記 号 処 理 66－2
（1 9 9 2. 9. 2 2）

引数の出力モード伝播による PROLOG の最適化方式

碓崎 賢一
九州工業大学 情報工学部

内容梗概

引数の出力モードの伝播を利用した，単純で効果的な PROLOG の最適化方式を提案する．本方式では，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，基本的な 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，3.3M LIPS の高い性能が得られることが明らかになった．

Optimization Method for PROLOG Argument Unification Using Write Mode Propagation

Ken'ichi KAKIZAKI

Faculty of Computer Science and Systems Engineering
Kyushu Institute of Technology
680-4, Kawazu, Iizuka, Fukuoka 820, Japan

Abstract

This paper describes a simple and efficient PROLOG optimizing method using argument write mode propagation. This method introduces argument classification called direct write mode variable, and passes the information about values put into argument registers from caller to callee. According to the information, this 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 using this optimizing method shows an improvement of 3.4 times faster than that of using conventional method, and it gains 3.3M 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実行速度向上の阻害要因となっている．このような処理コストは，決定性処理においては一般的に冗長なだけであり，記述能力が高いにも関わらず PROLOG が使用されにくい原因となっている．PROLOG が特殊な人工知能言語としてではなく汎用言語として広く利用されるためには，PROLOG に特有な高度な機能を必要としない一般的な決定性処理において，不必要で冗長な処理を除去することにより，プログラムを高速に実行できるようにしなければならない．

決定性処理では，単一化処理のデリファレンスやトレイル処理などが処理コストの大きな割合を占めている．本報告では，これらの処理に着目し，単一化処理の出力モードにおける冗長な処理を除去することによって，決定性処理を高速化させる方式を提案する．デリファレンスとトレイル処理に着目した最適化手法 [Kom86] [新井 87] [Tay89] [Roy90] はいくつか提案されているが，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，直接出力フラグと呼ぶ引数に関する情報を述語間で自動的に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，モード宣言や大域的なプログラム解析を行うことなく，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 2.1 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完全な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 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比較的高いために，単一化処理のコストが大きくなっている．

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．WAM [War83] 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．出力モードの単一化処理が行われる append/3 の第 3 引数には，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いる．したがって，デリファレンスが必要ないにも関わらず，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．

```
:- 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

```
% 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

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の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ないために，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な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，4 クロックサイクル程度で行な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関わらずその数倍の処理時間を要し，他の WAM 命令も含めた append/3 の 1 推論時間の中でも大きな比重を占めている．このため，単一化処理からこれらの処理を除去することにより，PROLOG の処理速度を向上することができる．

2.2 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構造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．この様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

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

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手法として，使用者の宣言による手法 [Kom86] [新井 87] と，プログラムの大域的な解析による手法 [Tay89] [Roy90] が提案されている．宣言を用いた最適化手法は，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大幅に長くな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．

本報告では，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．

- 宣言を必要とせず，自動的で安全な最適化
- 大域的なプログラム解析を必要としない最適化
- 述語間の大域的な最適化
- 実行時の検査による動的な最適化

従来の述語単位の最適化手法では，単一化処理の最適化に利用されるのは，述語の頭部の引数の情報のみである．一方，提案方式では，呼び出し側が設定した引数に対する情報も有効に利用するという特長がある．この様な最適化は，従来は大域的なプログラム解析を行う最適化方式のみで実現されているため，本方式は述語単位の最適化でありながら，大域的な最適化を行っているのと同等であると考えられる．

3 単一化処理の最適化

3.1 直接出力フラグとレジスタ

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．引数に関する情報は，本質的には引数レジスタに設定された値の解析により得ることができるが，その解析コストが高いために，単一化の処理速度が低下するという問題がある．このような引数の解析コストを低減する目的で，WAM 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フラグの集合となっており，各フラグを直接出力フラグと呼ぶ．語長が 32 ビットのマシンでは，実用上十分な 32 引数に対応したフラグを確保できる．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．レジスタ割当の最適化などで操作されていない引数に対応した直接出力フラグは変更を受けない．上記の条件に合致しない引数レジスタに対応した直接出力フラグはクリアされる．

直接出力フラグは，WAM の入出力モードフラグに対応させることができる．入出力モードフラグが，各引数ごとの単一化処理の中だけで伝達されるのに対して，直接出力フラグは，述語間を越えて伝達される点と，そのために，引数単位で複数のフラグが導入されている点が異なっている．

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．選択点が作成される際には，その時点における直接出力モードレジスタの内容を保存し，後戻り処理では，選択点に記録されている直接出力モードレジスタの内容を復帰させる．

