4X-9

Implicitな定義からの変換による Prologプログラムの導出

金森 直　　前地 真知

三菱電機(株) 中央研究所

1．はじめに

効率は考えずに判り易さを最優先に書いた宣言的プログラムから出発して(判りにくいかも知れないが)効率的なプログラムを作り出すというアプローチの基本的手法としてunfold/fold 技法が知られている。

この変換のヒューリスティクスの重要なものの1つに、ゴールの挿入がある。導き出されたclauseを最初に与えたclauseでたたみ込めるよう故意にあるゴールを挿入するというもので文献(1)(2)ではこれを"forced folding"として紹介している。また(3)では "(folding driven) goal insertion"と呼び図1のようなsortプログラムを扱っている。

```
sort(L,M) :- perm(L,M),ordered(M).
↓    展開 (unfolding)
sort([ ],[ ]).
sort([X|L],M) :- perm(L,N),insert-randomly(X,N,M),ordered(M).
↓    ゴール"ordered(N)"を挿入する (goal insertion)
sort([ ],[ ]).
sort([X|L],M) :- perm(N,L),ordered(N),
                 insert-randomly(X,N,M),ordered(M).
↓    sortの最初のプログラムで畳み込む (folding)
sort([ ],[ ]).
sort([X|L],M) :- sort(L,N),insert-randomly(X,N,M),ordered(M).
```

図1. ゴールの挿入の例(玉木－佐藤)

ゴールの挿入が行われる直前のclauseについて見てみよう。この本体中の insert-randomly(X, N, M), ordered(M)が成り立つとき常にordered(N)も成り立つ。もし、本体中にperm(L, N), ordered(N)が出そろっているとこれらはsortの最初のclauseでたたみ込めて再帰的なclauseが出来上がる。このたたみ込みを行うために"ordered(N)"を挿入している。

しかしながら一般には“ゴールの挿入”を unfold/foldルールと合わせると、最小Herbrandモデルの意味での“同値性”が失われるかも知れない。またそれを保証するには少し面倒な余分な処理が必要になる(3)。本稿では、この操作を自然に行う為のImplicitな定義によって一般化されたPrologプログラムの変換合成について述べる。

2．Implicitな定義

これは定義するclauseの頭部に複数個のアトムを許したもので、上のsortの例では図2の様に定義される。

```
ordered(N), insert-properly(X, N, M) :-
        insert-randomly(X, N, M), ordered(M).
```

図2. Implicitな定義の例

insert-properly が新しく定義した述語である。clauseの本体へ"ordered(N)"を挿入する代りにそのclauseを今定義したclauseでたたみ込むと同様の結果が得られる。

このImplicitな定義でunfold/fold を用いた変換合成規則の1つ“定義”を拡張し、他の規則の修正も行なう。

3．Implicitな定義によって一般化された変換合成規則

一般化された変換合成規則は以下に示す5つから成っている。このうちの規則“定義”は、新しい述語を定義するclauseをプログラムに追加し、他の規則はプログラム中の1つのclauseを変換した幾つかのclauseで置き換える。

ここで扱われるdefinite clause には2種類ある。1つはA0:-A1, A2, …, Am. (m≥0)という形の論理式で、変換合成過程の最初から与えられているものである。これをexplicit definite clause と呼び、カウンタを1とする。これで定義している述語のアトムを旧アトムと呼ぶ。もう1つが、B1, B2, …, Bs, p(X1, X2, …, Xn) :-Bs+1, Bs+2, …, Bs+r. (s, r≥0)という形のImplicit な定義で与えられるclauseで、general definite clauseと呼び、変換合成はこのclause に対してのみ行われる。ここに、B1, B2, …, Bs, Bs+1, Bs+2, …, Bs+r はすべて旧アトム、p は新しい述語である。このA1, A2, …, Amの中、B1, B2, …, Bsの中あるいはBs+1, Bs+2, …, Bs+rの中には∃で束縛されている変数(内部変数と呼ぶ)があっても良い。それ以外の変数は大域変数と呼ぶ。またdefinite clauseの :- の右側をそのclauseの本体、左側を頭部と呼ぶ。

それぞれの規則について説明していこう。(例参照)

定義(Implicit Definition) 次の条件を満たす上の様なgeneral definite clauseをカウンタを1として定義する。

i) 本体の中の大域変数は必ず頭部の中に現われる。

ii) アトムp(X1, X2, …, Xn)においてpは新述語、X1, X2, …, Xnは頭部中の大域変数をすべて含む相異なる変数である。

iii) 最初から与えられているexplicit definite clause全体から成るプログラムにおいて Bs+1∧Bs+2∧…∧Bs+rが成り立つとき常にB1∧B2∧…∧Bsが成り立つ。

Derivation of Logic Programs from Implicit Definition

Tadashi KANAMORI, Machi MAEJI

Central Research Lab. MITSUBISHI Electric Corp.

