PROLOG における大域的プログラム解析によらない単一化処理の述語間最適化方式

碓 崎 賢 一[†]

本論文では，述語呼び出し時に引数特性を伝達する，単純で効果的な PROLOG の単一化処理の最適化方式を提案する．提案方式では，直接出力変数と呼ぶ引数の分類と，その分類情報を格納する直接出力変数フラグを導入し，述語呼び出しでその情報を受け渡すことにより最適化を行っている．本方式は，述語単位のコンパイルで述語間の大域的な最適化が行えるという特長を持っており，不必要なデリファレンスやトレイル処理を除去することによって，出力モードの単一化の処理速度を大幅に向上させることができる．append/3 での評価により，従来の WAM に対して 3.4 倍の高速化が行え，58 MIPS のワークステーション上で，3.3 M LIPS の高い性能が得られることが明らかになった．

An Inter-Procedural Unification Optimization without Any Global Program Analysis in PROLOG

Ken'ichi Kakizaki[†]

This paper presents a simple and efficient PROLOG optimizing method using argument property information passing. This method introduces an argument property classification called direct write mode variable, and passes from caller to callee the information about the values which are put into argument registers. According to this information, our method improves performance of write mode unifications by removing unnecessary dereferencing and trailing. This optimization method realizes inter-procedural global optimization without any global program analysis. A result of performance evaluation for append/3 code compiled with this optimizing method shows an improvement of 3.4 times faster than conventional WAM based compiled code, and it realizes 3.3 M LIPS on a 58 MIPS RISC workstation.

1. はじめに

PROLOG は高い記述能力を持つ優れたプログラミング言語であるが，その高度な機能を実現するための処理コストが性能向上の阻害要因となっている．本論文では，出力モードの単一化処理に着目し，デリファレンスやトレイル処理などの冗長な処理を除去することによって，処理速度を向上させる最適化方式を提案する．

冗長なデリファレンスとトレイル処理に着目した従来の最適化方式は，使用者による宣言やプログラムの大域的な解析を必要とするという問題がある．本方式では，述語呼び出しの際に，直接出力フラグ[1]と呼ぶ引数に関する情報を述語間で受け渡すことによって，最適化処理を行っている．この方式には，各種の宣言や大域的なプログラム解析を行わない述語単位の最適化でありながら，述語間に渡る大域的な最適化が実現されているという特長がある．

append/3 での評価により，提案方式の基本的な最適化を施しただけでも，従来のコンパイルドコードに対して 2.2 倍ほど処理速度が向上することが明らかになった．また，プログラムの特性を利用した最大の最適化を行うことにより，従来のコンパイルドコードに対して 3.4 倍ほど処理速度が向上し，58 MIPS のワークステーション上で 3.3M LIPS の高い処理速度が得られることが明らかになった．

2. 出力モードの単一化処理

2.1 命令セット

WAM[2] の処理方式では，本論文での着目部分とは別に，複合項の単一化処理で頻繁に行われる入出力モードによる処理の切替が大きなオーバヘッドとなっ

† 九州工業大学情報工学部電子情報工学教室
Department of Computer Science and Electronics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology

ている．特に汎用プロセッサでは条件分岐のオーバヘッドが大きいため，最近の最適化コンパイラでは，このようなオーバヘッドを取り除く最適化が一般的に行われていると考えられる．したがって本論文では，提案方式の現実的な評価を示すために，このオーバヘッドを取り除いた Meier による拡張 WAM の命令セット[3]を基本として議論と評価を行う．

この命令セットでは WAM の基本方式とは異なり，複合項の単一化処理を深さ優先で行うことにより，頻繁に行われる入出力モードの検査とそれに伴う分岐を除去し高速化をはかっている．このために，複合項の単一化を行う unify 系の命令を入力モード用の read 系の命令と出力モード用の write 系の命令に分離してコードを生成している．単一化処理の入出力モードの切替は，モードフラグを検査するのではなく，一方の命令列を選択することによって行う．

2.2 出力モードの冗長な処理

単一化処理は，基本的には項の比較と変数への代入という非常に単純で低コストの操作で実現されている．しかしながら，比較する項の実体を求めるためのデリファレンス処理や，代入する変数の後戻りに備えたトレイル処理などのコストが高いために，単一化処理のコストが高くなっている．

