ＰＳＩ－Ⅱのコンパイラ

4Q-9

近藤誠一　三菱電機（株）
佐藤泰典　沖電気工業（株）
近山隆　（財）ＩＣＯＴ

1．はじめに

第５世代コンピュータプロジェクトの一環とし、ＩＣＯＴにおいてマルチＰＳＩの要素プロセッサを開発した。このプロセッサはＷＡＭ[1]に基づく機械語命令を実装し、スタンド・アロン・システムＰＳＩ－Ⅱとしても用いられている。ＰＳＩ－Ⅱは、ＰＳＩ同様、オペレーティングシステムとしてＳＩＭＰＯＳを搭載している。ＳＩＭＰＯＳはＰｒｏｌｏｇにオブジェクト指向機能を付加した言語ＥＳＰをサポートしており、それ自身すべてＥＳＰで書かれている。本稿ではレジスタ割り付け、及び、構造体のインデキシング等の最適化手法を中心にＰＳＩ－Ⅱのコンパイラについて示す。

2．概要

ＷＡＭ命令セットは以下の点が特徴的である。
・述語呼び出しの引数としてレジスタを用いる。
・種々のインデキシングの手法により、高速に該当するクローズを選択する。

したがって、コンパイラは、レジスタ割り付け等高度な最適化をはかることができる。反面、実用的な速度でコンパイルできることも重要である。ＰＳＩ－Ⅱでは以下のような手順でＥＳＰプログラムがコンパイルされる。
・オブジェクト指向言語でいわれるクラス単位でクローズを抽出する。クラスがひとつのコンパイル単位となり、クラス内の呼び出しは相対番地で、クラス間の呼び出しは原則としてメソッドと呼ばれる動的な呼び出しで行われる。
・クローズ単位でコンパイルする。
・述語単位にインデキシング命令を生成する。
・ラベルの解決、コード生成を行う。
・継承解決を行い、オブジェクト指向機能のための各種テーブル、クラスオブジェクトを生成する。

なお、ＰＳＩ－Ⅱのコンパイラ自身、すべてＥＳＰで書かれている。

```
A＊  …  temporary 変数（レジスタ）０から始まる。
Y＊  …  persanent 変数。０から始まる。
get __XX V0,V1  …  V0からV1へget
put __XX V0,V1  …  V1をV0へput
```

図１．　記法

3．クローズのコンパイル

クローズのコンパイルは以下の方針で行われる。
・引数のget は前から順に、put は後から順に行う。[2]のようにクローズの性質により順序をかえることはしない。
・組込述語に関しては、それぞれの述語専用の命令に対し、任意の引数レジスタで入出力を行う。（図２．括弧内の命令は省略可能である。）　入力引数はput 系命令で与え、出力引数は組込述語命令に続くget 系命令で得られる。また、組み込み述語の前後では出力引数を除きレジスタの値は元の値が保証される。
・図３のようにレジスタ上にある定数や、変数を優先的に用いる。

4．リストの生成とレジスタの制限

リストを引数として生成する際、図４の（２）のように前方から生成する方法と（３）のように後方から生成する方法が考えられる。ＰＳＩ－Ⅱでは、合成命令 unify_listの導入もあり、前方から生成するほうが速度、コード量の面で有利である。しかし、図４のように、各要素が構造体の場合それらの生成のためにより多くのレ

```
                    p(X,Y) :-
                        add(Y,X,Z),
      ↓  ↓  put          q(X,Z).
add (X, Y, Z)
         ↓  get
                    ( put_value A0,A0 )     % X
                    ( put_value A1,A1 )     % Y
                      add(A1,A0,A1)
                    ( get_tr_variable A1,A1 )  % Z
                    ( put_value A0,A0 )     % X
                      execute q/2
```

図２．組込述語のレジスタ割り付け

```
p(S,T) :-
    q(S,T),
    U is T ／＼ 16#"7FFFFFFF",
    r(16#"7FFFFFFF",T,U).

allocate(1)
get_variable A1,Y0  % get T
call q/2
put_value A1,Y0     % put T
put_constant A0,7FFFFFFF
and(A1,A0,A2)
execute_with_deallocate r/3
```

図３．レジスタ上の定数の利用

Compiler of PSI-II
Sei-ichi KONDOH.　MITSUBISHI ELECTRIC Corp.
Yasunori SATOH.　OKI ELECTORIC INDUSTORY Corp.
Takashi CHIKAYAMA　ICOT

ジスタを必要とする。したがって、レジスタの個数の制限により、以下のようにしていずれの手法を選択するかを決定する。コンパイラ自身もＥＳＰにより記述されているので、それぞれの条件が満足されなかった時点で失敗し、次のオルタナティブに進む。

