ソフトウェア工学　52 — 9
（１９８７・２・１３）

代数的記述に基づく抽象的順序機械の処理系の作成

鷲坂　光一　　井上　克郎　　杉山　裕二　　鳥居　宏次
大阪大学　基礎工学部

抽象的順序機械と呼ばれる代数的記述のクラスについては，関数型言語ＡＳＬ／Ｆに変換するシステムが作成されている．しかし，実行順序をある程度指定している抽象的順序機械は，手続き型言語に変換する方が望ましいと考えられる．現在，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．

本報告では，従来のシステムの仕様の改善点，手続き型言語に変換することにより生じる問題点，変換方法およびその際の最適化ついて報告する．特に，改善点として，制御構造および基本関数等の改善について，また，問題点として，実行順序の問題を取り上げその解決方法を考察する．

Problems on Translation of a class of Algebraic Specification

Mitsukazu WASHISAKA, Katsuro INOUE, Yuji SUGIYAMA and Koji TORII

Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

As for the implementation of a class of algebraic specifications, named abstract sequential machines, we constructed a system which translates the abstract sequential machine into a functional program in ASL/F.

Now, we develop a new translation system from the abstract sequential machine into a procedual language program, since we expect more efficent execution of the translated program than the previous system. In this paper, we show improvements in the new system and discuss problems on the translation.

1．まえがき

代数的記述言語は，意味の定義が簡明であり，形式的な検証が比較的容易かつ厳密に行える，自然言語の仕様に対応するような抽象度の高い記述から，効率的な実効が可能な低いレベルの記述まで，同一の意味定義の枠組の中で記述が行える等の利点を持つ．

一般の代数的言語を現実の計算機上で効率的に実行する方法は知られていないが，代数的記述のうち，抽象的順序機械と呼ばれるものの一部については，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の仕様が非常に限定されているため，問題記述が不自然で複雑なものになっている．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．本報告では，従来のシステムの仕様の改善点，手続き型言語への変換により生じる問題点，変換方法とその際の最適化について報告する．

2．抽象的順序機械

ここでは，抽象的順序機械の概念および記述例について述べる．

2．1抽象的順序機械の概念

ブール，整数などの基本データタイプ(通常ソートと呼ばれる)に関する基本演算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[3]において，基本データタイプ以外のデータタイプがただ一つであるような仕様のうち，新たに定義する関数および公理が次の①～③を満たすものを抽象的順序機械と呼ぶ．このとき，その新しいデータタイプを状態と呼び，stateで表わすことにする．

① 引数にデータタイプstateを２つ以上持つ関数は，IF関数を除いて，存在しない．便宜上，stateは第１引数に現われるものとする．

② 公理の左辺の形はｆ(x_1,x_2,…)または （ｇ(x_1,x_2,…),…)である．ただし，ｇの値域はstateである(ｆの第１引数の定義域，値域，　の値域はそれぞれstateであってもよい)．また，変数x_1,x_2,…はすべて異なる．

③ 同じ左辺を持つ公理は複数個存在しない．また，左辺が （ｇ(x_1,…),…)の形の公理があるとき，左辺が (x_1,…)やｇ(x_1,…)の形の公理はない．

この仕様は，Church-Rosserの性質を持ち，無矛盾である．

```
   SPEC   BUBBLE_SORT;
{1} TYPE   ary : array(int,50);
   PRODUCTION
       state  ->  bubble(state);
                  bub(state);
                  init(ary,int);
                  next(state);
{2}               exch(state);
       ary    ->  bubble_sort(ary,int);
                  a(state);
                  swap(state);
       int    ->  i(state);
                  k(state);
   AXIOM
{3}    bubble_sort(A,N) == a(bubble(init(A,N)));
{4}    bubble(S) == if i(S) = 1
                      then S
                    else bubble(bub(S));
{5}    bub(S) == if k(S) = i(S)
                   then next(S)
                 else bub(exch(S));
{6}    a(init(A,N)) == A;
{7}    i(init(A,N)) == N;
{8}    k(init(A,N)) == 1;
{9}    a(next(S)) == a(S);
{10}   i(next(S)) == i(S)-1;
{11}   k(next(S)) == 1;
{12}   a(exch(S)) ==
           if aget(a(S),k(S)) > aget(a(S),k(S)+1)
             then swap(S)
           else a(S);
{13}   i(exch(S)) == i(S);
{14}   k(exch(S)) == k(S)+1;
{15}   swap(S) ==
           aput(aput(a(S),k(S),aget(a(S),k(S)+1)),
                k(S)+1,
                aget(a(S),k(S)));
   END;
```