図1に示した append/3 の例では，第1引数によるインデキシングで処理が決定的に行われ，単一化処理によって第3引数にリストが作成される．拡張 WAM におけるリストの単一化処理には，**図2**に処理概要を示す get_list 命令が使用される．この処理は，基本的な WAM の get_list 命令とは異なり，モードフラグの操作を行うかわりに，入力モードと出力モードに対応した read 系命令列と write 系命令列に分岐するようになっている．

```
:- append([1, 2, 3, 4], [5, 6], X).

append([X| L1], L2, [X| L3]) :-
    append(L1, L2, L3).
append([], L, L).
```

図 1　出力引数
Fig. 1　Write mode argument.

```
% get_list AN, READ_UNIFY
DEREF(AN) ;                         /* (1) */
if (IS_LIST(AN)) {
    S = DETAG(AN) ;
    goto READ_UNIFY ;
} else if (IS_VAR(AN)) {
    if (REQUIRE_TRAIL(AN))          /* (2) */
        TRAIL(AN) ;
    *AN = TAG(LIST_TAG, H) ;        /* (3) */
    S = H ;
    H += 2 ;
    goto WRITE_LABEL ;
} else
    goto FAIL ;
WRITE_LABEL:
```

図 2　get_list の処理概要
Fig. 2　Operation for get_list.

図1の例では，append/3 の第3引数に直前のゴールで作成されたリストの変数セルを直接指すポインタが与えられており，出力モードの単一化処理が行われる．この場合，デリファレンスが必要ないにもかかわらず，図2(1)の DEREF によってその処理が行われる．また，後戻りに備えたトレイルの記録は行われないものの，図2(2)の REQUIRE_TRAIL によってその必要性の有無の検査が行われる．

引数が変数の場合のデリファレンス処理では，まずタグによるデータ型検査のための条件判断が行われ，次に変数セルの実体を求めるためのメモリ参照とその内容を引数と比較する条件判断が行われる．また，トレイルの記録の必要性がないことを判断するためには，変数セルがヒープ領域にあって，選択点が生成された時点以降に確保されたことを検査しなければならず，2回の条件判断が行われる．これらの処理を合計すると，少なくとも1回のメモリ参照，4回の条件判断（分岐），それに付随した各種の処理が行われることになる．

引数が変数の場合に決定性の単一化処理で行わなければならない本質的な処理は，図2(3)以降に示される代入処理であり，汎用プロセッサで4マシンサイクル程度で行える非常に単純なものである．一方，デリファレンスやトレイルの検査処理は，不必要な処理であるにもかかわらずその数倍の処理時間を要し，プログラムの処理時間の中でも大きな比重を占めている．

2.3 デリファレンスとトレイル処理の除去

デリファレンスとトレイル処理の除去は，呼び出し側のサブゴール列において初めて出現した変数や，頭部において作成された複合項やリストに含まれる変数が初めてサブゴールの引数として与えられた場合に行うことができる．このような変数が引数として与えられた場合には，その変数に関して次のことが保証される．

1. デリファレンスが必要ない

　変数のアドレスが直接与えられているため．

2. トレイル処理が必要ない

　変数セルが選択点以降に作成されているため．

2の特性は，呼び出された述語側で選択点が生成さ

れた場合には無効になるが，その場合でも1の特性は有効である．この意味で，上記の条件を満たす変数を直接出力変数と呼ぶことにする．

直接出力変数に対する最適化方式として，使用者の宣言による方式と，プログラムの大域的な解析による方式が提案されている．宣言を用いた最適化方式[4),5)]は，プログラムの特性に熟知することを使用者に強要する上に，最適化されたプログラムのセマンティックスが変化してしまうために，宣言に誤りがあれば発見と修正が困難なバグが生じてしまうという問題がある．大域的なプログラム解析を用いた最適化方式[6),7)]では，プログラムが大きくなるにつれて述語相互の関係の解析時間が爆発的に増大するという問題がある．また，プログラムを構成している述語の特性が相互に強く関連づけられて解析されるために，PROLOG の特性を生かしたプロトタイピングに用いようとする場合に，プログラムを部分的に修正しインクリメンタルコンパイルを行うことができないという問題がある．

本論文では，これらの問題を解決する単一化処理の最適化方式として直接出力変数が引数として設定された場合に，その情報を述語の呼び出し側から呼び出された側へ伝達し，冗長なデリファレンスとトレイル処理を除去する方式を提案する．提案方式は，次のような特長を持つ．

