4X-7

一般化したUnfold/Fold技法を用いたPrologプログラムの変換合成

金森 直　　堀内謙二

(株)三菱電機 中央研究所

1. はじめに

プログラムの動きの詳細を最初から頭の中で追わずに、問題をいったん表現力の高い記述言語で書き下ろしてしまい、それから計算機の支援の下でより実用的なプログラムへたどりつくことができれば…というのが将来のプログラミング環境への夢の一つである。

本稿では、一般化したUnfold/Fold技法を用いた論理プログラムの変換合成について述べる。この方法は、四つの基本的な変換合成規則よりなり、新しく変換合成したい手続きを記述する式として一階述語論理式のある部分クラスが許されている。これは、玉木－佐藤のプログラム変換[3]とClarkのプログラム合成[1]の一部を含み、同値性も保証される[2]。

2. 準　備

まず、大域変数と自由変数を定義する。$\mathcal{F}$を一階論理式とする。$\mathcal{F}$内で限量されていない変数を大域変数といい、$\forall X\mathcal{G}$が$\mathcal{F}$の正の部分式であるか$\exists X\mathcal{G}$が$\mathcal{F}$の負の部分式である時、この変数Xを自由変数という。

次に、通常のアトムをゴールに一般化する。

(1) 新述語によるアトムはゴールである。

(2) 大域変数と自由変数しか含まれない旧述語によるアトムだけからなる論理式に対して、自由変数Xを!Xで置き換え、すべての限量記号を取り除いた論理式はゴールである。

ここで、新述語とは以下で述べるdefinite formula プログラムで定義される述語であり、旧述語とはdefinite clause プログラムによって定義される述語である。

次に、definite clauseを新述語を定義するためのdefinite formulaに拡張しよう。次のような式はdefinite formulaと呼ばれる。

$$A \text{ :- } G_1, G_2, \ldots, G_m. \qquad (m \geq 0)$$

ここで、G_1, G_2, ..., G_mは共通の自由変数を持たないゴールである。

最後に、代入のあるクラスを定義しよう。ゴールGに対するある代入σがG内のどの自由変数もinstantiateしない時σを正の代入と呼び、G内のどの大域変数もinstantiateしない時σを負の代入と呼ぶ。

3. Unfold/Foldによる変換合成

3.1. 変換合成のプロセス

我々の変換合成システムは、玉木－佐藤による変換と同様に次の様に行われる。

P_0 := 初期プログラム;　D_0 := { };
P_0内の各clauseには"foldable"とマークを付ける。
for i :=1 to 任意のN
　変換合成規則のどれかを適用することによって
　P_{i-1}とD_{i-1}からP_iとD_iを得る。

図1　変換合成のプロセス

3.2. 変換合成規則

基本的な変換合成規則は、定義、正の展開、負の展開、たたみ込みの四つである。

定　義　Cをp(X_1,X_2,...,X_n) :- G_1,G_2,...,G_m の形のdefinite formulaとしよう。ここで、

(1) pはP_{i-1}やD_{i-1}に現れない新しい述語である。

(2) X_1,X_2,...,X_nは相異なる大域変数である。

(3) G_1,G_2,...,G_mの述語はすべてP_0に現れる。

この時、P_iは$P_{i-1}\cup\{C\}$となり、D_iは$D_{i-1}\cup\{C\}$となる。Cには"foldable"のマークは付けない。

〔例〕　N×Nの将棋盤に互いに取られないように各列に"竜"を置く問題を考えよう。perm,insert等の基本的なプログラムを用いて、PがQの置換リストでPのすべての隣接する要素の差が2以上であるような関係dragons(P,Q)を定義しよう。

```
perm([ ],[ ]).
perm([X|L],M) :- perm(L,N),insert(X,N,M).
insert(X,N,[X|N]).
insert(X,[Y|N],[Y|M]) :- insert(X,N,M).
away(X,Y) :- distance(X,Y,D),1<D.
adjacent(X,Y,[X,Y|P]).
adjacent(X,Y,[Z|P]) :- adjacent(X,Y,P).
```

C1 : dragons(P,Q) :-
　　perm(P,Q),adjacent(!X,!Y,P)⊃away(!X,!Y).

Construction of Logic Programs Based on Generalized Unfold/Fold Rules
Tadashi KANAMORI, Kenji HORIUCHI
Central Research Lab. MITSUBISHI Electric Corp.

正の展開　P_{i-1}内の新述語を定義するdefinite formulaをCとし、その本体にあるゴールG内の自由変数を持たない正のアトムをAとし、アトムAが停止的であるか、または、must-be-true[2]であるならば、

(1) もしP_{i-1}にAとunify可能なclauseがないなら、C内のAを*false*で置き換えたものをC'_1とし、

(2) もしAとunify可能なP_{i-1}のすべてのdefinite clause (C_1,...,C_kとする)に対して、それらすべてがAと正の代入σ_iでunify可能であるなら、C内のAを$\sigma_i(C_i)$の本体で置き換えたものをC'_iとしよう。新しく導入された変数は大域変数とする。

その時、P_iは$(P_{i-1}-\{C\})\cup\{C'_1,...,C'_k\}$となり、$D_i$は$D_{i-1}$となる。$C'_i$には"foldable"マークが付けられる。

