ソフトウェア基礎論　17－4
(1986. 7. 3)

入出力構造に基づく変換プログラムの合成法
A generation method of conversion programs based on input and output data structures

森本　真一
Shin-ichi Morimoto
(日本電気株式会社　マイクロコンピュータ・ソフトウェア開発本部)
NEC Corporation, Microcomputer Software Development Laboratory

あらまし　本発表では、入出力データの構造が属性文法によって記述されている場合に、そのデータ間の変換プログラムを生成する新しい方法を述べる。この方法はデータの属性値からその構文構造を求める演算によって出力データの構造を入力データの属性値から導出するものである。また入出力データの属性が代数的仕様で記述されている時に入出力の変換を効率的に行なうアルゴリズムを述べる。

Abstract　This report shows a new method to generate conversion programs between input and output data, those structures are specified by attribute grammars. This method uses a reverse operations of attribute evaluations, which get syntax structures from their attribute values. By these operations, output structures are obtained from input attribute values. An efficient algorithm for algebraic data specifications is also shown.

1．はじめに

従来からプログラム言語に対する各種の定式化と、それに基づくコンパイラの各フェーズの自動生成の研究が行なわれてきた。例えば、プログラム言語の構文の構造を文脈自由文法（ＣＦＧ）によって記述する事により、コンパイラの構文解析部の自動生成が可能になった。また意味構造の属性文法による記述に基づいて意味解析部が自動生成できる様になった。そこで現在では入出力の変換の記述からコード生成部を自動生成する研究が行なわれている。

入出力間の変換を記述する主な方法として、従来は次の２つがあった。

1）各入力構造に対応する出力構造を入力の属性として定義する。［1］

2）各入力構造と対応する出力構造をテンプレートとして定義する。［2］

ここでは第三の方法として次の方法を示す。

3）出力の各意味規則（属性の求め方を示す規則）に対して、その逆関数（構文規則の左辺の記号の属性値から右辺の構文記号とその属性値を求める関数）を定義し、その逆関数に基づき出力全体の属性値（これは入力データの意味解析の結果として求まる）から出力の構造を求める。

本稿では、第二節で本方法の説明を行ない、第三節でこの変換の例を示し第四節でこの方法と他の方法の比較を行なう。第五節では入出力の属性が代数的仕様で与えられた場合に変換を効率的に行なう方法を示し、第六節で具体的なアルゴリズムを示す。

2．変換方法

本稿では次の条件を満たす変換プログラムを対象とする。

1）入出力データの構造は属性文法によって記述されている。
2）入出力データの意味は各開始記号の合成属性として与えられる。
3）プログラムは　入力データを同じ意味の出力データに変換する。

この時、出力の各構文規則に対して次の関数を定義する。

a）Selecter
　左辺の構文記号の属性値を真偽値に対応させる（その構文規則の構造を持ち、その属性値を算出する出力構造が存在すれば真）

b）Decomposer
　この構文規則のselecterの値が真の場合に、左辺の非終端記号の属性値を右辺の各構文記号の(合成)属性の値に対応させる。この時、与えられた非終端記号の属性値と、それからdecomposerによって求められた右辺の構文記号の属性値は、その構文規則の意味規則を満たす。

例　出力の構文規則の１つ（ｒとおく）を
　　ｒ０ ： ｒ１ ｒ２ ・・・ ｒｎ　とし
　　ｒ０の合成属性を算出する意味規則をｒ０Ｓ
　　ｒのselecterをＳｅｌ、
　　右辺のｉ番目の記号に対するdecomposerを
　　　Ｄｅｃｉ（１≦ｉ≦ｎ）とする時
ｒｏの相続属性,合成属性の値をＳ０,Ｉ０とおくと
　Ｓｅｌ(Ｓ０,Ｉ０) ＝真
　Ｓ０ ＝ ｒ０Ｓ(Ｉ０, Ｄｅｃ１(Ｓ０,Ｉ０),
　　…,Ｄｅｃｎ(Ｓ０,Ｉ０))
　　が成立する。