- 述語を単位とした最適化
- 実行時の検査による動的な最適化
- 宣言を必要とせず，自動的で安全な最適化
- 述語間の大域的な最適化

3. 単一化処理の最適化

3.1 直接出力モードレジスタと操作命令

デリファレンスやトレイル処理を除去するためには，引数が直接出力変数であることが明らかになっている必要がある．このような情報を管理する目的で，拡張 WAM のレジスタセットに直接出力モードレジスタを導入する．直接出力モードレジスタは，各引数レジスタに対応する直接出力フラグの集合となっており，語長が 32 ビットのマシンでは，レジスタ上に実用上十分な 32 引数に対応したフラグを確保できる．また，Meier の拡張 WAM では，入出力モードフラグを必要としないために，入出力モードフラグに用いられていたレジスタを直接出力モードレジスタに転用することにより，余分な資源を消費することなく本提案方式を導入することができる．

直接出力フラグは，引数レジスタに引数を設定する際に操作され，直接出力変数が設定される引数レジスタに対応したものがセットされる．このような操作を行う命令を拡張 WAM の命令セットに追加する．direct_write_set は，指定された引数レジスタに対応した直接出力フラグをセットする命令で，direct_write_clear は，指定された引数レジスタに対応した直接出力フラグをクリアする命令である．呼び出し側で引数レジスタに値を設定する場合には，これらの命令を使用して，引数の特性を対応するフラグに設定する．レジスタ割当の最適化などで操作されていない引数レジスタに関しては，直接出力フラグの値を操作せずそのまま伝播させる．

直接出力モードレジスタの導入にともない，選択点フレームには直接出力モードレジスタの内容を保存する領域を付加する．また，選択点を生成する try 系の命令は，選択点に直接出力モードレジスタの値を保存するように拡張する．後戻り処理において，選択点の情報を回復させる処理ルーチンは，選択点に保存されている直接出力モードレジスタの値も回復させるように拡張する．

3.2 節の分類と単一化命令

本最適化方式では，その効果を向上させるために，述語を構成する節を2種類に分類し，それぞれに最適なコード生成を行う．節は，インデキシングを行い頭部の単一化処理が開始される時点での選択点の生成状態によって分類され，選択点が生成されていない場合には選択点非生成節，選択点が生成されている場合には選択点生成節と呼ぶ．選択点生成節か否かは，節の本体にカットがあって選択点が除去されるか否かには直接関係がない．図1の append/3 の例では，第1引数でインデキシングされる場合，2つの節は選択点非生成節に分類される．**図3**の partition/4 の例では，第1引数でインデキシングされる場合には，1番目の

```
:- partition([1, 5, 2, 6, 3, 7], 4, X, Y).

% 選択点生成節
partition([X| L1], Y, [X| L2], L3) :-
      X =< Y,
      !,
      partition(L1, Y, L2, L3).
% 選択点非生成節
partition([X| L1], Y, L2, [X| L3]) :-
      partition(L1, Y, L2, L3).
% 選択点非生成節
partition([], _, [], []).
```

図 3 節の分類
Fig. 3 Clause classification.

節は選択点生成節に分類され，２番目の節と３番目の節は選択点非生成節に分類される．

選択点非生成節と選択点生成節では，選択点の有無により出力モードの単一化に必要とされる処理が異なっている．このため，それぞれの場合に効果的な処理を行うために，直接出力フラグは選択点非生成節と選択点生成節の単一化処理で，それぞれ以下のような意味に解釈される．

1. 選択点非生成節
   引数には変数のアドレスが設定されており，代入する場合にはデリファレンスやトレイルの記録は必要ない
2. 選択点生成節
   引数には変数のアドレスが設定されており，代入する場合にはデリファレンスは必要ないがトレイルの記録は必ず行う

本最適化手法のために，get_nocp と get_cp の２種類の系統の命令を導入する．選択点生成節では get 系の命令の代わりに get_cp 系の命令を使用し，選択点非生成節では get_nocp 系の命令を使用してコードを生成する．get_nocp 系と get_cp 系の命令の機能例として，図２に示した get_list に対応する２系列の処理の最初の部分に付加する処理を**図４**に示す．図４の DWM_REG は直接出力モードレジスタ，DWM_MASK(AN) は引数レジスタ AN に対応した直接出力フラグを検査するためのマスク値を示す．get_nocp 系と get_cp 系の命令は，直接出力フラグがセットされている場合には，不必要なデリファレンスやトレイル処理の検査を行わず，直接出力変数の単一化処理を高速に行う．一方，対象となる引数の直接出力フラグがクリアされている場合には，両系統とも従来の get 系命令と同様の処理を行う．