〔例〕　C1をperm(P,Q)で正の展開し、

```
C2 : dragons([ ],[ ]) :- adjacent(!X,!Y,[ ])⊃away(!X,!Y).
C3 : dragons([U|P],Q) :- perm(P,R),insert(U,R,Q),
       adjacent(!X,!Y,[U|P])⊃away(!X,!Y).
```

その後、C3をperm(P,R)でもう一度正の展開する。

```
C4 : dragons([U],Q) :-
       insert(U,[ ],Q),adjacent(!X,!Y,[U])⊃away(!X,!Y).
C5 : dragons([U,V|P],Q) :- perm(P,S),insert(V,S,R),
       insert(U,R,Q),adjacent(!X,!Y,[U,V|P])⊃away(!X,!Y).
```

負の展開　P_{i-1}内の新述語を定義するdefinite formulaをCとし、その本体にあるゴールG内の負のアトムをAとしよう。アトムAが停止的であり、Aとunify可能なP_{i-1}内のすべてのdefinite clause (C_1,...,C_kとする)に対して、それらすべてがAと負の代入σ_iでunify可能ならば、G内のAを*false*で置き換えたものをG_0とし、G内のAを$\sigma_i(C_i)$の本体で置き換えたものをG_iとしよう。新しく導入された変数は自由変数である。そしてC内のGをゴール列G_0,...,G_k置き換えたものをC'としよう。その時、P_iは$(P_{i-1}-\{C\})\cup\{C'\}$で、$D_i$は$D_{i-1}$である。$C'_i$には"foldable"マークが付けられる。

〔例〕　C2,C4,C5をadjacentでそれぞれ負の展開すると、次のようなformula C6,C7,C8を得る。

```
C6 : dragons([ ],[ ]).
C7 : dragons([U],Q) :- insert(U,[ ],Q).
C5 : dragons([U,V|P],Q) :-
       perm(P,S),insert(V,S,R),insert(U,R,Q),
       adjacent(!X,!Y,[U,V|P])⊃away(!X,!Y).
                ↓
   dragons([U,V|P],Q) :-
       perm(P,S),insert(V,S,R),insert(U,R,Q),
       false⊃away(!X,!Y),
       true⊃away(U,V),
       adjacent(!X,!Y,[V|P])⊃away(!X,!Y).
                ↓
C8 : dragons([U,V|P],Q) :-
       perm(P,S),insert(V,S,R),insert(U,R,Q),
       away(U,V),adjacent(!X,!Y,[V|P])⊃away(!X,!Y).
```

たたみ込み　CをP_{i-1}内の"A :- F_1,...,F_n"の形をしたdefinite clauseとし、C_{folder}をD_{i-1}内の"B :- G_1,...,G_m"の形をしたdefinite formulaとする。この時、代入σとCの本体の部分集合$\{F_{i1},...,F_{im}\}$について、

(1) j=1,...,mに対して、$F_{ij}=\sigma(G_j)$であり、

(2) σはC_{folder}の内部変数には異なる変数を代入し、それらの変数は$\{F_{i1},...,F_{im}\}$以外には現れず、

(3) Cに"foldable"マークが付いているかまたはm<nであるならば、

C内の$\{F_{i1},...,F_{im}\}$をσ(B)で置き換えたものをC'としよう。その時、P_iは$(P_{i-1}-\{C\})\cup\{C'\}$となり、$D_i$は$D_{i-1}$となる。$C'$はCのマークを引継ぐ。

〔例〕　C8の本体をpermでたたみ込んだ後、dragonsの定義でたたみ込むことによってC9を得る。また、C7をinsert(U,[ ],Q)で正の展開することによってC10を得る。

```
C7 : dragons([ ],[ ]).
C10 : dragons([U],[U]).
C9 : dragons([U,V|P],Q) :-
       insert(U,R,Q),away(U,V),dragons([V|P],R).
```

これは盤の端から駒を置いて試していくタイプのプログラムになっている。

4. おわりに

以上、一般化したunfold/fold技法を用いた論理プログラムの変換合成について述べた。また、これらの手法を導入したシステムとして、Prologプログラムの変換合成システムArgus/Cが試作されており、DEC-2060のDEC10-Prolog上で稼働している。

謝　辞

この研究は、第五世代計算機プロジェクトの一環として行われたものである。このような研究の機会と激励を頂いたICOT淵一博所長、横井俊夫第2研究室長に感謝致します。また、多くの助言やコメントを頂いた玉木久夫氏(茨城大学)と佐藤泰介氏(電総研)の両氏に感謝致します。

参考文献

[1] Clark,K.L., "Predicate Logic as A Computational Formalism", Chap.5, Research Monograph : 79/59, TOC, Imperial College, 1979.

[2] Kanamori,T. and K.Horiuchi, "Construction of Logic Programs Based on Generalized Unfold/Fold Rules", ICOT TR-1??, to appear, 1986.

[3] Tamaki,H. and T.Sato, "Unfold/Fold Transformation of Logic Programs", Proc. of 2nd International Logic Programming Conference, pp.127-138, 1984.