内蔵型 Prolog プロセッサ IPP の最適化コンパイル方式の提案と性能評価†

桐山 薫†† 阿部重夫†† 黒沢憲一††

Prolog の高速処理を実現し，かつ既存のソフトウェアとの接続を可能とするために，汎用アーキテクチャ上に Prolog 高速処理機構を付加した内蔵型 Prolog プロセッサ IPP の開発を進めている．本論文では，この IPP の高速化を図る最適化コンパイラ方式と各方式の性能評価結果について述べる．Warren の命令セットをもとに高速化のための命令拡張を行うとともに以下の最適化方式を開発した．（1）次の条件を満たす引数によりインデキシングを行う最適引数によるインデキシング方式，①引数の種類によりユニフィケーションするクローズが1つに決まる引数，②第1ゴールの var (X), integer (X) 等の組込述語により種類が制限される引数，③定数，構造体を2個以上含む引数，（2）構造体およびリストのハッシングの時，その第1要素のハッシュ値まで参照する 2-level indexing 方式，（3）caller の引数が変数の時，他の引数でインデキシングする 2-way indexing 方式，（4）命令展開の実行順序最適化方式等のコード生成最適化方式．さらにモード宣言を用いて，read-mode の引数から最適引数を選択する等の拡張を行った．一般的なプログラムにより上記のインデキシング方式を静的評価した結果，97.8% の述語において最適な引数でインデキシングでき，モード宣言を用いると 98.9% になった．また，ベンチマークにより推論性能を評価した結果，append ではこの方式による効果はなかったが，より一般的なプログラムである q-sort, 8-queen において，各々 Warren 方式の 1.2 倍，2.8 倍の高速化が得られた．

1. まえがき

近年，知識処理機能を備えた計算機の開発が進められ，知識処理言語として Prolog が注目されている．Prolog は，従来の逐次型言語と比較して，プログラムの記述，およびメンテナンスが容易であるが，ユニフィケーションやバックトラックなどの非決定的な処理やタグ付きデータ構造を特徴とするため，実行速度が遅いという欠点がある．

このため，実行速度の高速化を図るために Warren が提案した Prolog 命令セット[1]をベースとした専用マシン，あるいは汎用マシン上の処理系の開発が進められている[2)~6)]．我々は，高速化とともに，既存の手続型言語で構築したソフトウェアの接続が必要であるという観点から，汎用マシン上に Prolog 高速処理支援機構を付加した，内蔵型 Prolog プロセッサ IPP の開発を進めている[7),8)]．

本論文では，最初に IPP が採用した Warren の高速化方式について概説し，次に新たに開発したProlog 最適化方式について論じ，最後にその性能評価結果について述べる．

† Optimized Compiler for Integrated Prolog Processor IPP and Performance Evaluation by KAORU KIRIYAMA, SHIGEO ABE and KEN-ICHI KUROSAWA (Hitachi Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所日立研究所

2. Warren の処理方式

Warren の高速化の考え方は次の3つに要約される．

1) 呼び出し側のゴール (caller) に対するクローズの選択は，述語名が同じで，かつ第1引数の種類 (Tag) が，**表1**に示すようにユニフィケーション可能であるクローズの中から選択する Tag indexing 方式を採る．さらに caller の第1引数が定数または構造体の時は，その定数または構造体名でハッシングを行う．

2) クローズ内のヘッドと第1ゴール，または第2ゴール以後の1つのゴールにしか現れない変数を一時変数 (tvar : temporary variable) とし，2つ以上のゴールに現れる変数を大域変数 (pvar : permanent variable) とする．tvar はレジスタに割り当てるだけでよいため，メモリアクセス回数が減少して高速化を図ることができる．

3) caller の引数は引数レジスタ (AGi : Argument Register i) にセットしてからユニフィケーションを実行する．このようなレジスタベースの処理は，メモリアクセス回数が減少して高速化を図ることができる．

以上を実現するために Prolog 命令として，インデキシング等を行う制御命令，AGi にセットする put 命令，AGi とユニファイする get 命令，リスト，構造体の要素とのユニフィケーションを行う unify 命令を持つ．

表 1 caller の引数に対するユニフィケーション可能な引数の種類
Table 1 Unifiable argument for caller's argument.