選択点非生成節で使用される get_nocp 系命令では，選択点が生成されていないことを仮定してトレイルの記録を省略している．したがって，インデキシング対象の引数に具体化されていない変数が与えられたために，インデキシングが行えず選択点が生成された場合には，この命令で単一化処理を行うと後戻りを適切に行えなくなる．このような問題を除去するために，インデキシング用の switch 命令を拡張し，インデキシングが行えず選択点が生成される場合には，直接出力モードレジスタ全体をクリアするようにする．この拡張により，選択点が生成された場合には，選択点非生成節でも従来の get 系命令の機能が選択され，必要に応じてトレイルの記録が行われるために，後戻り処理を適切に行うことができる．

```
% get_nocp_list AN, READ_UNIFY
% get_cp_list AN, READ_UNIFY
if (DWM_REG & DWM_MASK(AN)) {
#ifdef CP_CLAUSE    /* get_cp_list の場合 */
     TRAIL(AN) ;
#endif
     *AN = TAG(LIST_TAG, H) ;
     S = H ;
     H += 2 ;
     goto WRITE_LABEL ;
}
     /* 図 2 のコードが以下に続く */
```

図 4　拡張された get 系命令の処理
Fig. 4 Enhancement for get instructions.

3.3 コンパイルドコードと実行時の処理

図１の append/3 の最初の節を例として，本最適化方式によって生成されたコードを**図５**に示す．図５の第１引数の処理に着目すると，呼び出し側で設定された直接出力フラグを（1R）で参照している．第１引数がリストの場合には，直接出力フラグはクリアされているために，通常の入力モードの処理が選択される．第１引数が変数の場合には，インデキシング命令によって事前に直接出力モードレジスタがクリアされる．この結果，get_nocp 系の命令を使用していても，すべての単一化処理は従来の方式で処理され，必要なトレイルの記録が行われる．フラグの値は，get_nocp 系命令で参照した後は必要なくなるため，自由に変更してよい．この例では，新たに第１引数に設定される値の特性にしたがって，（1S）もしくは（1C）で直接出力変数フラグの再設定を行っている．

第３引数が変数のときには，２種類の場合がある．直接出力フラグがクリアされている場合には，（3R）で従来の出力モードの処理が選択され，セットされて

```
     get_nocp_list        A1, L1    % (1R)
     write_variable       X4
     write_variable       A1
     direct_write_set     A1        % (1S)
     branch               E1
L1:  read_variable        X4
     read_variable        A1
     direct_write_clear   A1        % (1C)
E1:  get_nocp_list        A3, Lr    % (3R)
     write_value          X4
     write_variable       A3
     direct_write_set     A3        % (3S)
     branch               La
Lr:  read_value           X4
     read_variable        A3
     direct_write_clear   A3        % (3C)
     branch               append/3
```

図 5　append/3 の最適化コード
Fig. 5 Optimized code for append/3.

いる場合には，デリファレンスとトレイルの記録を省略する最適化された出力モードの処理が選択される．どちらの場合も get_nocp_list に引き続いて実行される write 系命令で直接出力変数が引数レジスタに設定されるために，(3S) で直接出力フラグがセットされる．このフラグは再帰的に呼び出された append/3 の (3R) で改めて参照され，最適化された出力モードの単一化処理が選択されるために，次回以降の第3引数の単一化処理は高速化されることになる．第3引数が定数項のときには，直接出力フラグがクリアされているので，(3R) で通常の入力モードの処理が選択されるとともに，(3C) で直接出力フラグがクリアされる．

図3の partition/4 の単一化処理では，節の特性に合わせて，1番目の節は get_cp 系，2番目の節は get_nocp 系の命令が使用されている．このために，第3，第4引数が変数の場合には，デリファレンスはともに行われないが，1番目の節ではトレイルの記録が行われ，後戻りによって2番目の節が選択される場合に備える．この場合，トレイルの記録は行われるものの，その条件判断が除去されているために，一般的なトレイル処理よりもコストが低く抑えられている．また，2番目の節は決定的に実行されるので，トレイルの記録とその条件判断が行われない．

4. 最適化の強化

引数の入力と出力に専用化された単一化コード列を用いたコンパイルドコードを生成することにより，最適化を強化する方式を示す．各引数の単一化コードがどのように専用化されているかを示すために，以下の表現法を導入する．また，各単一化コードで行われる操作を**表1**に示す．

- －直接出力変数専用コード
- -? 直接出力変数を検査する入出力両用コード
- ? 直接出力変数を検査しない入出力両用コード
- ＋入力専用コード

たとえば，append/3 で，すべての引数を入出力両用にコンパイルしたコードは (-?, -?, -?) と表現され，第3引数を出力専用にコンパイルしたコードは (-?, -?, -) と表現される．

表1 単一化コードの操作内容
Table 1 Elemental operations for unification.

