7P-7

例題からのプログラム合成について

石橋勇人，西田豊明，堂下修司
(京都大学工学部)

1．はじめに

例題からプログラムを合成しようとする場合の主な問題として、1）例題によって示された関係をいかにして一般化し、プログラム化するか、2）一度合成したプログラムがユーザの意図する仕様を満たさなかった場合に、どのようにして修正を行うか、の2点が挙げられる。本稿では、このうち、1）の与えられた例題からプログラムを合成する方法について、リスト構造に変形を加えるLispプログラムの合成を対象に取り扱う。このために、ＣＯＮＳ木の一般化による合成、入出力とトレースからの合成、例題プログラムの拡張の３つの手法を示す。

2．例題からのプログラム合成

2．1　基本方針

プログラム合成とは、与えられた情報をもとにユーザの描いている仕様を満たすプログラムを生成することである。しかし、あるクラスのすべての問題に対して正しく動作するプログラムを合成することは、一般に困難である。これに対して、１つの例題（入出力関係）を満たすプログラムを合成することは、比較的容易である。また、１つの例題を満たすプログラムのカバーする範囲を広げていくことによって一般性を持つプログラムを生成することが可能であると考えられる。

我々は、与えられた例題の解法を発見し、それを一般化することによって求めるプログラムを生成するという方針に基づいてプログラム合成の研究を進めている。すなわち、まず、与えられた例題を満足するベースプログラムを合成し、それをさらに一般化することによって最終的なプログラムを生成する。

2．2　ベースプログラムの合成について

一般に、ドメインに関する知識が全くない状態からプログラムを合成することは、困難であり、また、我々がプログラムを書く場合であっても、そのような方法をとっているとは考えにくい。従って、ある程度現実的にプログラム合成を行おうとすれば、知識の導入は不可欠であると思われる。しかし、全く知識のみに頼ってプログラムを合成する場合には、複数のモジュールから構成されるようなプログラムの場合に、それぞれのモジュールを知識から導くことはできても、導かれたそれぞれのプログラムモジュール間の結合が難しい。従って、知識に頼らず全く白紙の状態からプログラムを合成できる能力も必要である。

次章では、３．１，３．２において白紙の状態からプログラムを合成する手法について、また、３．３ではそれらを拡張、一般化する手法について述べる。

3．合成アルゴリズム

3．1　ＣＯＮＳ木の一般化による合成

例題として与えられた入出力対からプログラムを合成するには、出力リストがいかにして入力リストから得られたかを解析する必要がある。

例題として｛（Ａ　Ｂ　Ｃ），（Ａ　Ａ　Ｂ　Ｂ　Ｃ　Ｃ）｝を与えた場合を考えると、まず、出力リストの構造を図１(a)のような形に表現する。この木構造の各非末端ノードに対して、一つずつ関数を割り当てることによって、図１(b)のＦ１～Ｆ６が得られる。ここで、(b)の各関数の間でマッチングをとり、同一と考えられる関数をマージしていくことによって再帰関数を合成する。マッチングは以下のような規則に基づいて行う。

(a) 定数は、自分自身あるいはプリミティブ（car, cdr, cons）でない関数式（(F1 (CDR X))等）とマッチする。

(b) 関数式は、関数式とマッチする。

(c) 対応する位置の要素がすべてマッチしたとき、プリミティブでない関数どうしがマージされる。

具体的には、まず、Ｆ５とＦ６を同一と仮定してマージするとＦ５’が得られる。しかし、関数Ｆ５’は明らかに停止しない。従って、Ｆ５’は放棄し、次にＦ６とＦ４をマージする。この結果、関数Ｆ４’が得られる。Ｆ４＝Ｆ６＝Ｆ４’よりＦ３＝Ｆ５、さらにＦ２＝Ｆ４、Ｆ１＝Ｆ３となり、最終的に図１(c)のような結果が得られる。

3．2　入出力対とそのトレースからの合成

例題として、入出力対という静的な情報を与えるだけでは、情報量が不足してプログラムが合成できない場合がある。本節では、入出力対｛Ｉ，Ｏ｝の他に、その入力に対してプログラムがどのように動作するかを示すトレースＴを加えた３つ組｛Ｉ，Ｏ，Ｔ｝を例題として扱う。

図２は、｛Ｉ，Ｏ｝＝｛（Ａ　Ａ　Ｂ　Ｃ　Ｂ），（（Ａ　Ａ）（Ｂ　Ｂ）（Ｃ））｝、すなわち、入力Ｉとして与えられたリスト中の同じ要素を集めるというプログラムを合成する例である。

図２(a)に示すように、トレースは、各ノードにその時点での入出力対を持ち、与えられた入力から出力が合成されていく過程を示している。このトレースの隣り合うノード間の入力、出力をそれぞれ比較することにより、プログラムの各段階において行われる操作を推測することが可能となる。図２(a)の場合には、入力リストの差異より、その時点での入力リストのＣＤＲをとったものが次のステップの入力となっていることがわかる。また、出力リストの変化と入力リストを比較することにより、各段階において入力リストのＣＡＲから出力が構成されていくことがわかる。さらに、入力リストのＣＡＲによって出力の得られかたが決定されることが推測される。

これらの結果より、トレースのグラフをマージし、一般化することによって、求めるプログラムのコントロールフローを作成する（図２(b)）。最終的に、図２(c)のようなプログラムが得られる。