caller の引数の種類	ユニフィケーション可能な引数の種類
変 数	変数，定数，リスト，構造体
定 数	変数，定数
リスト	変数，リスト
構造体	変数，構造体

3. コンパイラの最適化方式

前章からも明らかなように，Prolog の高速処理を実現するには，クローズ選択時の対象となるクローズを限定し，ユニフィケーションが成功するクローズを早く見つけ出すクローズ間最適化，およびゴールの各引数レジスタへ値を最小ステップでセットするクローズ内最適化が必要となる．IPP の Prolog 命令は Warren の命令をベースとしているが，さらに，クローズインデキシング命令の強化，命令展開の順序付け，算術演算等の組込述語の命令化等を行っている[7]．

以下では我々が開発した最適化方式について述べる．

3.1 クローズ間最適化方式

Warren 方式のインデキシングよりユニフィケーションするクローズをさらに限定するインデキシングを実現するため，以下に示す3つの最適化方式を開発した．

（1） 最適引数によるインデキシング方式

最適引数によるインデキシング方式とは，各述語ごとにインデキシングの効果の高い引数をコンパイル時に決めて，Tag indexing の高速化を図る方式である．Warren 方式では，各クローズの第1引数に変数が多い場合，ユニフィケーション可能なクローズ（別解）が多くなってしまう．そこで，Tag indexing や Hashing を実現するときに別解が少なくてすむ最適なインデキシング引数を次のように決める．

①Tag indexing によってユニフィケーションするクローズが1つに決定する引数，

②項の現在状態を判定する組込述語で指定される引数，

③定数，構造体を2個以上含む引数，

④これらの条件を満たす引数がない場合は第1引数．

最適引数によるクローズインデキシングを実現するために call, execute 命令は switch_on_term 命令の機能を組み合わせ，オペランドに引数番号を加えた．

第1ゴールが項の現在状態を調べる組込述語（以下，タイプチェック述語と呼ぶ）で，ヘッドの引数Xのタイプを宣言しているとき，ユニフィケーション可能な caller の引数の種類は**表2**のように制限される．タイプチェック述語で指定した引数は caller の引数の種類を限定することができるので，インデキシング引数として適している．

表 2 タイプチェック述語で限定される caller の引数の種類
Table 2 Unifiable argument limited by type-check predicate.

タイプチェック述語	caller の引数の種類
var (X)	変 数
nonvar (X)	定数，リスト，構造体
atom (X)	定数（文字列）
integer (X)	定数（整数）
real (X)	定数（実数）
atomic (X)	定数（文字列，整数，実数）

例えば，**図 1**（a）に示すプログラムの最適なインデキシング引数は，タイプチェック述語で指定されている第1引数となる．このプログラムの生成コードは，従来方式では，図1（b）のように caller の種類にかかわらず2つのクローズとユニフィケーション可能となる．しかし，タイプチェック述語を考慮することにより，図1（c）のように caller が変数のときは2つのクローズとユニフィケーション可能であるが，caller が変数でないときは下のクローズ1つだけになる．また，var (X) のコードの生成も不要となる．

この最適引数による Tag indexing の機能を高めるために，2-level indexing 方式，2-way indexing 方式を導入した．

（2） 2-level indexing 方式

2-level indexing 方式は，caller のインデキシングする引数が構造体のときは，（構造体名のハッシュ値＋第1要素のハッシュ値）/ハッシュテーブルサイズをハッシュ値として，リストのときは，（第1要素のハッシュ値）/ハッシュテーブルサイズをハッシュ値とする．これによりインデキシングする引数が構造体，およびリストの多いクローズであっても，選択する別解を減少することができ高速化を図ることができる．

そこで，構造体名とその第1要素によりハッシングする命令を switch_on_str_arg 命令，リストの第1要素によりハッシングする命令を switch_on_lst_arg 命令とし，それぞれオペランドにハッシュテーブルサイズを加える．構造体名が等しく第1要素が異なるプログラムの場合，この方式により従来方式よりも効率