単一化処理の分類	直接出力モードの検査	直接出力変数の操作	直接出力フラグの操作	入出力モードの検査
－	×	○	×	×
-?	○	○	○	○
?	×	×	○	○
＋	×	×	○	×

4.1 入力モードへの最適化

直接出力フラグの検査による最適化は，出力モードの単一化に対して効果的であるが，入力モードの単一化に対しては冗長になってしまう．したがって，ほとんどの場合に入力モードの単一化処理になると考えられる引数に対しては，直接出力フラグを検査しない通常の get 系命令を生成し，検査のオーバヘッドを除去した方が効果的である．

入力モードになる頻度が高いと考えられる引数としては，インデキシング用に選択されている引数がある．図1の append/3 では第1引数がインデキシングに用いられ，入力モードになる頻度が高いと考えられるために，(?, -?, -?) のコードを生成する．また，定数項を要求する組込み述語の引数を含む引数がある場合には，入出力モードを検査しない入力モード専用のコードを生成することにより，さらに最適化を進めることができる．図3の partition/4 では，最初の節の第1，第2引数に，組込み述語で具体化されている必要がある変数が含まれており，これらの引数は入力モードになるために，(＋, ＋, -?, -?) のコードを生成することができる．

4.2 直接出力モードへの最適化

入出力両用のコードでは，直接出力変数が引数となっている場合でも，直接出力フラグの検査が行われ，実行時のオーバヘッドが残っている．このオーバヘッドを取り除くためには，引数の入出力の組み合わせそれぞれに最適化された専用のコードを生成しておき，実行時に最適な専用コードを選択する方法が考えられる．**図6**の append/3 の例のように，専用コードと組み合わせて作成されたコードは，呼び出された最初の時点では，入出力両用の一般的なコードが実行され

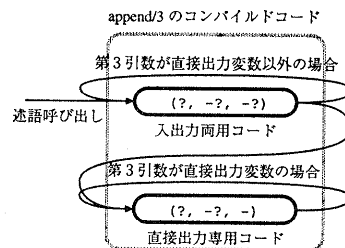


図6 直接出力専用コードの利用
Fig. 6 Optimization using specific code for direct write mode.

る．このコードでは，再帰処理を行う際に直接出力モードレジスタを検査し，第3引数に直接出力変数に対する専用コードを使用できる場合には，専用化されたコードに分岐する．この方法では，単一化処理における実行時の検査を省略できるとともに，直接出力モード専用の引数では，直接出力フラグがセットされていることが明らかなために，引数を設定する際の直接出力フラグのセットを省略できるという利点もある．

一般的に，すべての引数に対して直接出力モード専用のコードを単純に展開すると，引数の個数をNとした場合，入出力モード両用のコードだけを作成したのと比較して，基本的には 2^N 倍のコードサイズとなってしまう．このような問題は，出力モードになることが少ない引数や，処理速度に影響が少ない引数の，直接出力モード専用コードの作成を抑制することによって解決することができる．

出力モードになることが少ないと考えられる引数には，インデキシングが行われる引数や，入力引数をとる組込み述語に与えられている引数がある．また，処理速度に影響が少ないと考えられる引数には，繰り返し処理が行われる節で操作されていない引数がある．append/3 の例では，前者に相当するのが第1引数，後者に相当するのが第2引数である．したがって，図6に示したように，一般的な (?, -?, -?) のほかに，第3引数のみを直接出力モード専用にしたコード (?, -?, -) だけが追加されることになり，コードサイズは2倍弱に抑えられる．また，partition/4 の例では，前者に相当する引数として第1引数と第2引数がある．したがって，第3引数と第4引数をそれぞれ直接出力モード専用にしたコードの組合わせだけが追加されることになり，コードサイズは4倍弱に抑えられる．さらに，第3引数と第4引数が共に直接出力モードのコード (+, +, -, -) 以外を入出力両用のコード (+, +, -?, -?) で代替すれば，コードサイズは2倍弱に抑えられる．ここで，入出力両用のコードのほかに第3，第4引数がともに直接出力モードの専用コードのみを生成するのは，直接出力モードが伝播する性質を利用して，再帰呼び出しで直接出力フラグによる条件分岐を行う必要がなくなり，最適化の効果が大きいためである．

4.3 決定性のコードへの変換

partition/4 のように，節の本体の最初の部分に，組込み述語による検査とカットによる選択点の削除がある場合には，これらの特性を利用して，選択点が生成されない決定性のプログラムに変換する方式[8),9)]が提案されている．このような方式を導入することにより，選択点生成節に分類されていた1番目の節は選択点非生成節に分類され，第3，第4引数の単一化処理を決定的に行えるようになる．これにより，引数の単一化処理にトレイルの記録処理を除去した get_nocp 系の命令を使用できるようになるために，さらに高速化させることができる．

```
RXW: switch_on_list         NIL, VAR
     get_list_read          A1
     read_variable          X4
     read_variable          A1
     get_nocp_list_write    A3
     write_value            X4
     write_variable         A3
     branch                 RXW
```

図7 プログラムの停止性の分析による最適化
Fig. 7 Optimized code based on program analysis for direct write mode.