本体展開(Body-Unfolding)　カウンタγの general definite clause の本体中の"禁止"マークがついていない旧アトムをそれと単一化可能な頭部をもつ explicit definite clauseで展開する。出来たclauseのカウンタはγ+1とする。

頭部展開(Head-Unfolding)　カウンタγが正数の general definite clause の頭部中にある旧アトムをそれと単一化可能な頭部をもつexplicit difinite clauseで展開し、カウンタをγ-1とする。これの適用はこの旧アトムと単一化可能なexplicit definite clauseが唯一つの時に限る。

たたみ込み(Folding)　カウンタがγの general definite clauseの本体中の旧アトム集合が、定義した形の general definite clause の本体のあるinstanceと一致する時それでたたみ込みカウンタをγ-1とする。これで現われた本体中の旧アトムには"禁止"マークを付ける。これの適用は各clauseに対して1度までで次の条件を満たす時に限る。

i) たたみ込まれる旧アトム集合の中の内部変数は、そのclauseの他のアトム内に現われない。

ii) たたみ込まれるclauseの本体中のアトム数をn そのカウンタをγ たたみ込むclauseの本体中のアトム数をmとした時m+1<n+γが成り立つ。

相殺(Cancellation)　general definite clause の頭部と本体に同じ旧アトム集合がある時これらを消し去る。カウンタは変わらない。これの適用はこの旧アトム集合の内部変数がclauseの他のアトム中に現われない時に限る。

簡単な例について上記の変換規則の適用を図3に示そう。?のついた変数は∃で束縛された内部変数である。

```
{ explicit definite clauses }
 [1] append([ ],M,M).
 [1] append([X|L],M,[X|N]) :- append(L,M,N).
{ general definite clauses }
 [1] append(M,N,?P),append(L,?P,A), p(L,M,N,A) :-
                                append(L,M,?Q),append(?Q,N,A).
  |        本体展開  (Body-Unfolding)
  ↓
 [2] append(M,N,?P),append([ ],?P,A), p([ ],M,N,A) :- append(M,N,A).
 [2] append(M,N,?P),append([X|L],?P,A), p([X|L],M,N,A) :-
                                append(L,M,?Q), append([X|?Q],N,A).
  |        本体展開  (Body-Unfolding)
  ↓
 [2] append(M,N,?P),append([ ],?P,A), p([ ],M,N,A) :- append(M,N,A).
 [3] append(M,N,?P), append([X|L],?P,[X|A']), p([X|L],M,N,[X|A']) :-
                                append(L,M,?Q),append(?Q,N,A').
  |        頭部展開  (Head-Unfolding)
  ↓
 [1] append(M,N,A),p([ ],M,N,A) :- append(M,N,A).
 [2] append(M,N,?P),append(L,?P,A'), p([X|L],M,N,[X|A']) :-
                                append(L,M,?Q),append(?Q,N,A').
  |        畳込み  (Folding)
  ↓
 [1] append(M,N,A), p([ ],M,N,A) :- append(M,N,A).
 [1] append(M,N,?P),append(L,?P,A'), p([X|L],M,N,[X|A']) :-
                        *append(M,N,?P'),*append(L,?P',A'), p(L,M,N,A').
  |        相殺  (Cancellation)
  ↓
 [1] p([ ],M,N,A).                          *  ･･･ 禁止マーク
 [1] p([X|L],M,N,[X|A']) :- p(L,M,N,A').    [γ] ･･･ カウンタ
```

図3．変換合成の例

4．同値性の保存について

この変換合成規則に対して同値性は保たれるのだろうか。

Implicitな定義で定義されたclauseの頭部は1つ以上のアトムから成っている為、証明木を考える場合も個々のアトムを個別に考えるのではなく幾つかのアトムの集合（分子と呼ぶ）として考える必要がある。また、頭部の旧アトムは変換合成の過程で頭部展開、相殺などによって変化するので、変換の前後でのプログラムの成功する分子の集合は必ずしも一致しない。しかし我々の目的とするプログラム変換のためには完全に一致する必要はない。最初に与えたclauseと変換合成過程で定義したclauseをすべて合わせたプログラムにおいてそれらのclauseの頭部の大域変数すべてにground term を代入して得られた分子全体をHで表わすと、Hの中の成功する分子の集合が変換の前後で等しいことさえ示せばよい。これは(i)禁止マークのついた本体のアトムを展開、たたみ込みの対象としない、(ii)同じclauseに2回以上たたみ込みを適用しない、という2つの制限のもとでは（普通の、頭部にアトム1つだけの場合に較べると少し面倒ではあるが）証明できる。[7]

5．おわりに

以上、Implicitな定義によって一般化されたPrologプログラムの変換合成について述べた。この方法は現在変換合成システムArgus/C(1.5版）に実装中である。

謝　辞

本研究は第5世代計算機プロジェクトの一環として行われた。研究の機会と激励を頂いた渕一博ICOT所長、横井俊夫同第2研究室長に感謝致します。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。

文　献

[1] Clark, K. L. and J. Darlington. "Algorithm Classification Through Synthesis". The Computer Journal, Vol. 23, No.1, pp. 61-65, 1980

[2] Darlington, J.. "An Experimental Program Transformation and Synthesis System". Artificial Intelligence, Vol. 16, pp. 1-46, 1981.

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

[4] Tamaki, H. and T. Sato. "Compatibility of Replacement Rules with Unfold/Fold Transformation". to appear, 1986.

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

[6] Kanamori, T. and H. Fujita. "Unfold/Fold Transformation of Logic Programs with Counters". ICOT TR-1??, to appear, 1986.

[7] Kanamori, T. and M. Maeji. "Derivation of Logic Programs from Implicit Definition". ICOT TR-1??, to appear, 1986.