的なハッシングができる．しかし，構造体の第1要素が等しいプログラムの場合，構造体だけを参照してインデキシングすれば十分である．また，**図 2**(a)のプログラムのように第1要素に変数を含む場合は，構造体名だけでハッシングするときのコードは図2(b)となるが，構造体名と第1要素でハッシングするときのコードは図2(c)となり，第1要素が変数であるクローズは，第1要素がどんな値であってもユニフィケーションできるようにするため，すべてのハッシュ値に対してエントリされてしまう．

そこで，プログラムの最適引数がリストの場合，リスト第1要素の種類が2種類以上あれば，2-level indexing を行うが，構造体の場合は，第1要素に変数を含まず，構造体名の種類数<構造体第1要素の種類数のときにかぎり，2-level indexing を行う．

(3) 2-way indexing 方式

2-way indexing 方式は，caller のインデキシングする引数の種類が変数の時にリニアサーチになることを防ぐため，(1)で示した引数の選択条件にあてはまるような引数が2個以上あるときは，2個の引数に対応するハッシュテーブルを作成し，1番目に選んだ引数の caller が変数のときは，2番目に選んだ引数によってインデキシングできるようにする方式である．

図 3 に，2-way indexing の例を示す．caller の第1引数が変数のとき，ラベル V1 にジャンプし，第1引数でなく第2引数により Tag indexing を実行する．

3.2 モード宣言を考慮したクローズ間最適化方式

モード宣言とは，ユーザが前もって述語の各引数に read-mode（呼び出し側の値が決まっていて，引数の判定または引数の代入処理をする），write-mode（呼び出し側の値が変数で，その変数への書き込み処理をする），および read/write-mode（read-mode にも write-mode にもなる）のいずれかを宣言することをいう．この宣言によりプログラミング時のユーザの負担は増大するが，述語の条件が限定されるため，最適化方式においてその効率を高めることができる．

最適引数によるインデキシング方式で，インデキシングに最適な引数を選択する時にモード宣言を考慮すれば，caller が変数で必ずリニアサーチになってしまう write-mode の引数ではなく，caller の値が必ず決

```
～, p(X, Y).
p(X,a):-var(X).
p(X,b).
(a) ソースプログラム

   execute  AG1(V1,V1,V1,V1)
V1:try_me_else  V2
   get_constant  a,AG2
   var  AG1
   proceed
V2:trust_me_else  fail
   get_constant  b,AG2
   proceed
   (b) 生成コード(従来方式)

   execute  AG1(V1,el,el,el)
V1:try_me_else  V2
   get_constant  a,AG2
   proceed
V2:trust_me_else  fail
el:get_constant  b,AG2
   proceed
   (c) 生成コード(新方式)
```

図 1 Type-check indexing 方式の例
Fig. 1 Example of type-check indexing.

```
p(f(X)).
p(g(b)).
(a) プログラム例

   switch_on_structure
        2(f:s1,g:s2)
s1:get_structure  f,AG1
   unify_variable  AG1
   proceed
s2:get_structure  g,AG1
   unify_constant  b
   proceed
(b)生成コード(構造体名)

   switch_on_str_arg
        4(f:s11,g:s1,gb:s1)
 s1:try_me_else  s2
s11:get_structure  f,AG1
   unify_variable  AG1
   proceed
 s2:trust_me_else  fail
   get_structure  g,AG1
   unify_constant  b
   proceed
(c)生成コード(構造体名と第1要素)
```

図 2 構造体のハッシング例
Fig. 2 Example of structure hashing.

```
～, p(X, b).
p(x,a).
p(y,b).
(a) ソースプログラム

   put_variable  X,AG1
   put_variable  b,AG2
   execute  AG1(V1,C1s,fail,fail)
 V1:execute  AG2(C1,C2s,fail,fail)
C1s:switch_on_constant  AG1,2,(x:c11,y:c21)
C2s:switch_on_constant  AG2,2,(a:c11,b:c21)
 C1:try_me_else  C2
c11:get_constant  x,AG1
   get_constant  a,AG2
   proceed
 C2:trust_me_else  fail
c21:get_constant  y,AG1
   get_constant  b,AG2
   proceed
   (b) 生成コード
```

図 3 2-way indexing 方式の例
Fig. 3 Example of 2-way indexing.

```
～, p (X, Y) , ～
p(X,a):-var(X),!.
p(_,b).
(a) ソースプログラム

 put_variable  X,AG1
 put_variable  Y,AG2
 execute  AG1,(V1,C1,C1,C1)
V1:get_constant  a,AG2
 proceed
C1:get_constant  b,AG2
 proceed
(b) 生成コード
```

図 4 モード宣言を含むインデキシング例
Fig. 4 Example of indexing using mode information.