{1}はデータタイプが ary，要素のデータタイプが int，要素数が50の配列の宣言．
aget，aput は配列要素を参照・更新する関数．
swapは配列要素のk番目とk+1番目の交換を行う関数．

図１　抽象的順序機械によるバブルソートのプログラム

2．2記述例

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，初期状態から最終状態までを，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記述する．

例として抽象的順序機械によるバブルソートのプログラム(図１)，流れ図(図２)とアルゴリズム(図３)を示す．状態出力関数としてａ，ｉ，ｋを，状態遷移関数としてnext，exchを用いる．ａは整数配列を表わすために，ｉ，ｋはループの制御のために用いられる．入出力を記述する関数を目的関数(bubble_sort)と呼び，この場合，整数配列Ａ，要素数Ｎを入力とする．入力から初期状態を作る状態初期化関数(init)により初期状態の状態出力関数の値を，状態遷移関数と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記述する．例えば，図１{12} k(exch(S)) = k(S)+1 ；は状態遷移exchの後のｋの値が，exchが生じる前の値よりも１増加することを表わす．制御の流れを記述する関数をループ関数(bubble，bub)と呼び状態遷移の生じる順を記述する．公理の記述を簡単にするため，補助関数(swap)を導入することもできる．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．

```
for  i = N  downto  1  do
  for  k = 1  to  i-1  do
    if  a[k] > a[k+1]
      then swap(a, k, k+1) ;
```

図２　バブルソートのアルゴリズム

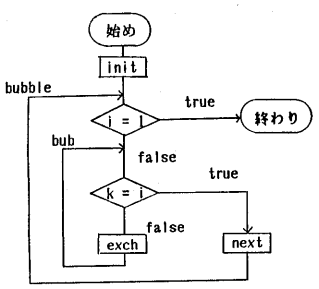

図３　図１のプログラムの流れ図

3．仕様の主な改善点

従来のシステムでは，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たので，以下の改善を行った．

1)　制御構造の改善

制御構造として，while文的なループ関数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．複数のループ関数，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．図１の以前の記述を図４に示す．図４では，図３の内側のループで行う処理をexchのみで行っているため複雑になっている．

```
{16}    bubble(S) == if i(S) = 1
                       then S
                     else bubble(exch(S));

{17}    a(exch(S)) ==
                if aget(a(S),k(S)) > aget(a(S),k(S)+1)
                  then swap(S)
                else a(S);
{18}    i(exch(S)) ==
                if i(S) = k(S)
                  then i(S)-1
                else i(S);
{19}    k(exch(S)) ==
                if i(S) = k(S)
                  then 1
                else k(S)+1;
```

図４　図１{4},{5},{9}～{14}の従来の記述

2)　状態出力関数の改善

状態出力関数は，状態だけを引数に持つ１引数関数であった．このため，配列は図１{12}{15}のように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(整数以外の添字を持つテーブルを表わす場合，以前は２つの配列を用いて書き手がその対応を管理する必要があった)．状態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．

```
{20}    a(exch(S),j) ==
                if a(S,k(S)) > a(S,k(S)+1)
                  then swap(S,j)
                else a(S,j);
{21}    swap(S,j) ==
                if j = k(S)
                  then a(S,k(S)+1)
                else if j = k(S)+1
                then a(S,k(S))
              else a(S,j);
```

