宣言的プログラムのプログラム変換ルールの自動生成法

3U－9

瀧内邦弘　赤間清　宮本衛市

北海道大学工学部

1　はじめに

知識処理の分野では複雑な問題解決を効率良く行うための計算モデルが必要である．この要求に応えるひとつの考え方として宣言型計算モデルがある．それは，宣言的プログラムのプログラム変換を計算とみなす計算モデルである．宣言的プログラムは確定節の集合であり，通常の論理プログラムより強力な表現力を持つ．

この計算モデルに基づいて提案された計算の枠組がルールを基礎としたプログラム変換 [1]（Rule Based Program Transformation:RBPT）である．RBPT では，複数のルールを繰り返し適用してプログラムを等価変換する．したがって，プログラム変換の効率はルールを如何に用いるかに依るところが大きい．

計算を高速化するために，本論文では RBPT のためのルールの自動生成法を与える．ここでのルールの自動生成は，宣言的プログラムから正当なプログラム変換ルールを自動的に生成するものである．また本論文では，意味解釈システムを例題として自動生成されたルールを実際に適用し，高速化の効果を検証する．

以下本論文でプログラムという場合は宣言的プログラムを指し，プログラム変換という場合は宣言的プログラムのプログラム変換を指す．

2　プログラムからのルール生成

プログラム変換を行う際には複数のルールを用いる．ルールはプログラムから別のプログラムに変換する手順の記述である．本章では宣言的プログラムからルールを生成する方法を述べる．

2.1　方法1

1回の Unfold 変換を基礎とすればプログラムからルールが生成できる．

append のプログラムを例にとって説明する．append は第1引数がニルの場合とコンスの場合で場合分けされる3引数のプログラムである．第1引数がニルの場合は第2，第3引数を単一化するルールを作ればよく，コンスの場合は再帰を用いてリストの連結を実現するルールを作る．このようにプログラムからルールを生成することができる．

2.2　方法2

1回の Unfold 変換を基礎とした方法1では，もとのプログラムの記述に近いルールしか得られない．そこでプログラムを充分に等価変換（複数回の Unfold 変換やそれ以外の等価変換）してから方法1を適用するのが方法2である．

これにより，方法1では得ることのできないルールを生成できる．

3　ルール生成の例

本章では，方法2に基づくルール生成の例を述べる．最後に生成されたルールを意味解釈システムに適用してその効果を示す．

なお本論文で述べる意味解釈システムは対象領域を将棋の世界に限定したシステム [2] である．このシステムでは将棋のルール，その時点での盤面の状況などの知識をプログラムとして表現し，プログラム変換を用いて意味解釈を行う．

3.1　変換前の節

上記のプログラムは次の節を含む．

```
(動かせる *O *G)
    ← (実在する *O),
      (盤面上 *G),
      (ルール動ける *O *G),
      (通り道に駒がない *O *G),
      (手 *O *T),
```

A Method of Automatic Rule Generation for Program Tansformation of Declarative Programs
Kunihiro TAKIUCHI, Kiyoshi AKAMA, Eiichi MIYAMOTO.
Faculty of Eng., Hokkaido Univ.
Kita 13, Nishi 8, Sapporo 060, Japan.

	処理時間 (秒)	変換試行回数 (回)	ルール適用数 (個)
生成ルールなし	11.70	176	79
生成ルールあり	2.78	52	28

表 1:「先手の歩を７－６に動かせ」の意味解釈の効率比較

```
    (自分の駒がない *G *T).
```

これは,駒*O が移動先*G へ移動できる条件を将棋の規則に照らし合わせて節として表現している.歩に関するルールを得るために,まず,この節の駒を歩に特殊化する.それには*O に情報 (kind . fu) を付加して以下のようにすればよい.

```
(動かせる *O~((kind . fu)) *G)
    ← (実在する *O~((kind . fu))),
       (盤面上 *G),
       (ルール動ける *O~((kind . fu)) *G),
       (通り道に駒がない *O~((kind . fu)) *G),
       (手 *O~((kind . fu)) *T),
       (自分の駒がない *G *T).
```

この節を含むプログラムをプログラム変換して新しいプログラムを作る.その結果得られた節から方法 1 によりルールを生成する.

3.2 生成されたルールの例

変換前の節をプログラム変換により,節の増加を禁止する条件下で,計算可能なところまでプログラム変換する.この例の場合,約 50 個のルールが適用されてプログラム変換が停止する.プログラム変換の結果得られた変換後の節から次のルールを生成することができる.

```
(Rule1  動かせる_歩
  (Head (動かせる *O~((kind . fu)) *G))
  (Cond (true))
  (Body (盤面上 *G)
        (MEMBER *O~((kind . fu)) *B)
        (盤面 *B)
        (tr-player *T *N)
        (MEMBER *N (1 -1))
        (arith minus *Ct *Gt *N)
        (unify *O~((kind . fu))
               ?~((place . *C~((yoko . *Gy)
                               (tate . *Ct)))
                  (player . *T)
                  (kind . fu)))
        (unify *G ?~((yoko . *Gy)
                     (tate . *Gt)))))
```

このルールは,歩を所定の場所*G へ動かすには,その時の盤面の歩の位置*C の横座標*Ct は変えずに,縦座標を１変化 (*Cy-*Gt=1) させればよいことを述べている.

3.3 効果と考察

もとのプログラムから方法 2 により生成されたルールは,歩の場合の特殊なルールであり,RBPT のモジュラー性より,もとからあるルール集合に加えられる.

本論文では,「先手の歩を７－６に動かせ」という文と盤面の情報から意味解釈を行ない,それを実行する例題を取り上げる.

3.2 節で生成したルールをこの例題に適用することにより,約 50 のルール適用数の短縮化が期待できる.実際に,生成されたルールを適用したことで,時間にして,約 4 倍の処理の高速化が達成できた (表１) .

4 おわりに

本論文では,プログラム変換の結果得られた節からルールを生成し,そのルールがプログラム変換による計算の高速化に有効であることを示した.ルール生成により新しいルールが作られ,ルールが増加する.このため適用ルールの探索効率の悪化が予想される.この問題は,コンパイラ [3] により解消できる.

参考文献

[1] 赤間 清:ルールを基礎としたプログラム変換,「自然言語処理における実働」シンポジウム論文集,pp.86-94,1994.

[2] 野村 祐士, 赤間 清, 宮本 衛市:プログラム変換による意味の解釈, 情報処理学会, 自然言語処理研究会,Vol.93,No1,pp.87-94,1993.

[3] 中尾 公俊, 赤間 清, 宮本 衛市:プログラム変換に基づく計算システムのためのコンパイラと仮想マシンの作成, 情報処理学会第 49 回全国大会,1995.