1N-1

Prologのプログラム変換について

—— 補助的な知識の取扱いについて ——

松方 純・田中英彦・元岡達
(東京大学 工学部)

0. はじめに

Prologのプログラム変換は，人間が理解しやすいPrologプログラムから，実行効率の高いPrologプログラムを導出するための一つの手法であるとともに，Prologプログラムの等価性を示す手段としても有用な方法である。[1,2,3]

Unfold／Fold式のプログラム変換[1,3]では，しばしば，補助的な知識が使用される。たとえば，reverseあるいはpalindromeのプログラムのdata structure mappingによる変換[2,3付録]では，appendの結合法則という補助的な知識が適用される。

本論文では，appendの結合法則を，Horn clauseで記述されたプログラム同士の等価性という形で表現することを導入する。そして，appendの結合法則をUnfold/Fold式のプログラム変換によって証明できることを示す。最後に，appendの結合法則の適用方法について論じる。

1. プログラムの等価性によるappendの結合法則の表現

一般に，

```
A1,A2,...,Am <-> B1,B2,...,Bn                                    (1)
```

の形をしている命題は，命題の同値性 ⟷ を，プログラムの等価性に写すことにより，

「プログラムPの二種類の定義(2)と(3)は等価である。」

という形で表現できる。ただし：

```
P <- A1,A2,...,Am                                                (2)
P <- B1,B2,...,Bn                                                (3)
```

このことを，appendの結合法則

```
append(X,Y,U),append(U,Z,V) <-> append(X,W,V),append(Y,Z,W)       (4)
```

に適用すると，appendの結合法則は，次のように表現できる。

「append3の二つの定義(5)と(6)は，等価である。」

ただし：

```
append3(X,Y,Z,V) <- append(X,Y,U),append(U,Z,V)                   (5)
append3(X,Y,Z,V) <- append(X,W,V),append(Y,Z,W)                   (6)
```

このような表現法の導入により，appendの結合法則も，Prologのプログラム変換の世界の中で取扱うことができるようになる。

2. appendの結合法則の証明

appendの結合法則は，append3の二つの定義(5)と(6)に対し，Unfold／Fold式のプログラム変換を施すことにより，どちらの定義も，次のような定義{(7),(8),(9)}に等価変換できることから証明される。

```
append3([],[],Z,Z) <-                                            (7)
append3([],[U|Y],Z,[U|V]) <- append3([],Y,Z,V)                   (8)
append3([U|X],Y,Z,[U|V]) <- append3(X,Y,Z,V)                     (9)
```

3. append の結合法則の適用

プログラム変換において，appendの結合法則は，(4)の一方の辺をもう一方の辺に置き換える変換規則として使用される。この変換は，(5)[(6)]とのfoldを行なった直後に，foldの結果として出現した，append 3 を(6)[(5)]によって unfold を行なうという過程とみることもできる。

4. 結論

appendの結合法則に関する以上の議論は，

A1,A2,...,Am <-> B1,B2,...,Bn

の形をしている命題一般に適用できると考えられる。

Unfold／Fold式のプログラム変換において，補助的な知識の取扱いは，重大な問題である。補助的な知識は，一般にHorn clauseで表現できるとはかぎらないので，その取扱いは，Unfold／Fold式のプログラム変換の外で行なわざるを得ない。しかしながら，本論文で提示したような方法を用いることにより，補助的な知識の一部を，Unfold／Fold式の枠内で取扱うことができる。

参考文献

[1] R.M. Burstall, J. Darlington: A Transformation System for Developing Recursive Programs, Journal of the ACM, Jan. 1977.

[2] Å. Hansson, S-Å Tärnlund: Program Transformation by Data Structure Mapping, in Logic Programming, Academic Press, 1982.

[3] 佐藤，玉木：Prologに於けるプログラム変換，ロジック・プログラミング・コンファレンス'83.

付録

reverseのプログラムの data structure mapping

```
1.      reverse(Ø,Ø).
2.      reverse(X.Y,Z) <- append(U,X.Ø,Z),reverse(Y,U).

3.      append(Ø,U,U).
4.      append(X.Y,Z,X.W) <- append(Y,Z,W).

Definition
5.      reverse2(X1,X2,Y1,Y2) <- reverse(X,Y),append(X,X2,X1),append(Y,Y2,Y1).
Note: this definition comes from
        m(X,(Y Z)) <- append(X,Z,Y).          ... mapping function
        reverse2(X',Y') <- reverse(X,Y),m(X,X'),m(Y,Y').

Unfold 5
6.      reverse2(X1,X1,Y1,Y1).
7.      reverse2(X1,X2,Y1,Y2) <- append(U,X.Ø,Z),reverse(Y,U),append(X.Y,X2,X1),append(Z,Y2,Y1).
Unfold 7
8.      reverse2(X.X1,X2,Y1,Y2) <- append(U,X.Ø,Z),reverse(Y,U),append(Y,X2,X1),append(Z,Y2,Y1).

Apply a rule:
        append(X,Y,Z),append(Z,U,V) <-> append(X,W,V),append(Y,U,W)
9.      reverse2(X.X1,X2,Y1,Y2) <- reverse(Y,U),append(Y,X2,X1),append(U,W,Y1),append(X.Ø,Y2,W).

Unfold 9
1Ø.     reverse2(X.X1,X2,Y1,Y2) <- reverse(Y,U),append(Y,X2,X1),append(U,X.W,Y1),append(Ø,Y2,W).
Unfold 1Ø
11.     reverse2(X.X1,X2,Y1,Y2) <- reverse(Y,U),append(Y,X2,X1),append(U,X.Y2,Y1).
Fold 11 with 5
12.     reverse2(X.X1,X2,Y1,Y2) <- reverse2(X1,X2,Y1,X.Y2).
```