図5　図1{12},{15}の要素ごとの記述

3)　基本データタイプ，基本関数の改善

配列のデータタイプ名，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データタイプ名・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．また，構造体に関する基本データタイプ，基本関数が不足していたため，構造体を用いた記述が煩雑であった．基本データタイプの追加，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)　処理方法の改善

分割コンパイルができ，書き手が既に定義済みである抽象的順序機械を，新しいプログラムで用いることが可能である．

4．変換の際の問題点

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．

ここでは，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点について述べる．

4．1変換の問題点

抽象的順序機械は，状態遷移後の状態出力関数の値を状態遷移前の状態出力関数の値を用いて記述する．プログラムの意味から，状態遷移が生じる際は，その状態遷移により値が更新される状態出力関数はすべて同時に評価しなければならない．このため，各状態出力関数に対して，値の保存用と更新用の２つの領域を用意し，状態遷移の前にすべての値をコピーしてから，状態遷移後の値を計算してやれば用意に実行可能である．

しかし，この方法ではループの内部で頻繁に実行される部分(状態遷移)で不必要なコピーが多数存在する可能性が有り，実行効率が低下することが考えられる．また，配列・ファイル等コピーを作ることに無理のあるデータタイプが存在するという問題点もある．また，配列・ファイル等の処理では，状態変化をもたらす基本関数（配列要素の更新，ファイルからの読み込み等）が存在するため，部分項の実行順序によって値が異なる．例えば，図1{15}は，部分項を右側から計算する場合と左側から計算する場合で値が異なる．プログラムの意味を変えずに効率良く実行するためには，公理および公理の部分項の実行順序を正しく決定してやる必要がある．

4．2問題点の考察

問題点を明確にするために，定式化を行う．

項ｔを有向木で表わしたものをtree(ｔ)=(Ｖ,Ｅ)とする．ｖ　Ｖである各頂点は，ラベルとして関数名または変数名を持ち，それをlabel(ｖ)とする．label(ｖ)として，関数名を持つものの子供のすべて，およびIF関数を持つものの第１子を必須子と呼び，必須子と必須子をつなぐ枝を必須枝と呼ぶ．ｖ　Ｖから必須枝のみを通って訪問できる頂点の集合を，ｖの必須頂点集合という．tree(ｔ)の根をｒとすると項ｔの値を求めるためには，ｒの必須頂点集合に含まれるすべての頂点の値を求めなければならない．

今，状態遷移関数next，状態出力関数$a_1(S),\cdots,a_n(S)$に対して，

$a_1(next(S)) == t_1$; ……

$a_n(next(S)) == t_n$;

の公理が存在するとする．このとき，新たな項として

$T = [t_1, \cdots, t_n]$ を構成する（[] は並びを表わす）．状態遷移後の各状態出力関数の値を求めることは，項Tの値を求めることである．

各 t_i にはIF関数はないとし，tree(T)の根をRとすると，全頂点がRの必須頂点集合に含まれる．次に，tree(T)の同一の部分項を表わす部分木をすべて一つにまとめた有向アサイクリックグラフをdag(T)とする．

大域化する必要のあるデータタイプをdとするとき，公理を正しく実行する順序を見つけることは，dag(T)において，ラベルがデータタイプを表わすと考えた場合，one pebbling problem[5]であり，一般にはNP-完全である．

大域化するデータタイプに対して，正しく実行できる実行順序が存在する場合について考える．このとき，次のような方法が考えられる．

1) 同時実行される公理は，テキスト上で先に現われる順に優先し，部分項は右優先(IF関数を除く)をとる．

この場合，図1{15}のswapは正しく実行できるが，図6の場合は正しく実行できない({22}の実行により，配列aのi番目の要素が更新されてしまうため)．

```
{22}   a(next(S)) == aput(a(S),i(S),aget(b(S),i(S)));
{23}   b(next(S)) == aput(b(S),i(S),aget(a(S),i(S)));
```