これらの関数によって次の様に変換を行なう。

1）入力の属性文法に基づき、入力データの解析を行ない、入力の開始記号の合成属性の値を求める。
2）1）で求めた値と出力の相続属性の値からselecterが真になる構文規則を求める。
3）2）で求めた構文規則のdecomposerから右辺の各構文記号の(合成)属性の値を求める（相続属性の値は意味規則から求める）
4）2）で求めた構文規則の右辺の各構文記号に対して、3）で求めた属性値に基づき2）3）と同様にしてその要素の下部構造とその属性値を求める。
　これを終端記号になるまで繰り返す。

3．変換例

ここでは例として、２進小数を同じ値の３進小数に変換する問題を考える。この場合、入力データ（２進小数列）と出力データ（３進小数列）の構造は次の属性文法で与えられる。［3］

◎入力構造
　非終端記号：Ｓ，Ａ，Ｂ
　終端記号　：０，１，．
　属性　：Ｉｎｈ(Ｓ)＝φ　Ｓｙｎ(Ｓ)＝{ｖａｌ}
　　Ｉｎｈ(Ａ)＝{ｐｏｓ}　Ｓｙｎ(Ａ)＝{ｖａｌ}
　　Ｉｎｈ(Ｂ)＝{ｐｏｓ}　Ｓｙｎ(Ｂ)＝{ｖａｌ}
　構文規則および意味規則
　　１．Ｓ ： ． Ａ
　　Ｓ．ｖａｌ＝Ａ．ｖａｌ　Ａ．ｐｏｓ＝１
　　２．Ａ１： Ｂ Ａ２
　　Ａ１．ｖａｌ＝Ｂ．ｖａｌ＋Ａ２．ｖａｌ
　　Ａ１．ｐｏｓ＝Ａ２．ｐｏｓ＋１
　　Ｂ．ｐｏｓ＝Ａ１．ｐｏｓ
　　３．Ａ ： Ｂ
　　Ａ．ｖａｌ＝Ｂ．ｖａｌ
　　Ｂ．ｐｏｓ＝Ａ．ｐｏｓ
　　４．Ｂ ： ０
　　Ｂ．ｖａｌ＝０
　　５．Ｂ ： １
　　Ｂ．ｖａｌ＝２＊＊(－Ｂ．ｐｏｓ)

◎出力構造
　非終端記号：Ｓ’，Ａ’，Ｂ’
　終端記号　：０，１，２，．
　属性　：Ｉｎｈ(Ｓ’)＝φ　Ｓｙｎ(Ｓ’)＝{ｖａｌ}
　　Ｉｎｈ(Ａ’)＝{ｐｏｓ}　Ｓｙｎ(Ａ’)＝{ｖａｌ}
　　Ｉｎｈ(Ｂ’)＝{ｐｏｓ}　Ｓｙｎ(Ｂ’)＝{ｖａｌ}
　構文規則および意味規則
　　１’．Ｓ’： ． Ａ’
　　Ｓ’．ｖａｌ＝Ａ’．ｖａｌ　Ａ’．ｐｏｓ＝１
　　２’．Ａ’１： Ｂ’ Ａ’２
　　Ａ’１．ｖａｌ＝Ｂ’．ｖａｌ＋Ａ’２．ｖａｌ
　　Ａ’１．ｐｏｓ＝Ａ’２．ｐｏｓ＋１
　　Ｂ’．ｐｏｓ＝Ａ’１．ｐｏｓ
　　３’．Ａ’： Ｂ’
　　Ａ’．ｖａｌ＝Ｂ’．ｖａｌ
　　Ｂ’．ｐｏｓ＝Ａ’．ｐｏｓ
　　４’．Ｂ’： ０
　　Ｂ’．ｖａｌ＝０
　　５’．Ｂ’： １
　　Ｂ’．ｖａｌ＝３＊＊(－Ｂ’．ｐｏｓ)
　　６’．Ｂ’： ２
　　Ｂ’．ｖａｌ＝２＊３＊＊(－Ｂ’．ｐｏｓ)

