2D-3

代数的記述に基づく抽象的順序機械の処理系の設計

鷲坂　光一　　杉山　裕二　　鳥居　宏次

(大阪大学　基礎工学部)

1．まえがき

抽象的順序機械は，代数的記述の一種であり，関数型プログラムあるいは手続き型プログラムに系統的に変換する手法が知られている[1]．その手法を用いて，対話形式により，抽象的順序機械のプログラムを関数型言語ＡＳＬ／Ｆのプログラムに変換するシステムが既に作成されている[2]．このシステムでは，入力として許される抽象的順序機械が非常に限定されている．また，処理順序がある程度指定されている抽象的順序機械をそのような概念のない関数型プログラムに変換しているため，実行系としては回り道をしており（ＡＳＬ／Ｆには最適化コンパイラが作られているが）効率の低下はまぬがれない．このため，変換可能な抽象的順序機械の仕様を拡大するとともに，抽象的順序機械を手続き型言語に変換する処理系を作成中である．ここでは，従来のシステムの仕様の改善点と，手続き型言語への変換の際に生じる問題点について報告する．

2．抽象的順序機械

抽象的順序機械は，ブール，整数などの基本データタイプ（ソートと呼ぶ）に関する基本関数および条件文に相当するIF関数が定義されている基底代数を前提とする代数的仕様記述法の一つである[3]．

抽象的順序機械は，記述の対象を状態遷移するものとしてとらえる．そして，状態遷移に相当する状態遷移関数，状態から値を取り出す状態出力関数により，状態遷移後の各状態出力関数の値を公理の形で記述する(図１の{18}～{23})．また，初期状態を作る状態初期化関数により初期状態の状態出力関数の値(図１の{15}～{17})，ループ関数により状態遷移の制御の流れ(図１の{13}～{14})，目的関数により入出力(図１の{12})を公理の形で記述する．プログラムは書き手が新たに導入した関数（定義関数と呼ぶ）の引数および関数値の宣言部と前述の公理の記述部から構成される．

3．仕様の主な改善点

入力として許される抽象的順序機械が非常に限定されていたので，以下の改善を行った．

```
      SPEC BUBBLE_SORT;
{1}   INCLUDE array(ary,int,50);
{2}   RANGE I: 0..50 of int;
      op   /* target function  */
{3}      bubble_sort : ary,int   -> ary   ;
           /* loop function    */
{4}      bubble      : state     -> state ;
{5}      bub         : state     -> state ;
           /* initial function */
{6}      initial     : ary,int   -> stste ;
           /* s_function       */
{7}      exch        : state     -> state ;
{8}      next        : state     -> state ;
           /* v_function       */
{9}      a           : state,int -> int   ;
{10}     j,k         : state     -> int   ;
           /* aux function     */
{11}     swap        : state,int -> int   ;
      AXIOM
{12}     bubble_sort(A,n) == a(bubble(initial(A,n)),*);
{13}     bubble(S) ==
               if J(S) = 0
                 then S
               else bubble(bub(S));
{14}     bub(S) ==
               if k(S) >= J(S)
                 then next(S)
               else bub(exch(S));
{15}     a(initial(A,n),I) == aget(A,I);
{16}     J(initial(A,n)) == n;
{17}     k(initial(A,n)) == 0;
{18}     a(next(S),I) == a(S,I);
{19}     J(next(S)) == J(S)-1;
{20}     k(next(S)) == 0;
{21}     a(exch(S),I) ==
               if a(S,k(S)) < a(S,k(S)+1)
                 then swap(S,I)
               else a(S,I);
{22}     J(exch(S)) == J(S);
{23}     k(exch(S)) == k(S)+1;
{24}     swap(S,I) ==
               if I = k(S)
                 then a(S,k(S)+1)
               else if I = k(S)+1
                 then a(S,k(S))
               else a(S,I);
      END;
```

{1}はソートが ary，要素のソートが int，要素数が50の配列の宣言．agetは配列要素を参照する関数．

図１　抽象的順序機械によるバブルソートのプログラム

1)　制御構造の改善．

制御構造として，while文的なループ文を一つしかテキストに記述できなかった．また，連続した状態遷移の記述も許されなかった．このため，問題記述の際は，アルゴリズムを組み直す，問題を展開する，ループごとにテキストを分離する等の必要があった．複数のループ文，連続した状態遷移の記述を許したため，アルゴリズムに即して問題の記述が行える．

On A Translator from Abstract Sequential Machines to A Procedual Language

Mitsukazu WASHISAKA, Yuji SUGIYAMA and Koji TORII

Osaka Univ.

2) 状態出力関数の改善.