3.2 命令の追加と拡張

直接出力モードレジスタを使用した最適化手法のための WAM 命令への追加と拡張を示す．基本的には，文献 [Mei90] に示されている拡張 WAM 命令セットをさらに拡張したものになっている．

直接出力フラグを利用して出力モードの単一化処理を効率良く実行するために，get 系命令を拡張した get_nocp と get_cp の2種類の系統を追加する．両系統とも，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，従来の get 系命令と同様の処理を行うが，直接出力フラグがセットされている場合には，それぞれ以下のような処理を行う．

- get_nocp

  デリファレンスとトレイル処理をしない

- get_cp

  デリファレンスをせずトレイルを記録する

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 3 に示す．図 3 の DWM は直接出力モードレジスタ，DWF(AN) は引数レジスタ AN に対応した直接出力フラグを示す．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，unify 系の命令や文献 [Mei90] に示されている write 系の命令と同様に不必要なデリファレンスやトレイル処理を行なわず，述語呼び出しを越えて作成される一組のリストや複合項を冗長な操作を行うことなく作成する．これは，頭部の単一化処理で，複合項やリストの入れ子が深く unify 命令で単一化処理が行われる場合には，一度出力モードになった後は構造やリストを作成する操作でデリファレンスやトレイル処理が必要なくなるのと同様である．これらの系統の命令を使用することにより，直接出力変数の単一化処理では，1回のメモリ参照と4回の条件判断を1回の条件判断に置き換え高速化することができる．

引数レジスタに設定する値が直接出力変数か否かを示す直接出力フラグの設定命令を新たに導入する．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．

インデキシング用の switch 系の命令は，インデキシングを行う引数が変数で，インデキシングが行えなかった場合に直接出力モードレジスタをクリアするように拡張する．

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．

```
% get_nocp_list AN, READ_UNIFY
% get_cp_list AN, READ_UNIFY
if (DWM & DWF(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 3: 拡張された get 系命令の処理

また，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．

3.3 節の分類とコード生成

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 4 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．

```
:- 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 4: 節の分類

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

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．また，呼び出し側で引数レジスタに直接出力変数を設定する場合には direct_write_set 命令を使用して引数に対応する直接出力フラグをセットし，それ以外の値を設定する場合には direct_write_clear 命令を使用して対応する直接出力フラグをクリアする．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．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．

```
     get_nocp_list          A1, L1
     write_variable         X4
     write_variable         A1
     direct_write_set       A1
     branch                 E1
L1:  read_variable          X4
     read_variable          A1
     direct_write_clear     A1
E1:  get_nocp_list          A3, Lr
     write_value            X4
     write_variable         A3
     direct_write_set       A3
     branch                 La
Lr:  read_value             X4
     read_variable          A3
     direct_write_clear     A3
     branch                 append/3
```

図 5: append/3 の最適化コード

3.4 実行時の処理

図 5 の append/3 の例で，第 1 引数がリストの場合には，インデキシングにより最初の節が決定的に選択される．この場合，第 1 引数の直接出力フラグはクリアされているので，通常の入力モードの処理が選択され実行される．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，従来の出力モードの処理が選択され実行される．直接出力フラグがセットされている場合には，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実行される．どちらの場合も直接出力フラグがセットされるために，次回の第 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，通常の入力モードの処理が選択され実行される．

第 1 引数が変数の場合には，節の分類に関わらず選択点が作成されるため，後戻り処理に対処するためにインデキシング命令によって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るため，セマンティックスが変化するなどの問題は生じない．

図 4 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なわれ，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．

- \- 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．

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，検査のオーバー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 4 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，実行時のオーバーヘッドが残っている．このオーバーヘッドを取り除くためには，モード宣言で指定された入出力モードに最適化されたコードを生成するのと同様に，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 に示される 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，全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．この様な問題は，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．

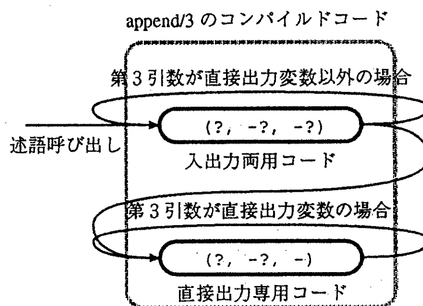

図 6: 直接出力専用コードの利用

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 (?, -?, -?) の他に，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倍程度に抑えられる．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4倍程度に抑えられる．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倍程度に抑えられる．入出力両用のコードの他に第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．上記の2例の出力モード専用にしたコードでは，汎用のコードと比較して単一化処理のコードサイズが減少しているので，実際のコードの増加は，1.5 ～ 2倍の間にとどまると考えられる．