まっている read-mode の引数を選ぶことができる.

また, 第1ゴールがタイプチェック述語の時, タイプチェックに使われる引数以外の引数が write-mode であるか, read-mode でもクローズ中に1個しか現れない変数 (void 変数) の場合は, その組込述語の後にカット(!)があれば, そのクローズの別解がなくなる.

例えば, 図 4(a)に示すようなプログラムで, 第2引数が write-mode であるとき, mode 宣言を考慮し第1引数でインデキシングすると, Tag indexing により, 上のクローズとユニフィケーションするときは, 第2引数は write-mode なので失敗することがなく, 第1引数は変数で var(X) が成功しカット命令を実行するので別解がなくなる. 下のクローズとユニフィケーションするときは, 第1引数が変数以外でやはり別解はない. すなわち, Tag indexing のみで選択するクローズが1つに決まる. このときの生成コードを図4(b)に示す.

このように, 引数のタイプチェックをしてクローズを1つだけ選択し, 他の引数で失敗しない場合は, 選択するクローズを分類するタイプチェック述語の後に, カット(!)をいれるようにすれば, Tag indexing により組込述語が成功した時, 他のクローズへの別解がなくなる.

3.3 クローズ内最適化方式

クローズ内最適化方式は, レジスタを効率的に使う Warren 方式に伴い, さらに冗長命令, 一時退避命令の削除, メモリアクセス回数の削減, 実行ステップ数の削減を図る方式である. このため, 次の2つの最適化方式を開発した.

(1) 大域最適化方式

大域最適化方式は, 頻繁に使われる組込述語を命令化することにより, 組込述語を越えてレジスタ割り当てを行うもので, メモリアクセス回数を削減することができる. 詳細については別報で述べる.

(2) 連結部分グラフによる実行順序最適化方式

最初のヘッドと第1ゴールの処理で, 変数の値をヘッド部でユニフィケーションした後, 第1ゴールの AGi にその値をロードする時レジスタ競合が生じる場合は, 一時的に他のレジスタやメモリに値を保持しておかなければならない. 連結部分グラフ (DSG : Directed Sub Graph) による実行順序最適化方式は, まず, ヘッド部の引数レジスタから第1ゴールの引数レジスタへの移動を示す変数集合 i の有向グラフ (DSGi) をつくる. ここで, 変数集合とは, 同一リスト/構造体に現れる変数の集合または引数変数のことを指す. この有向グラフを用いて連結部分グラフを作り, レジスタ競合が少なくてすむように変数の処理順序を決める[7]. 例えば, 図 5(a)のプログラムで左から順に変数の展開を実行すると図5(b)に示すコードが生成される. ステップ3, 4はレジスタ競合による一時退避命令で, このため, ステップ6, 7が必要になる. そこで, DSG による実行順序最適化方式によりコードを生成すると有向グラフは次のようになる.

DSGx, x_1 = {1}→{1, 2}
DSGy = {2}→{3}
DSGz = {3}→{4}

この有向グラフからレジスタ競合の生じない実行順序① DSGz ② DSGy ③ DSGx, x_1 が決まる. この順序に従って生成したコードが図5(c)で命令は2ステップ減る.

上記の順序付けに定数は含まれていないが, 定数や

```
p([X|X1],Y,Z):-q(X,X1,Y,Z).
(a) ソースプログラム

1. get_list  AG1
2. unify_variable  AG1
3. unify_variable  AG5
4. put_value  AG2,AG6
5. put_value  AG3,AG4
6. put_value  AG5,AG2
7. put_value  AG6,AG3
8. execute  AG1,q
(b) 左から順に展開したときの
    生成コード
1. put_value  AG3,AG4
2. put_value  AG2,AG3
3. get_list  AG1
4. unify_variable  AG1
5. unify_variable  AG2
6. execute  AG1,q
(c) DSGによる実行順序最適化方式
    による生成コード
```

図 5 生成コードの比較
Fig. 5 Comparison of generated codes.

