2D-5 自動化を考慮したループ結合によるプログラム変換システムの作成

元木 誠　永田 守男　(慶応義塾大学理工学部)

1．はじめに

プログラムのソース・レベルでの変換方法の一つとしてUnfold/Fold変換がある1)。この手法をもとにプログラムのループ結合を行なうことで効率の良いプログラムを変換生成する研究がいくつかなされているが2)3)、この過程を自動化している例は少ない。本研究では、対象をPrologによるリスト処理プログラムとした上で、リスト処理述語において各々の間で成り立っている等価な関係についての補助知識をUnfold/Fold変換規則に加えることで、これらの述語に関連したループ結合をほぼ自動的に行なうプログラム変換システムを作成した。

2．変換の概要

本システムでは、変換規則としてUnfold規則，Fold規則，ゴール移動規則，補助知識適用規則の４つを用いている。変換の手順は以下の通りである。

ここでは、例としてリストxの最後の要素をyに置きかえるプログラム

```
change_last(X,Y,Z) :-
        reverse(X,[W|U]),
        reverse([Y|U],Z).
```

(Zは出力)

の変換を行なってみる。

1) 定義された節のbodyに現れる書くアトムに対して、Unfold規則を適用する。

Unfold規則とは、A:-A1,A2,A3とA2:-B1,B2,B3から (A:-A1,B1,B2,B3,A3)θを得る変換規則のことである。このときθ、両節のA2を単一化する操作を示している。

例では

```
reverse([],[]).
reverse([A|X],Y):-
reverse(X,Z),append(X,[A],Y).
```

とchange_lastのbody中のreverseでUnfoldを行なって次のようになる。

```
change_last([X],Y,[Y]).                    (*)
change_last([X|Y],U,Z):-
 reverse(Y,[W1|W2]),append(W2,[X],V),      (**)
 reverse(V,V1),append(V1,[U],Z).
```

2) Unfold規則の適用と同時に補助知識の適用が可能となる時には、優先的にその適用を行なう。例では、(**)の下線部に上記の知識が適用できて、この節は、

```
change_last([X|Y],U,Z):-
 reverse(Y,[W1|W2]),reverse(W2,W3),
 reverse([X],X1),append(X1,W3,V1),
 append(V1,[U],Z).
```

となり、さらにUnfold規則の適用で、Fold規則の適用できる

```
change_last([X|Y],U,[X|Z]):-
 reverse(Y,[W1|W2]),reverse(W1,W2),
 append([X|W2],[U],Z).
```

となる。

3) Fold規則が適用可能となると、Fold規則を適用できる限り適用して変換を終了する。

Fold規則とは、A:-A1,A2,A3とB:-A2,A3から(A:-A1,B)θを得る変換規則のことである。ここでθは、両節のA2,A3を単一化する操作を示している。

この規則の適用によって定義されたプログラム節の再帰的な定義が完了することになる。この時、Foldできるようにゴールの移動という操作を行なう。ここでは、その節中に同じ引数の前にくるようにするという方法をとっている。

例は、Fold規則の適用によって

```
change_last([X|Y],U,[X|Z]):-
 reverse(Y,[W|W1]),reverse([U|W1],Z).
```

となる。さらにFold規則の適用によって

```
change_last([X|Y],U,[X|Z]):-
 change_last(Y,U,Z)                        (***)
```

が得られることになる。

結果として、最初に定義されたchange_lastは、reverseの２重ループが(*),(***)のchange_lastによる１重のループに結合されて、効率が良くなる。

3．補助知識の構成

前述のように補助知識とは、述語の組合せの間で成り立っている等価関係を表現したものである。この知識の適用によってUnfold規則やゴール移動規則では不可能な変換をプログラムの意味を変えずに実行できる。すなわち、ゴールの移動やあるいはappendの結合律などの規則の適用だけではうまくFold規則が適用できるところまで変換が進まない時に、変換を促す規則である。しかし、これらの知識は、あらかじめ用意されたものにし

Implementing a program transformation system,which is aimed at automation,taking into consideration the loop structures