図6　2つの配列(a,b)間での要素の交換

2) 大域化されたデータタイプの更新と，参照を一度の状態遷移で書くことを禁じる．この場合は，プログラムの書き手に陽に値の退避を書くことを強制する．

これらは処理系にとっては有利であるが，プログラムの書き手の立場から納得のいけるものではない．

3) 1)2)とは方針が異なるが，副作用を生じるデータタイプの代わりにリストを用いる(リストには，内容を更新する演算がないため，副作用が生じない)．この場合は，ガーベジコレクションを行う必要がある．

今回は，2)と同様の方法を採用することにする．まず，2)の方法で大域化されたデータタイプdの値が正しく求まる十分条件①を考える．

① dag(T)において，値のデータタイプとしてdを持つ頂点はすべて一つの有向道上にあり，各頂点の親は，一つしかない．このとき，各頂点を葉に近い方から順に $v_1, v_2, \cdots, v_n$ とすると，v_n はtree(t_i)の根であり，label(v_i)＝a_iが成立する．

これを少々改良して，v_1だけは複数の親を持つことを許す．このとき，次の②が必要である．

② v_1が複数の親を持つ場合，v_2以外の各親uの実行はv_2よりも前である．

処理としては，②を陽に示すためにv_2から各uに有向枝を付ける(図7②の点線の有向枝)．このグラフをdag(T')とする．①より各uの子孫にv_i(2<i<n)は含まれないため，②で付加した枝は，dag(T')において，先行枝または交差枝になる．よって，dag(T')の根からトポロジカルソートで得られるすべての実行順序でデータタイプdの値は正しく計算できる．

ここでは，dag(T')にIF関数がないことを前提としたが，IF関数がある場合(①は少々異なる)，uがRの必須頂点集合に含まれない場合(IF関数の分岐によっては到達しない頂点であるとき)は，無駄な計算を行う等の不都合はある．決定表を用いることによりこれを除去することもできる．

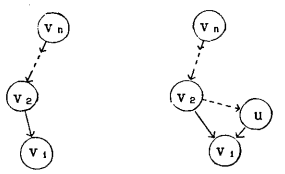

図7　十分条件①,②を表わすグラフ

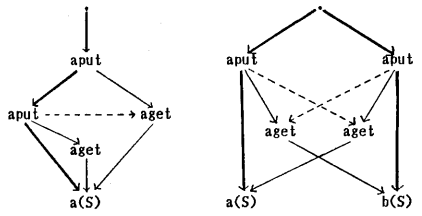

図8　配列の参照・更新の依存関係

5．処理系の概要

処理系は，構文解析部，決定表作成部，最適化部および目的プログラム生成部の４つの部分から構成される．以下，各部の説明を行う．

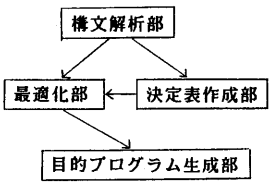

図９　処理系の構成図

５．１構文解析部

抽象的順序機械のプログラムを構文解析し，内部テーブル，および各定義関数の右辺の木表現を作成する．この部分は，yacc，lexを利用して作成を行っている．

５．２決定表作成部

状態遷移ごとに，状態出力関数の値が更新される条件と更新後の値の対応表を作成する．この対応表により，状態出力関数の公理をIF関数のない形に変換できる．IF関数のない形に変換することで，目的プログラムに対する各種の最適化が非常に効率良く行えるが，条件分岐による場合の数が指数的に増加するため，処理系自体の実行効率は大きく低下することになる．この部分がどの程度利用できるかについては，実際に実行可能なレベルで記述された抽象的順序機械のプログラムがあまりにも乏しく，現在検討中である．図10に図４{17}～{19}の決定表による変換を示す．

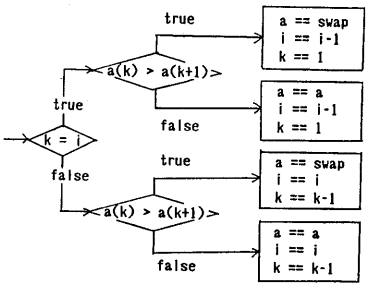

図10　決定表利用後の状態遷移

５．３最適化部

目的プログラムの実効効率を向上させる各種の最適化を行う．各定義関数の右辺の木表現をDAG表現に変換し，目的プログラム生成用の各種テーブルを構成する．ここで行う最適化の手法は，文献[6]により述べられている．以下，４で述べた以外の最適化について処理順に説明する．