同一変数のユニフィケーションを先に実行することにより，ヘッド部の失敗を早く見つけることができる．そこで，次のように実行順序を決める．

①ヘッド部の定数または定数リスト/構造体のユニフィケーション
②ヘッド部に現れる同一変数のユニフィケーション
③実行順序最適方式により順序付けした，ヘッドと第1ゴールに現れる変数のユニフィケーションとロード
④組込述語の処理
⑤第1ゴールにしか現れない変数のロード
⑥第1ゴールにしか現れない定数または定数リスト/構造体のロード

3.4 モード宣言を考慮したクローズ内最適化方式

上で述べたように，変数のユニフィケーションやレジスタへのセットの前に，ヘッド部に現れる定数のユニフィケーション処理を先に実行することによって，ユニフィケーションに成功するクローズ選択の高速化を図っている．この時，モード宣言を考慮すれば，caller が定数でクローズを選択するときの判定に使われている read-mode の定数および定数リスト/構造体のユニフィケーションだけを先に実行することができ，変数への代入処理となる write-mode の定数および定数リスト/構造体のユニフィケーションを先に実行することを防ぐことができる．同様に，変数についても read-mode 同士のユニフィケーション処理やタイプチェックの組込述語の処理を先に実行することにより，クローズ選択の判定処理が優先される．

ただし，Tag indexing や Hashing によって別解がないクローズについては判定を先に実行する必要はないので，レジスタを解放するために write-mode であっても定数のユニフィケーション処理を先に実行する．

4. 性能評価

第3章で述べた最適化方式について，ベンチマークプログラム (append, q-sort, 8-queen) と一般的なプログラムを使い，最適化コンパイラによって生成した Prolog 命令を評価した．なお，推論性能は，IPP[7),8)] を使って計測し，組込述語については，1推論として数えた．

4.1 ベンチマークによる性能評価

3つのベンチマークプログラムに対する評価結果と推論性能を**表3**に示す．append では性能向上はないが，q-sort ではクローズコードの最適化により，1.2倍，8-queen では最適引数によるインデキシングとクローズコードの最適化により 2.8 倍の高速化が実現できた．

表3 最適化による推論性能と向上率
Table 3 Ratio of speed up by optimization and LIPS.

プログラム	方式1	方式2	方式3	推論性能
append	1	1	1	1125 kLIPS
q-sort	1	1	1.2	458 kLIPS
8-queen	1	2	2.8	1133 kLIPS

方式 1: Warren 方式
方式 2: 最適引数によるインデキシング
方式 3: 最適引数によるインデキシングとクローズコードの最適化

4.2 一般的なプログラムによる性能評価

4.2.1 評価プログラム

一般的なプログラムによる性能評価では，我々が開発したコンパイラのプログラムの一部を使って静的評価を行った．プログラムは，事実 (fact) 文が 455 クローズ，規則 (rule) 文が 559 クローズで合計 1014 クローズ，全述語数が 269 個，平均別解数 4.65 個（ただし，すべて事実 (fact) 文からなり，データベースとして使われる述語を除くと 3.4 個），平均引数個数 3.91 個である．また，このプログラムは最適化方式を意識せずに書いたものである．

4.2.2 クローズ間最適化方式の評価

(1) 最適引数によるインデキシング方式の評価

この方式により決定したインデキシング引数についてインデキシングに最適な引数かどうかを評価した結果を**表4**に示す．

これにより，第1引数による Warren のインデキシング方式と最適化方式とを比較すると，前者は 80.7% の述語が最適な引数でインデキシングするのに対して，後者は 97.8% の述語が最適な引数でインデキシングすることがわかった．さらに，モード宣言を考慮して read-mode の引数を選ぶようにすると最適な引数の選択率が 98.9% になった．ここで，残りの 1.1

表4 選択した引数の分析
Table 4 Analysis of selected argument.