Makoto MOTOKI and Morio NAGATA
KEIO Univ. Faculty of Science and Technology

か用いることができないためどんな状況においても有効というわけではなく、また場当たり的なアイデアの寄せ集めとなる可能性がある。

そこで本システムでは、対象範囲を基本的なリスト処理述語(ここでは、append,reverse,length,delete,member,divide（リストの分割）,substitute（要素の置き換え）,insert,flat）に限定して、補助知識の有効範囲を明確にするとともに、その中の述語間の関係についてもいくつかのまとめ上げを行なうことで簡単な体系化を試みた。

具体的には、各述語間で成り立つ有用な等価関係のほとんどすべてが、append,reverse,lengthと他の述語との間の関係に限られていることから、等価関係を３つに大別して表現することにした。

・append, P ⟷ P, P, append
P={reverse((1,2,3,3,4),(1,5,2,6,6,5,4))以下、引数の照合部分は省略、delete,substitute,insert,flat}
※Pにはこれらの述語が入りそのときの引数の決め方は照合部分を参照する。
・reverse, P ⟷ P, reverse
P={length,delete,member,substitute,insert}
・P, length ⟷ length, length, add
P={append, insert}

４．システムの実現と評価

C-Prolog上で、システムを作り、以下に示す実験を行なってみた。

```
flat_length(X,Y) :- flat(X,Z),length(Z,Y).
flat([],[]).
flat(X,[X]) :- atomic(X).
flat([A|X],Y) :- flat(A,U),flat(X,V),
                 append(U,V,Y).
length([],0).
length([A|X],N) :- length(X,N1),add1(N1,N).
```

⇩

```
flat_length([],0).
flat_length(X,1).
flat_length([X|Y],Z) :- flat_length(X,U),
                        flat_length(Y,V),
                        add(U,V,Z).
```

flatとlengthの操作を結合したものだが、効率は２倍ほど良くなった。
append,length↔length,length,add
の補助知識を用いている。

```
flat_reverse(X,Y) :- flat(X,Z),reverse(Z,Y).
```

⇩

```
flat_reverse([],[]).
flat_reverse(X,[X]) :- atomic(X).
flat_reverse([X|Y],Z) :- flat_reverse(X,U),
                         fr(Y,Z,U).
fr([],X,X).
fr(X,[X|Y],Y) :- atomic(X).
fr([X|Y],Z,U) :- fr(X,V,U),fr(Y,Z,V).
```

flatとreverseの結合を行なっている。効率は６倍ほど上っている。
append,length↔reverse,reverse,append
の補助知識を用いている。

５．検討

変換に補助知識を導入することによってどのように改良がなされ、どのような問題点が残っているかを検討してみる。

1) 変換の過程で補助知識の適用規則とまた別の（Fold規則以外の）規則が適用可能となりうる時には必ず補助知識適用を優先させることにしているため、無駄な変換規則の選択が減り、変換の方向づけが容易にできるようになった。したがって、より変換の自動化が容易になったといえる。
2) Unfold/Fold規則とゴールの移動規則だけではうまく変換できないような複雑なプログラムの変換も補助知識の有効範囲内では可能となった。
3) 補助知識が不足している場合でも、現存のある程度体系化された知識からどのような等価関係が必要となるかが比較的予想がつけ易い。
4) 有効範囲は明言してあるものの、知識としてどれだけ持っていれば十分なのかがあまり明確ではない。
5) 現存のままでは、まだ体系化が不十分なので、補助知識の有効範囲を大幅に広げると破綻を来たすことになる。

６．おわりに

Unfold/Fold変換では、変換の過程において非決定的な部分が非常に多いので、ここで取り上げたような補助知識の導入は不可欠と考えられる。しかしその際、各等価関係の適切な分類とさらには、その分類された知識を利用する場合に特有の変換パターンについての知識等も必要となってくる。

参考文献

1) Burstall, R. M. and Darlington, J.: A Transformation System for Developing Recursive Programs, JACM, Vol.24, No.1, pp.44-67(1977)
2) Darlington, J.: An Experimental Programs Transformation and Synthesis System, Artificial Intelligence 16, pp.1-46(1981)
3) 佐藤,玉木: Appendオプティマイザについて, ICOT, The Logic Programming Conference'84 9-1(1984)