4.4 プログラムの停止性の利用

プログラムの停止性の分析により，引数の入出力情報を取得し，最適化に役立てる手法が考えられる．節の順序に依存するが，図1の append/3 の例では，第1引数と第3引数が共に変数の場合には停止しないという特性があるために，その組合せで使用されることはない．したがって，第1引数と第3引数は，片方が出力引数であれば，他方は必ず入力引数である．高速化のために出力専用のコードを作成する場合，(?, -?, -?)，(?, -?, -)，(-, -?, -?)，(-, -?, -) の4種類が考えられるが，上記の入出力の制約から，(?, -?, -) は (+, -?, -)，(-, -?, -?) は (-, -?, +) でなければならず，(-, -?, -) はあり得ないことになる．この結果，**図7**に示すように出力だけでなく入力も専用コードを生成できるようになり，処理速度をさらに向上させ，コードサイズも減少させることができる．

append/3 の (+, -?, -) のコードでは，第3引数の出力変数は2番目の節によって最終的に設定されることが明らかなので，第3引数の単一化処理で作成される直接出力変数の値を自分自身のアドレスで初期化させる必要がない．このような処理を省略することにより，さらに高速化させることができる．

5. 評　　価

5.1 最適化の効果

append/3 のコンパイルドコードに対応した処理をC言語で記述し，提案した最適化手法の性能評価を行

表 2 単一化処理の最適化の効果
Table 2 Evaluation of optimization effect.

コードの種類	HP 720 (PA-RISC 1.1)		Solbourne S 5/600 (SPARC)	
	K LIPS	Rate	K LIPS	Rate
最適化なし	966	1	520	1
モード宣言	1,414	1.46	750	1.44
大域解析法	2,909	3.01	1,651	3.18
最適化 (-?,-?,-?)	2,151	2.23	1,097	2.11
最適化 (?,-?,-?)	2,359	2.44	1,224	2.35
最適化 (?,-?,-)	2,752	2.85	1,428	2.75
最適化 (+,-?,-)	2,909	3.01	1,643	3.16
最適化 (+,-?,-) (変数の初期化なし)	3,303	3.42	1,735	3.34

表 3 1推論あたりのマシンサイクル数
Table 3 Machine cycles per inference.

コードの種類	HP 720 (PA-RISC 1.1)	Solbourne S 5/600 (SPARC)
最適化なし	51.8	63.5
モード宣言	35.4	44.0
大域解析法	17.2	20.0
最適化 (-?,-?,-?)	23.2	30.1
最適化 (?,-?,-?)	21.2	27.0
最適化 (?,-?,-)	18.2	23.1
最適化 (+,-?,-)	17.2	20.1
最適化 (+,-?,-) (変数の初期化なし)	15.2	19.1

った．すべてのコードは，引数レジスタ，スタックポインタをプロセッサのレジスタに割り当て，スタックのオーバフロー検査は行っていない．評価は PA-RISC と SPARC の2種類の RISC プロセッサを用いた2種類のワークステーション上で行った．それぞれの公称性能は 58 MIPS と 22 MIPS である．処理速度の測定結果を**表 2** に示す．最適化されたコードの評価は，上から，3章，4.1，4.2，および4.4節で述べた2方式の順で示されている．また，比較のために，提案方式を利用していないコード，モード宣言を利用したコード，大域解析法によるコードの評価を示す．なお，各コードは，拡張 WAM を基本として作成されている．

提案方式による最適化を行ったコンパイルドコードは，最適化を行っていないコードと比較して，最適化の程度により2.1倍から3.4倍の大幅な高速化が実現されている．この評価結果は，出力モードの単一化処理において，不必要なデリファレンスとトレイル処理のオーバヘッドが非常に大きなものであったことを明らかにしている．

モード宣言によって最適化されたコードに対しては最高で2.3倍，簡単な最適化を行ったコンパイルドコード (?,-?,-?) でも，1.6倍ほど高速であるという結果が得られている．この評価では，モード宣言による静的な最適化方式よりも，提案方式による動的な最適化方式の方が効果的であるという興味深い結果が得られている．この結果から，実行時の検査が増加しても，提案方式によるデリファレンスとトレイル処理の除去の方が，入出力モードの検査の除去よりも効果的であることが明らかになっている．