（１）変数を含まない構造体の場合は特別な命令を用いて、直接、put，get，unify を行う。

（２）前方からリストを生成する。レジスタの制限に達した時点で失敗する。

（３）後方からリストを生成する。レジスタの制限に達した時点で失敗する。

（４）レジスタ番号を決定する際、最適化よりも、あいている小さな番号のレジスタを優先する。

（５）（４）で制限を越えたものについてpermanent 変数とする。（環境フレームに退避させる。）

ＰＳＩ－Ⅱでは３２個のレジスタが用意されているので、大部分は（２）で充分である。多くのレジスタを必要とするのは、上記のように構造体がネストしている場合と組み込み述語が連続して現れた場合等が考えられる。

５．構造体のインデキシング

ＥＳＰではスタック上の構造体としてベクタをサポートしている。ベクタは要素数固定の配列である。先頭要素がアトムで要素数が２以上のものについては複合項と同様の記法を許している。｛Ｘ，Ｙ｝，ｆ（１），｛ｇ（ａ）｝等がベクタである。｛ｆ，１｝とｆ（１）は等価である。関数子は必ずしもアトムとは限らないので、そのインデキシングの方法も従来のものとは異なる。複合項の場合、関数子とその引数個数により一意に該当するものが決定されるが、ベクタの場合、先頭要素として未定義変数を含め、すべてのデータタイプが許される。したがって、ベクタ｛Ｘ，Ｙ｝はｆ（０）、ｇ（ａ）、｛１，２｝のいずれともユニファイが成功する。

ＰＳＩ－Ⅱではベクタをその先頭要素と長さを元にアトミックデータにコード化して、その後は定数のインデキシングと同様の手法を用いることができるようにした。たとえば、ハッシュ付きのインデキシングもコード化されたアトミックデータを用いて実現することができる。

コード化用の命令encode_vectorは先頭要素により、以下のように処理される。

encode_vector Ai,Aj,Lelse,Lvar

アトミック：　Aiの内容をコード化してAjに入れ、次命令会令に進む。

未定義変数：　Lvarにジャンプする。

その他：　Lelse にジャンプする。

この命令はswitch_on_term等でAiがベクタであることが保証されている場合に限る。

Lvarの先はｉに対応するヘッド引数がベクタであるクローズすべてが選択され、Lelse の先は、先頭要素がアトミックでないベクタであるクローズが選択される。図５に例を示す。

６．おわりに

レジスタ割り付け、構造体のインデキシングを中心にＰＳＩ－Ⅱのコンパイラについて示した。５のベクタのインデキシングの手法は、リストの先頭要素をキーにしたインデキシングに容易に拡張することができる。

［参考文献］

[1] D.Warren "An Abstract Prolog Instruction Set" AI Center,SRI International1983

[2] S.Abe et al.,"A New Optimization Technique for a Prolog Compiler." Compcon 86 Spring

```
p(X) :- q({a(X),b(X),c(X),d(X)})

        (1)

put_vector_2 A1     % a(X)
unify_atom a
unify_l_value_t A0
put_vector_2 A2     % b(X)
unify_atom b
unify_value_t A0
put_vector_2 A3     % c(X)
unify_atom c
unify_value_t A0
put_vector_2 A4     % d(X)
unify_atom d
unify_value_t A0
put_list A0         % [
unify_value_t A1    % a(X)
unify_list          % ,
unify_value_t A2    % b(X)
unify_list          % ,
unify_value_t A3    % c(X)
unify_list          % ,
unify_value_t A4    % d(X)
unify_atom []       % ]
execute q/1

        (2)
```

```
put_vector_2 A1     % d(X)
unify_atom d
unify_l_value A0
put_list A2         % [
unify_value A1      % d(X)
unify_atom []       % ]
put_vector_2 A1     % c(X)
unify_atom c
unify_value A0
put_list A3         % [
unify_value A1      % c(X)|
unify_value A2      % [d(X)]]
put_vector_2 A1     % b(X)
unify_atom b
unify_value A0
put_list A2         % [
unify_value A1      % b(X)|
unify_value A3      % [c(X)..]]
put_vector_2 A1     % a(X)
unify_atom a
unify_value A0
put_list A0         % {
unify_value A1      % a(X)|
unify_value A2      % [b(X)..]]
execute q/1

        (3)
```

図４．リストの生成

```
p({1,2}).
p({}).
p(f(X)).
p({X,Y}).

    jump_on_non_vector A0,L12else,L04var
    encode_vector A0,A1,L11else,L03var
    jump_on_value A1,int2,L00
    jump_on_value A1,int0,L01
    jump_on_value A1,f/1,L02
    jump  L11
L00: try  L05     % Size = 2
    trust L11     % First is not atom .
L01: try  L07     % Size = 0
    trust L11
L02: try  L09     % f/1
    trust L11
L03: try  L05     % First is variable
    retry L07
    retry L09
    trust L11
L04: try_me_else L06
L05:  [ p({1,2}) ]
L06: retry_me_else L08
L07:  [ p({}) ]
L08: retry_me_else L10
L09:  [ p(f(X)) ]
L10: trust_me_else_fail
L11:  [ p({X,Y}) ]
L12: fail
```

図５．構造体のインデキシング