1)　補助関数の展開

補助関数を書き換える．この処理は，大域化されたデータタイプの依存関係を認識するために必要である．また，公理の右辺が大きくなり共通部分項が増え重複計算の除去が行えることがある．

2)　依存関係の認識

状態遷移後の状態出力関数の値を計算する際に，大域化されていないデータタイプを値として持つ状態出力関数に対しては，計算前に参照用の領域に値をコピーしておくが，公理の実行順序によっては，その値の計算以降で状態遷移前の値を参照してやる必要がないとき，または公理の右辺に状態遷移前の値が現われていないならば，参照用の領域に値をコピーしてやる必要はない(図１では，a,i,kまたはa,k,iの順で実行すればコピーの必要はない)．ここでは，各状態出力関数の依存関係を認識して，公理の実行順序，およびコピーの必要な状態出力関数の決定を行う．

1)　共通部分項の検出

一つの定義関数の右辺，または同時実行される状態出力関数の右辺中の共通部分項に対しては，同一の計算を再度行わないように，一度求めた値を保存しておき，再び必要となった際には，その値を参照する．ここでは，定義関数右辺の木表現をDAG表現に変換する．この処理は，データタイプの大域化のためには不可欠のものである．

3)　tail recursionの検出

ループ関数ｆの公理の右辺にｆが現われるとき，右辺のｆの外側にかかる関数はすべてIF関数であり，ｆまたは各IF関数は親のIF関数の第２または第３引数に現われているならば，ｆはtail recursiveであるという．このときループ関数は目的プログラムのうえでは

単純な繰返し文に変換できる．また，状態出力関数ｆにおいて上の条件が成立する時は，その分岐においてｆの値は状態遷移前と同じであり，目的プログラムでは何も行う必要がない．ここでは，tail recursionの検出を行う．

５．４目的プログラム生成部

内部テーブル，および各定義関数の右辺のDAG表現からＣ言語の目的プログラムを生成する．

抽象的順序機械は，その意味から，状態遷移が生じるごとに，各状態遷移関数の値がすべて決まる必要がある．よって，引数の遅延評価を行う必要はなく，値呼びで関数値を求めるプログラムを生成する．以下，どのようなプログラムを生成するかについて述べる．

抽象的順序機械における状態出力関数は，一般の手続き型言語の変数に相当する．目的プログラムでは，各状態出力関数に対して，値の参照用と更新用の２つの領域を静的に割り当てる．そして，各関数を手続きの形で出力する．主手続きに相当する関数は，コンパイル時にコンパイラの引数として指定される．

プログラムは２本のスタックを使用し，１つは各手続きでの項の計算用に，もう一つは各手続きが使用する領域用(フレームと呼ぶ)に用いられる．フレームの中には，共通部分項の保存用の領域，値が計算済みかどうかを示すフラグの領域等が取られる．

６．あとがき

現段階では，実際に実行可能なレベルで記述された抽象的順序機械のプログラムがあまりにも乏しく，プログラム経験の蓄積が要求される．また，処理系に関しても，まだまだ解決すべき問題点があり，部分的にしか作成が行われていない．特に，実行順序を決める問題では，静的な解析だけでは解決することは困難であり，動的にも処理を施す必要があると思われる．

【謝辞】

日頃御指導頂く嵩忠雄教授，適切な御助言を戴いた嵩研究室関浩之氏，ならびに本処理系の作成を御協力戴いている稲田良造氏に感謝致します．

【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-SW-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).

[5]Sethi.R ："Pebble Games for Studing Storage Sharing", Theoretical Computer Sience 19, pp.69-84,(1982).

[6]関，井上，谷口，嵩：”関数型言語ＡＳＬ／Ｆとその最適化コンパイラ”，信学論（Ｄ），J67-D，4，pp.458-465 (1984).