5.2 PROLOG 専用機との比較

評価に使用した PA-RISC と SPARC を使用したシステムのクロックは，それぞれ 50 MHz と 33 MHz である．表2に示した処理速度を基にして算出した1推論あたりのマシンサイクル数を**表 3** に示す．

本提案方式により最も最適化されたコードの1推論あたりのマシンサイクル数は，PA-RISC では 15 程度，SPARC では19程度であることが示されている．一方，専用ハードウェアを備えた PROLOG 専用機では，Tick の PROLOG マシン[10]では 22 マシンサイクル，ICOT の PSI-II[11]では 15 マシンサイクルを1推論に要しており，本最適化手法の採用によって，汎用の RISC プロセッサ上で専用のハードウェアを備えた PROLOG 専用機と同等の高い効率で処理を行えることが明らかになった．

Tick の PROLOG 専用機では，22 マシンサイクルのうち 13 マシンサイクルが，出力モードのリストの単一化処理を行う get_list 命令によって費やされている．本最適化方式を PROLOG 専用機に導入した場合，get_list 命令が2〜3マシンサイクルで実行できるようになるために，PROLOG 専用機の処理速度を2倍程度向上できるものと考えられる．

5.3 最適化の安全性

4章では，さまざまな仮定を用いてコード生成を行う方式を示したが，モード宣言を従来のように利用するのではなく，本方式で仮定を用いて最適化を行っていた部分へのヒントとして利用することにより，プログラムのセマンティックスを変えることなく高速化に役立てることができる．この方式では，宣言が誤っている場合には，処理速度が向上しないだけで安全であるという利点がある．

5.4 大域解析による最適化との比較

大域解析による最適化は，プログラムが大きくなるにつれて解析時間が爆発的に増加するという問題がある．一方，本論文で提案した最適化方式は，述語単位

でコンパイルを行えるために，従来のコンパイラと同等のコンパイル時間しかかからない．このため，大域解析による最適化手法と最適化の効果が同等でありながら，コンパイル時間を非常に短く抑えることができるという利点がある．

本方式は，プログラムの大域解析によって静的に最適化する方式と比較して，実行時の検査によるオーバヘッドが生じている．表2に示されるように，提案方式による簡単な最適化を行ったコード（?, -?, -?）は，大域解析法によるコードと比較して，19%～26% ほど処理速度が低く，この差が動的な検査によるオーバヘッドであると考えられる．しかしながら，（+, -?, -）の性能に示されるように，本提案方式でも最適化を進めることにより，同等の性能のコンパイルドコードを生成することができる．

大域解析による最適化手法では，述語の呼び出し側で最適化に適した引数が渡されていることが1カ所でも保証できない場合には，最適化できないという問題がある．一方，本方式は，実行時に最適化の検査を行うために，最適化された処理を選択できるすべての単一化処理で，最適化された処理を行うことができるという利点がある．このため，プログラムの特性によっては，大域解析による最適化と比較して処理速度が高くなる場合もある．

5.5 他の動的最適化手法との比較

本論文と同様に，動的な検査によって単一化処理を最適化する方式[12]が Beer により提案されている．この方式では，最適化対象となる変数に新たなタグを割り当てると共に，単一化処理のモードを従来の入出力の2種類から3種類に増加させることにより最適化処理を実現している．このため，専用ハードウェアを備えていない汎用プロセッサでは，最適化の検査のためのタグの切り出し操作や，タグやモードの増加による条件分岐の増加が大きなオーバヘッドを生じてしまい，最適化の効果が相殺されてしまうという問題がある．一方，本提案方式は，そのようなオーバヘッドの増加がなく，最適化の可能性がフラグの単純な検査でわかるため，汎用プロセッサで効果的に利用できる最適化方式となっている．

また，汎用プロセッサではレジスタ数が限られているため， WAM の引数レジスタのすべてを内部レジスタに割り当てることができず，その多くをメモリ上に割り付ける必要がある．Beer の方式では，メモリ上に収容されている引数のタグ検査を行う際にコストの高いメモリアクセスが生じ，最適化に要するオーバヘッドが増加するという問題がある．一方，直接出力変数レジスタを用いる本提案方式では，実用上十分な個数の引数レジスタに対して，内部レジスタ上で高速な検査を行えるという特長がある．

