2(op1,op2);
}
membchk2(op1,op2)
LONG op1,op2;
{
    LONG op3;
    DEREF(op2);
    if(!ISLIST(op2))
      return FALSE;
    FETCH_VAR(op3);
    FETCH_VAR(op2);
    DEREF(op1);
    DEREF(op3);
    if(!identical(op1,op3))
      goto label1;
    return TRUE;
   label1:
    return membchk2(op1,op2);
}
```

図 2: membchk の C プログラム

4 最適化

TOAM 命令を C へ変換した後、もとは分割不可能であった命令が分割可能となることがある. 例えば、conditional jump は、DEREF マクロと if 文に分けられ、変換されたプログラムから不要な命令を取り除くことができる. この節では、tag 操作の減少および dereference の減少について述べる.

4.1 tag 操作の減少

以下の例を考える.

```
add(T,X,Y). /* T = X + Y */     (1)
add(W,T,Z). /* W = T + Z */     (2)
```

この 2 つの命令はまず X と Y を足して、一時変数 T に入れ、そして T と Z を足して W に入れる. 最初の命令 add(T,X,Y) はまず X と Y の tag を外し、次に加算し、最後に加算の結果に tag を付け T に入れる. T に付けられた tag は、すぐ次の命令によって外される. この 2 つの命令は、tag 外しを 4 回および tag 付けを 2 回行なう. X,Y,Z および W に対する C の変数がそれぞれ x,y,z および w とすると、この 2 つの命令は次の代入文に変換できる.

$$w = MAKENUM(NUMVALUE(x) \quad (1)$$
$$+ NUMVALUE(y) + NUMVALUE(z)); \quad (2)$$

この文は、tag 外しを 3 回および tag 付けを 1 回行なう.

4.2 dereference の減少

上の例 (membchk/2) について考える. ループの中で第一引数 (op1) に DEREF を行なっているが、op1 はループの中で変化しないため余分である. そのため、DEREF(op1) はループの外に出す. この例では、op1=ARG(1) の後に入れることにより、DEREF が 1 度しか評価しなくなる.

5 評価

この節では、本論文で提案した新しいコンパイラをネイティブコードコンパイラである Sicstus Prolog 2.1 および TOAM のバイトコードコンパイラと比較する. 比較には、membchk(100000 の要素), nreverse(500 の要素), 8-queen(全解探索), quicksort(600 個の要素) の 4 つのプログラムを用いる. これらのプログラムには,8-queen 以外にすべて決定的である. 表 1 はコンパイルされたコードの実行時間を示す.

表 1: 評価結果 (単位 ms)

	Program	Sicstus	TOAM	TOAM + C
1	membchk	930	600	220
2	nreverse	650	1010	430
3	8-queen	630	480	230
4	quicksort	190	130	70

この結果について、以下の二点が分かる. 第一にすべてのプログラムに対して本論文で提案したコンパイラがもっとも速いことである. 元の TOAM コンパイラと比べると、2 倍程度の速度向上を得た. 第二に、nreverse 以外のプログラムに対して TOAM コンパイラが Sicstus より速いことである. これは、TOAM と WAM が違うことと TOAM のコンパイラがモード情報を利用することによると考えられる.

6 結び

本論文では、決定的なプログラムを C に変換する方法を提案した. この方法により、プログラムの実行がかなり高速となった.

今後の課題として、他の最適化 [2] も取り入れる. また、他の変換プログラム [3] と比較する.

参考文献

[1] H. Ait-Kaci, *Warren's Abstract Machine*, MIT Press,1991.

[2] S.K. Debray, *A Simple Code Improvement Scheme for Prolog*, J. Logic Programming, 1992, pp.57-88.

[3] H.C.R. Lock, *Issues in the Implementation od Prolog, and their Optimization*, Microprocessing and Microprogramming 32 ,1991,pp.505-514.

[4] N.F. Zhou, *Global Optimization in a Prolog Compiler for the TOAM*, to apear J. Logic. Programming, 1993.