第 1 引 数		192
第 n 引数 ($n \neq 1$) 77	第1引数の方が良い	3
	どちらも同じ	22
	第 n 引数の方が良い	49
	その他の引数が良い	3
合 計		269

%（3個）については，ハッシングの重なりをチェックすれば，最適引数として選択できるものであった．

また，評価プログラムにおいて，第1ゴールにタイプチェックの組込述語が現れ，その引数を最適引数として選ぶ述語は，19個（7.1%）あり，この19個の述語について平均別解数を調べてみると，平均クローズ数4.3個に対して従来方式で最適な引数を決定した場合は平均別解数が3.7個であった．これに対し，第1ゴールに含まれるタイプチェックの組込述語を考慮し最適引数を決めた場合は2.7個，さらにmode宣言を考慮すると，別解数は2.2個に減少し，Tag indexingのみでクローズの決まる述語が増加した．

（2） 2-level indexing 方式の評価

269個の述語のうち定数のハッシングを実行するswitch_on_constant 命令，構造体名のハッシングを実行するswitch_on_structure命令，および，2-level indexing 方式によりハッシングを実行する命令の出現数を**表5**に示す．switch_on_lst_arg 命令は，switch_on_str_arg 命令に比べて出現数が多く，ハッシュ値の求め方が簡単であるため，命令化の効果が高い．

（3） 2-way indexing 方式の評価

callerの第1引数が変数の場合，Warren 方式により，リニアサーチになる時と2-way indexing 方式により他の引数でインデキシングする時の検索時間についてベンチマークプログラム[9]内のcountry (X, Y) を使って評価した結果を**表6**に示す．検索時間は，評価用シミュレータのマイクロプログラムのステップ数をヒット率100%としてカウントした．

country (X, Y) は，217個のfact文からなり，各クローズは2個の定数引数を持つ．callerとして，それぞれ第1引数が変数で，1番目のfactとユニフィケーションするcountry (X, first)，110番目のfactとユニフィケーションするcountry (X, middle)，217番目のfactとユニフィケーションするcountry (X, last)，の3つの場合について調べた．インデキシングする引数を第1引数とするとリニアサーチになるので，ユニフィケーションするクローズが後になるほど実行時間が増加するが，2-way indexing 方式によってインデキシングする引数が第2引数になれば，どのcallerでも検索時間はほぼ一定になる．

4.2.3 クローズ内最適化方式の評価

（1） DSG による実行順序最適化方式

(1-1) 定数のユニフィケーション処理を先に実行する方式の評価

定数および定数リスト/構造体のユニフィケーションを先に実行するルールは，評価した全ルールのうちの211個（37.7%）であった．この211個のルールのうち，ヘッドに含まれる定数および定数リスト/構造体のmodeを**表7**に示す．

ここで示すように，write-modeの定数のユニフィケーションを先に実行するルールは91個ある．そこで，write-modeの引数のユニフィケーション処理を先に実行するときのオーバヘッドについて調べた．すなわち，write-modeの定数だけあるルールと同一述語のクローズを取り出し，各クローズが成功するまでに余分に定数ユニフィケーションを実行するクローズ数を調べる．この結果，同一述語のクローズ197個のうち，クローズが成功するまでに余分な定数ユニフィ

表5 Hashing を実行する命令の出現数
Table 5 Occurrence frequency of hashing instructions.

命　令	述語数
switch_on_constant 命令	27
switch_on_structure 命令	34
switch_on_str_arg 命令	3
switch_on_lst_arg 命令	18

表6 リニアサーチと2-way indexing 方式の検索時間比
Table 6 Search time ratio of linear search to 2-way indexing.

caller	リニアサーチ	2-way indexing
country (X, first)	1	1.5
country (X, middle)	65	0.9
country (X, last)	128	1.6

表7 定数ユニフィケーションにおける定数の mode
Table 7 Mode for unification with constant data.

定数の mode	ルール数
read_mode のみ	57
read_mode と write_mode	63
write_mode のみ	91
合　計	211

表8 実行順序最適化方式の効果
Table 8 Effect of optimal execution sequence.