3．3　例題プログラムの拡張

ある例題を満たす典型例としてプログラムが与えられた場合に、あらかじめ与えられた知識を利用してそ

Some Considerations on Program Synthesis from Example

Hayato ISHIBASHI, Toyoaki NISHIDA, Shuji DOSHITA
Kyoto University

れを拡張し、一般化することが可能であると考えられる。例えば、リスト構造の１レベルだけの順序を逆にするプログラムを与えられた場合に、そのプログラムに対して一般化オペレータを作用させることによって、すべてのレベルにおいて順序を反転するプログラムを作り出すことができる。すなわち、リスト構造の一定の深さについて処理を行うプログラムから不定の深さについて処理を行うプログラムへと一般化（拡張）するわけである。

図３(a)は１レベルのみの順序を反転するプログラムであるが、これを観察してみると、その本体は関数APPENDの部分であること、APPENDの第１引数は再帰呼び出しとなっているので、実質的な“操作”が行われるのは第２引数の部分であることがわかる。従って、すべてのレベルの順序を反転するためには、この第２引数の部分に手を加えればよいことになる。第２引数の部分でも再帰呼び出しを行うように書き換えることにより、図３(b)が得られる。

一般化オペレータとしては、リスト構造を扱う場合、リストの深さ方向への一般化を行うもの（深さ一般化オペレータ）、長さ方向への一般化を行うもの（長さ一般化オペレータ）を用意する。

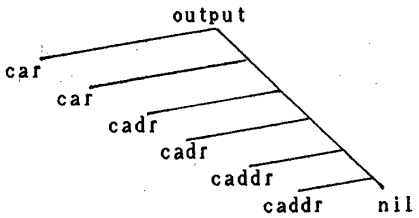
このように典型例からの一般化が可能なシステムにおいては、システム自身がそのような典型例、すなわち、与えられた例題を満足するような解プログラムを合成することができれば、それに対して一般化オペレータを作用させることによって、（与えられた例題だけでなく）一般の入力に対して正しく動作するプログラムを合成することができる。

３．１あるいは３．２で述べたような方法を単独に用いて最終的なプログラムを合成する方法では、しばしば過剰な一般化が行われたためにユーザの意図しない結果が得られることがある。しかし、このような一般化オペレータを用いることによって、まず控え目な一般化を行ってプログラムを合成し、その後システムあるいはユーザの判断によって一般化オペレータを作用させることにより、過剰な一般化を防ぎつつプログラムを合成することが可能となる。

４．おわりに

本稿では、与えられた例題からプログラムを合成するシステムについて、ＣＯＮＳ木の一般化による方法、入出力対とトレースを用いる方法、例題プログラムの拡張の３つの手法を示した。現在、これらの手法を用いたプログラム合成システムを作成中である。今後は、知識を利用した合成についても研究を進めるとともに、ここで述べた方法との融合について考慮していく予定である。

本研究の一部は、文部省科学研究費
特定研究(1)61102003　奨励研究(A)61780039
によって行われた。



(a) 出力リストの構造

```
F1 = (CONS (CAR X)(F2 X))
F2 = (CONS (CAR X)(F3 (CDR X)))
F3 = (CONS (CAR X)(F4 X))
F4 = (CONS (CAR X)(F5 (CDR X)))
F5 = (CONS (CAR X)(F6 X))
F6 = (CONS (CAR X) NIL)

F5' = (COND ((NULL X) NIL)
            (T (CONS (CAR X)(F5' X))))
F4' = (CONS (CAR X)
            (COND ((NULL (CDR X)) NIL)
                  (T (F5 (CDR X)))))
```

(b) (a)より得られた関数列

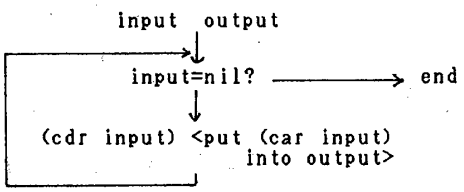
```
F1 = (CONS (CAR X)(F2 X))
F2 = (CONS (CAR X)
           (COND ((NULL (CDR X)) NIL)
                 (T (F1 (CDR X)))))
```

(c) 合成結果

図１　CONS木の一般化による合成

```
(A A B C B)     ()
      ↓
(A B C B)     ((A))
      ↓
(B C B)     ((A A))
      ↓
(C B)    ((A A)(B))
      ↓
(B)  ((A A)(B)(C))
```

(a) トレース



(b) コントロールフロー

```
F(INPUT OUTPUT)
 = (COND ((NULL INPUT) OUTPUT)
         (T (F (CDR INPUT)
               (G (CAR INPUT) OUTPUT))))
G(INPUT OUTPUT) = <put (car input)
                       into output>
```

(c) 合成結果

図２　トレースからの合成

```
(DEFUN REV_1 (X)
   (COND ((ATOM X) X)
         (T (APPEND (REV_1 (CDR X))
                    (LIST (CAR X))))))
```

(a) １レベルのREVERSE

```
(DEFUN REV_N (X)
   (COND ((ATOM X) X)
         (T (APPEND (REV_N (CDR X))
                    (LIST (REV_N (CAR X)))))))
```

(b) Ｎレベルの REVERSE

図３　例題プログラムの拡張