3E-4

Prolog最適化コンパイラの開発（VI）大域最適化評価と高速リスト処理命令の提案

黒沢憲一　阿部重夫　桐山薫
（日立製作所　日立研究所）

1．はじめに

我々は、Prolog高速処理計算機の基本技術を確立することを目的に、ハードウェア、命令仕様、コンパイラなどの面から研究を行っている。本稿では、コンパイラによる組込み述語の性質を利用したレジスタ割り当ての大域最適化方式[1]の評価と、高速リスト処理命令の提案を行う。

2．レジスタ割り当ての大域最適化方式

我々のProlog処理系は、dec-10 Prologとは異なり、インタプリタコードとコンパイルコードの混在時においてお互いのコードを制約無しに呼び出せる透明な言語仕様にすべく開発を進めている。このためクローズ間に渡る大域最適化は、assert,retractの問題からクローズインデクシング[2]以外には行わない方針とした。これにより、本稿で述べる大域最適化方式は、クローズ内にその範囲を限定することを前提条件としている。

大域最適化方式の考え方については、文献（1）にて述べたが、その特徴は、組込み述語の性質を利用してクローズを連続実行ブロックに分類する点にある。すなわち、図1の第2クローズ［8－queenの最多実行クローズ］を例に説明すると、Tick & Warrenの考え方[3]では、図2のごとく2ブロックに分類されるため変数A，Lは Permanent変数となる。しかし、大域最適化方式では第1ゴール' ＝￥＝' が連続実行可能な組込み述語であることから図3のごとく1つのブロックになるため、すべての変数はレジスタに割り当てることが可能なレジスタ変数になる。これにより、命令の省略やメモリアクセスの少ない命令を生成することができる。

```
1．notmem (_, [] ) ：－　!．
2．notmem (A, [B|L] ) ：－A=￥=B, notmem (A, L)
```

図1　8－queenの一部のプログラム

```
notmem(A, [B|L]):-A=￥=B ,  notmem(A, L)
```

図2　Tick & Warren方式によるブロック

```
notmem(A,  [B|L]) :-A=￥=B,  notmem(A, L)
```

図3　大域最適化方式によるブロック

3．高速リスト処理命令の提案

Warrenの命令セットを用いて、図4のルールのリスト部を展開すると図5の命令コードが生成されることが知られている。[4]

```
test：－do ( [birds, fly])
```

図4　リスト処理ルール

```
1．  put-list X4
2．  unify-constant fly
3．  unify-nil
4．  put-list A1
5．  unify-constant birds
6．  unify-value X4
```

図5　Warrenによる生成コード

すなわち、リストのcdr部のアドレスを命令1にてレジスタX4に保持した後に、cdr部を命令2～3を用いて生成し、命令4～6にてリストのcar部を生成し、レジスタX4を介してcdr部とのリンクを行っている。このように、Warrenの命令セットを用いてリストを展開する場合、cdr部を一度ワークレジスタに保持して、再度参照するため、n要素から成るリストの命令数Lは

$$L = 3n$$

となり、多くの命令を必要とする欠点がある。

そこで我々は、Prologがリスト処理言語であることからその重要性を考え、

$$L = n + 2$$

を実現するunify-list命令を提案する。
この命令は、図6に示すごとくリストの第1要素と最終要素を除いた要素に対応した使い方をする。

```
1．  put-list A1
2．   unify-constant birds
3．  unify-list-constant fly
4．  unify-nil
```

図6　unify-list命令の使い方

すなわち、unify-list命令は、①readモード時、Sレジスタを用いて wk1←deref ((S)) を行い、もし tag-part (wk1)＝listならば (data-part(wk1)) と unify-list命令のオペランドをunifyする。
［tag-part(wk1)≠listの場合は省略］

Optimized Prolog Compiler(VI)　Global Optimization and High Speed List Processing Instruction
Kenichi Kurosawa, Sigeo Abe, Kaoru Kiriyama,　HITACHI, Ltd.

②Writeモード時、Heapにリストポイントと命令オペランドを書き込む。

(H) + ← Tag(list)^data(H+1) ;

(H) + ← 命令のオペランド

以上述べたunify-list命令により命令コード数の大幅な削減と高速化が実現できる。

4．評価

(1) 大域最適化の効果

quick sortにおける本方式の効果[1]は、Warren方式に比較して1.24倍であったが、本節では、8－queenを用いて大域最適化の効果を明らかにする。

8－queenで最とも多く実行される図1の第2クローズをWarren方式と大域最適化方式を用いて生成したコードを、それぞれ図7，8に示す。

```
1.  allocate
2.  get-variable A, A1
3.  get-list A2
4.  unify-variable A2
5.  unify-variable L
6.  not-equal A1, A2
7.  put-value A, A1
8.  put-value L, A2
9.  deallocate
10. execute notmem,A2
```

図7 Warrenのコード

```
1.  get-list A2
2.  unify-variable A3
3.  unify-variable A2
4.  not-equal A1, A3
5.  execute notmem,A2
```

図8 大域最適化のコード

このように大域最適化のコードは5命令で良いことがわかる。

以上の結果、8－queenにおける大域最適化の効果を表1に示す。この表は、最適引数選択によるクローズインデクシングを前提にして性能比較を行ったものである。

表1 大域最適化の効果（単位：性能比）

方 式	最適引数 + Warren方式	最適引数 + 大域最適化
8－queen	1	1.6倍

このように、大域最適化方式により8－queenの場合1.6倍高速化することがわかった。

以上から、Warrenの方式と我々の方式を8－queenを用いて評価した総合評価を表2に示す。

表2 総合性能 （単位：性能比）

方 式	Warren	最適引数方式	最適引数方式 + 大域最適化方式
8－queen	1	2.1倍	3．4倍

(2) unify-list命令の効果

文献（5）記載のconslistベンチマークを用いて、unify-list命令の効果を評価した。

このベンチマークは、25要素のリスト処理を行うプログラムで、Writeモードで動作する。

まず、Warrenの命令セットを用いた場合のconslistのリスト処理に要する命令数Lは、

L＝3＊25＝75命令

であるのに対し、unify-list命令を用いた場合は、

L＝25＋2＝27命令

と、命令数にして 48命令削減できた。

また、命令数削減効果に伴い高速化も可能となった。その効果を表3に示す。

表3 unify-list命令の効果 （単位：性能比）

方 式	Warrenの命令	unify-list命令
conslist	1	2.1倍

5．おわりに

Warrenの処理方式及び命令セットの拡張とその効果について述べた。本方式により、重要なベンチマーク quick sort，8－queenにおいて1.24～1.6倍高速化できることがわかった。また、ハードウェア構成によるがunify-list命令により、高速リスト処理を実現できる見通しを得た。

6．参考文献

1) 情報処理第32回全国大会 3F1～3F4

2) 情報処理第33回全国大会 3E3

3) E.Tick; " Towards a Multiple Pipelined Prolog Processor ". High Level Computer Architecture '84

4) D.H.D.Warren; " An Abstract Prolog Instruction Set " Tech Note 309, AI Center, SRI, 1983.

5) 奥乃； " prologベンチマークコンテスト " 記号処理 28-1 Jun, 1984.