方　式	平均命令数
従来方式	13.6
最適化方式	12.0

ケーションを実行するクローズ数は平均 0.85 個にすぎなかった．これは，クローズの選択を実行するとき Tag indexing や Hashing によって別解の数を減らしているためと考えられる．

(1-2) レジスタ競合を最小とする変数の実行順序最適化方式の評価

DSG を使って変数の実行順序を決めたとき，命令数が減少したルールは評価した全ルールのうちの 42 個 (9.5%) であった．この 42 個のルールについて平均命令数を調べた結果，表 8 のようになった．

5. むすび

汎用マシン上で高速な Prolog 処理を実現するため，内蔵型 Prolog プロセッサ IPP の開発を進めている．本論文では，Warren の提案した方式に対し機能を拡張したコンパイラの最適化方式について述べ，これらの最適化方式についてベンチマークプログラムと一般プログラムを使って評価した．

ルールの選択の高速化を図るために，最適引数によるインデキシング方式，2-level indexing 方式，2-way indexing 方式，DSG による実行順序最適化方式等のコード生成最適化方式を導入した．この結果，タイプチェックの述語とモード宣言を考慮したインデキシング方式では，一般プログラムの 98.9% についてインデキシングに最適な引数を選択した．また，推論性能は，ベンチマークプログラム append では，最適化方式による効果はなかったが，q-sort, 8-queen においては，各々 Warren 方式の 1.2 倍，2.8 倍の高速化を実現した．

謝辞 本研究に関して，有益な討論と助言をいただいた，日立研究所第 8 部 82 研究室，坂東忠秋室長，ならびに，評価データの収集に協力していただいた，日本ビジネスコンサルタント，菊地俊二氏に感謝します．

参考文献

1) Warren, D. H.: An Abstract Prolog Instruction Set, Technical Note 309, Artificial Intelligence Center, SRI International (Oct. 1983).
2) 中島ほか：PSI へのコンパイラ向き Prolog 命令の試験実装と評価，情報処理学会論文誌，Vol. 28, No. 10, pp. 1091-1095 (1987).
3) 斎藤ほか：AI ワークステーション (WINE) の開発 I～VII，第 35 回情報処理学会全国大会論文集，pp. 1665-1678 (1987. 9).
4) 新井ほか：Prolog コンパイラの設計，昭和 62 年度人工知能学会全国大会 (第 1 回) 予稿集，pp. 185-188 (1987).
5) Tamura, N.: Knowledge-Based Optimization in Prolog Compiler, *Proc. of FJCC '86*, pp. 237-240 (1986).
6) Kurokawa, T. et al.: A Very Fast Prolog Compiler on Multi Architecture, *Proc. of FJCC '86*, pp. 963-968 (1986).
7) Abe, S. et al.: High Performance Integrated Prolog Processor IPP, *Proc. of the 14th Int. Symp. on Computer Architecture*, pp. 100-107 (June 1987).
8) Yamaguchi, S. et al.: Architecture of High Performance Integrated Prolog Processor IPP, *Proc. of Fall Joint Computer Conf.*, pp. 175-182 (Oct. 1987).
9) 奥乃：第 3 回 LISP コンテスト及び第 1 回 PROLOG コンテストの議題案，情報処理学会記号処理研究会，28-4 (1984. 6).

(昭和 62 年 12 月 24 日受付)
(昭和 63 年 3 月 9 日採録)

桐山 薫 (正会員)

昭和 35 年生．昭和 58 年宇都宮大学工学部情報工学科卒業．同年 (株) 日立製作所日立研究所入社．Prolog 最適化コンパイラに関する研究に従事．

阿部 重夫 (正会員)

昭和 22 年生．昭和 45 年京都大学工学部電子工学科卒業．昭和 47 年同大学院修士課程 (電気工学科) 修了．工学博士．同年 (株) 日立製作所日立研究所に入社．現在同所主任研究員．昭和 53 年～54 年テキサス大学客員研究員．電力系統の解析，アレイプロセッサ，Prolog プロセッサの研究開発に従事．昭和 59 年電気学会論文賞受賞．電気学会，ソフトウェア科学会各会員．IEEE senior member.

黒沢 憲一 (正会員)

昭和 28 年生．昭和 55 年東北大学大学院工学研究科修士課程情報工学修了．同年 (株) 日立製作所入社．知識処理計算機の命令アーキテクチャ，PROLOG 言語の最適コンパイラの研究に従事．人工知能マシンアーキテクチャ，並列処理に興味をもつ．現在，同社日立研究所第 8 部研究員．電子情報通信学会会員．