状態出力関数は，状態ソートだけを引数に持つ一引数関数であった．このため，配列は全体を単位とする記述しかできず，配列要素を参照・更新する関数を頻繁に用いる必要があった．また，テーブルを直接記述できないため，データ構造を分割しなければならなかった．例えば，図１の{21},{24}を従来通り記述すると図２になる．

```
{25}   a(exch(S)) ==
           if aget(a(S),k(S)) < aget(a(S),k(S)+1)
             then swap(S)
           else a(S);
{26}   swap(S) ==
           aput(aput(a(S),k(S),aget(a(S),k(S)+1),
                k(S)+1,
                aget(a(S),k(S)));
```

図２　図１の{21},{24}の従来の記述.

状態ソート以外の引数を持つ状態出力関数の記述を許すことにより，配列要素ごとの状態遷移，テーブルの記述が可能になる．例えば，配列要素のシフトは，次のように書くことができる．

```
a(next(S),i) == a(S,mod(i,50)+1);
```

3) 基本関数の改善.

配列のソート名，配列要素を参照・更新する関数名の重複を許さなかった．また，型変換を陽に記述する必要があった．このため，同型の配列に異なるソート名・関数名を割当てる必要性や複雑な関数のネストにより，記述の簡潔さや他のプログラムとの統一性に問題があった．基本関数名の統一，暗黙の型変換を許すことにより，プログラムを簡潔に記述できる．

４．変換の問題点

従来のシステムでは，抽象的順序機械を関数型言語ＡＳＬ／Ｆに変換し，UNIX上に作成されているＡＳＬ／Ｆコンパイラ[4]を用いてＣ言語に変換し実行していた．状態遷移を陽に記述した抽象的順序機械では，実行順序が状態遷移という形である程度指定されているにもかかわらず，そのような概念のない関数型プログラムに変換し，最適化によって再び実行順序（状態遷移）を決めるという処理が行われていた．今回は，直接，手続き型言語に変換することで実行効率の改善を図る．変換方法自体は従来のものと同様，抽象的順序機械から決定表を抽出し，それに従ってプログラムを生成するものである．しかし，手続き型言語へ変換するということで新たに発生した問題がいくつかある．その主なものを以下に挙げる．

1) 公理の実行順序の決定.

抽象的順序機械は，状態遷移前の各状態遷移関数の値をコピーしておき，状態遷移後の値を計算すれば容易に実行可能である．しかし，この方法は実行効率が低下するということや，ファイル・配列等コピーを作ることに無理のあるソートが存在するという問題点がある．若干のコピーは必要であるが，基本的には各状態遷移関数に関して，一つの記憶領域で実行が可能であると思われる．よって，コピーを最少にする公理の実行順序を決定しなければならない．

2) 公理の部分項の実行順序の決定.

ファイル・配列等の処理では，状態変化をもたらす基本関数（ファイルからの読み込み，配列要素の更新等）が存在するため，部分項の実行順序（図２の{26}では，右辺を右側から実行するか，左側から実行するか）によって値が異なる．プログラムの意味を変えないため，公理の部分項の実行順序を正しく決定しなければならない．また，配列要素ごとの状態遷移の記述を許したため，実行順序の決定の際は，各要素間の依存関係を認識する必要がある．

3) 複数の公理間の共通部分項の検出.

```
a(next(S)) == ...f(S)...f(S)...;
b(next(S)) == ...f(S)...;
```

同一の状態遷移後の状態出力関数の右辺では，プログラムの意味上，共通部分項（例のf(S)）は同一の値であり，実行効率をあげる上で再計算を除去できる．このため，複数の公理間の共通部分項の検出が必要である．

５．あとがき

変換の際に生じる問題の多くは，ソートの大域化（単一の記憶領域で実行が可能かどうか）に関するものが多く，完全な解決は困難であると思われるが，現在，その解決法を検討中であり，今後，変換システムの本格的な作成に入る予定である．

[参考文献]

[1]Torii,K.Morisawa,Y.Sugiyama,Y.and Kasami,T.: "Functional Programming and Logical Programming for the Telegram Analysis Problem", Proc. of IEEE 7th International Conf. on Software Engi., pp.463-472 (1984).

[2]安松，杉山，鳥居：" 抽象的順序機械から関数型プログラムへの変換システムについて"，情処研報，86-SE-46-4，(1986-2).

[3]杉山，谷口，嵩：" 基底代数を前提とする代数的仕様"，信学論(D)，Vol.J64-D，No.4，pp.324-331 (1981).

[4]関，八木，井上，杉山，後藤：" 関数型言語ＡＳＬ／ＦコンパイラのUNIX上での実現"，情処第30回全大，1Q-3，pp.373-374 (1985-3).