さらに Beer の方式では，従来の WAM 方式では処理できない新たなタグや初期化されていない変数セルを導入しており，処理系はすべての処理でこれらを正確に操作しなければ障害が発生してしまう．このため，新しい処理方式に合わせて処理系全体を再構築しなければならない．一方，本論文の提案方式でコンパイルされたコードは，呼び出す前に直接出力モードレジスタ全体をクリアすることにより従来方式の単一化処理を行わせることができる．また，従来方式でコンパイルされたコードやインタプリタが，直接出力モードレジスタを無視して従来方式の処理を行っても何ら問題がない．このため本最適化手法は，すでに構築されている処理系に少ない労力で容易に導入できるという特長がある．

6. まとめ

本論文では，出力モードの単一化処理に着目し，新たな変数分類情報を述語呼び出しで受け渡すことにより，宣言や大域的なプログラム解析を必要としない最適化方式を提案した．本最適化方式は，自己再帰的なプログラムを最適化できるだけでなく，複数の述語が相互に呼び出しあっているプログラムでも，述語単位の最適化で大域的な最適化を行えるという特長がある．また，非常に簡単に導入でき，セマンティックスに影響を与えない安全な最適化方式であるだけでなく，インクリメンタルコンパイルやインタプリタにも適用できる方式であるために，今後の PROLOG 処理系の基本的な最適化方式として用いられるものと考えられる．

本論文では，プログラムの停止性を利用した最適化の強化法について簡単に示したが，このようなプログラムの特性の分析法，最適化への応用法，適用範囲について，さらに研究を進めることが今後の課題としてあげられる．

参考文献

1) 硴崎賢一：引数の出力モード伝搬による PROLOG の最適化方式，情報処理学会記号処理研究会資料，SYM-66-2 (1992).
2) Warren, D. H. D.: An Abstract Prolog Instruction Set, Technical Note 309, SRI Interna-

tional (1983).

3) Meier, M.: Compilation of Compound Terms in Prolog, *Logic Programming: Proceedings of the 1990 North American Conference*, pp. 63-79, The MIT Press (1990).

4) Komatsu, H., Tamura, N., Asakawa, Y. and Kurokawa, T.: An Optimizing Prolog Compiler, *Proceedings of the Logic Progamming Conference '86*, pp. 143-149, ICOT (1986).

5) 新井　進, 岸本光弘, 久門耕一, 服部　彰: Prolog コンパイラの設計, 第1回人工知能学会全国大会講演論文集, pp. 185-188 (1987).

6) Taylor, A.: Removal of Dereferencing and Trailing in Prolog Compilation, *Logic Programming: Proceedings of the Sitxth International Conference*, pp. 48-60, The MIT Press (1989).

7) Roy, P. V.: The Benefits of Global Dataflow Analysis for an Optimizing Prolog Compiler, *Logic Programming: Proceedings of the 1990 North American Conference*, pp. 501-515, The MIT Press (1990).

8) 硴崎賢一, 松本一夫, 上原邦昭, 豊田順一: PROLOG 処理系アーキテクチャの拡張と最適化方式の提案, *Proceedings of the Logic Programming Conference '88*, pp. 151-160, ICOT (1988).

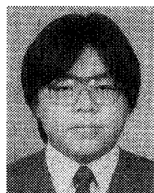
9) 阿部重夫, 川端　薫, 黒沢憲一: Prolog の最適化方式, 情報処理学会論文誌, Vol. 30, No. 5, pp. 587-595 (1989).

10) Tick, E. and Warren, D. H. D.: Towards a Pipelined Prolog Processor, *New Generation Computing*, Vol. 2, pp. 323-345, Springer-Verlag (1984).

11) Nakashima, H. and Nakajima, K.: Hardware Architecture of the Sequential Inference Machine: PSI-II, *Proceedings of the Symposium on Logic Programming*, pp. 104-113, IEEE (1987).

12) Beer, J.: The Occur-Check Problem Revisited, *The Journal of Logic Programming*, Vol. 5, No. 3, pp. 243-261 (1988).

(平成4年5月27日受付)
(平成5年7月 8 日採録)



硴崎　賢一（正会員）

1960 年生．1985 年大阪大学工学部造船工学科卒業．1987 年同大学院基礎工学研究科情報分野博士前期課程修了．1988 年同大学院博士後期課程退学．同年九州工業大学情報工学部電子情報工学教室助手．1993 年同大学助教授．工学博士．記号処理言語，オブジェクト指向言語の実現方式と応用，グループによるソフトウェア開発支援環境，マルチメディアシステム，文書処理システムなどの研究に従事している．電子情報通信学会，人工知能学会各会員．