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 [硴崎 88] [阿部 89] 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うことができるようになる．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．

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，その組み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．

```
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

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 単一化処理の最適化

WAM の処理方式に基づき，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った．全てのコードは，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，スタックのオーバーフロー検査は行なっていない．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 1 に示す．最適化されたコードの評価は，上から，3 章，4.1 節，4.2 節，および 4.4 節で述べた 2 方式の順で示されている．

表 1: 単一化処理の最適化の効果

	HP 720 (PA-RISC 1.1)		Solbourne S5/600 (SPARC)	
	K LIPS	Rate	K LIPS	Rate
WAM	966	1	520	1
モード宣言	1505	1.56	782	1.50
最適化 (-?, -?, -?)	2151	2.23	1059	2.04
最適化 (?, -?, -?)	2359	2.44	1068	2.05
最適化 (?, -?, -)	2752	2.85	1428	2.75
最適化 (+, -?, -)	2909	3.01	1498	2.89
最適化 (+, -?, -) (自己参照なし)	3300	3.42	1735	3.34

本方式により最適化を行ったコンパイルドコードは，WAM の基本的なコンパイルドコードと比較して，3.4 倍程度の大幅な高速化が実現されている．また，モード宣言によって最適化されたコードに対しても 2.2 倍ほど高速であるという結果が得られている．さらに，基本的で簡単な最適化を行ったコンパイルドコード (?, -?, -?) でも，セマンティックスが変らないという利点があるにも関わらず，モード宣言によって最適化されたコードに対して 1.6 倍ほど高速であるという結果が得られている．これらの評価結果は，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が処理時間に占める割合が非常に大きかったことを明らかにしている．

引数の入出力モードの組み合わせ別に最適化されたコードを生成して処理速度を向上させる方式や，引数の入出力特性を考慮してコードサイズの増加を抑制する方式を 4 章で示したが，これらは目的に応じて選択することができる．多くの LISP 処理系では，処理速度，コードサイズ，安全性などに着目して，それぞれ

の重要度を指定してコンパイルできるようになっている．本最適化方式を用いたPROLOG コンパイラも，このような指定により用途に最も適した特性を持つコンパイルドコードを得られる．

4 章では，様々な仮定を用いてコード生成を行っているが，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いた部分へのヒントとして利用することにより，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．

5.2 直接出力モードレジスタ

WAM 方式では，入出力モードにより異なる単一化処理を選択するために，入出力モードフラグを設けている．本方式では，従来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用いれば，余分な資源を消費することなく導入することができるという利点がある．

選択点が作成される場合には，直接出力モードレジスタは選択点に格納されるが，選択点への保存と復帰では，それぞれ1回のメモリ操作が増えるだけである．一方，この操作によって得られる効果は，直接出力変数の1回の単一化処理において，1回のメモリ参照と2回の条件分岐を除去することである．したがって，一般的な状態では，選択点の操作の増加よりも，最適化による操作の減少のほうが大きくなるものと考えられる．

提案方式とは逆に，引数に定数項が与えられていることを示すフラグを利用する最適化も考えられる．しかしながら，引数が定数項の場合には，従来の方式でもデリファレンスが不必要な場合には1回の条件判断だけで処理が行われているために，検査のコストは同等であり，逆にフラグの設定のコストが増加するという問題がある．したがって，引数が定数項であることを示すフラグは採用していない．

6 まとめ

本報告では，単一化の出力処理に着目し，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手法を提案した．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，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上に，非常に簡単に実現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手法として用いられるものと考えられる．

本報告では，提案方式の特性を基本的な1つの述語の評価で示したが，さまざまな特性を持つプログラムを対象にして，本最適化方式の効果と特性をさらに明かにすることが今後の課題として考えられる．

参考文献

[Kom86] Komatsu, H., Tamura, N., Asakawa, Y. and Kurokawa, T.: "An Optimizing Prolog Compiler", Proceedings of the Logic Programming Conference '86, pp. 143–149ICOT (1986).

[Mei90] Meier, M.: "Compilation of Compound Terms in Prolog", Logic Programming: Proceedings of the 1990 North American Conference, pp. 63–79, The MIT Press (1990).

[Roy90] 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).

[Tay89] Taylor, A.: "Removal of Dereferencing and Trailing in Prolog Compilation", Logic Programming: Proceedings of the Sixth International Conference, pp. 48–60, The MIT Press (1989).

[War83] Warren, D. H. D.: "An Abstract Prolog Instruction Set", Technical Note 309, SRI International (1983).

[阿部 89] 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).

[新井 87] 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 人工知能学会 (1987).

[硴崎 88] 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. 151–160ICOT (1988).