この時、出力の各構文規則（1'～6'）に対して次のselecterとdecomposerを定義する。

```
1'  Sel (S'.val) = True
    Dec1(S'.val) = S.val

2'  Sel (A'1.val, A'1.pos)
      =   A'1.val!=0 && A'1.val!=3**(-A'1.pos)
        && A'1.val!=2*3**(-A'1.pos)
    Dec1(A'1.val, A'1.pos)
      = [ A'1.val / 3**(-A'1.pos) ]
        ただし [x]は、x以下の最大の整数を示す
    Dec2(A'1.val, A'1.pos)
      = A'1.val
      - Dec1(A'1.val, A'1.pos)*3**(-A'1.pos)

3'  Sel (A'.val, A'.pos)
      =   A'.val==0 || A'.val==3**(-A'.pos)
        || A'.val==2*3**(-A'.pos)
    Dec1(A'.val, A'.pos)
      =   A'.val / 3**(-A'.pos)

4'  Sel (A'.val, A'.pos) = A.val==0

5'  Sel (A'.val, A'.pos) = A.val==1

6'  Sel (A'.val, A'.pos) = A.val==2
```

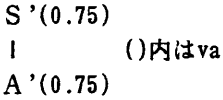
Note

厳密な計算では、S'.valが0でない場合は3'のselecterは常に偽になる（整数以外の有限2進小数は有限3進小数では表現できない）。ここでは通常の計算機で行なわれている様な有限の桁数での計算を想定している。

例えば、入力列が　．１１である場合の変換は次の様に行なわれる。（なお入力列の解析と属性評価の過程は省略する。また、計算の精度は１０＊＊－２とする。）

a）入力を解析して、S.val = 0.75　を得る。

b）この場合、1'のＳｅｌ(0.75)=TRUE、
Ｄｅｃ1(0.75)=0.75だから
この時点での出力構造は図１の様になり、
A.val=0.75　A.pos=1（意味規則による）
となる。



()内はval属性の値を示す。

図１

c）2'のＳｅｌ(0.75, 1)=TRUE、
Ｄｅｃ1(0.75, 1)=2、
Ｄｅｃ2(0.75, 1)=0.08　より
この時点での出力構造は図２の様になり、
B'.val=2　B'.pos=1
A'2.pos=0.08　A'2.val=2　となる。

d）2'のＳｅｌ(0.08, 2)=TRUE、
Ｄｅｃ1(0.08, 2)=0、
Ｄｅｃ2(0.08, 2)=0.08　より
この時点での出力構造は図３の様になり、
B'.val=0　B'.pos=2
A'2.pos=0.08　A'2.val=3　となる。

e）3'のＳｅｌ(0.08, 3)=TRUE、
Ｄｅｃ1(0.08, 3)=2より
この時点での出力構造は図４の様になり、
B'.val=2　B'.pos=3　となる

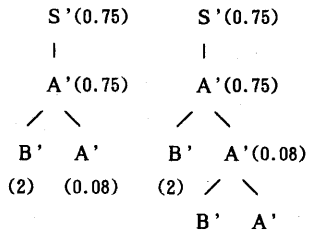


図 2

図 3

図 4

()内はval属性の値を示す。

4．他の方法との比較

1．で述べたように属性文法で表現された入出力データの変換を記述する方法として、従来は次の2つがあった。

1）入力の各構文規則に対応する出力の構造（構文規則と属性の値）を入力の属性として与える。

2）各入力構造とそれに対応する出力構造をテンプレートとして定義する。

本稿で述べた方法は1）2）の方法に対して次の点ですぐれている。

a）入出力構造の記述が対等

1）の方法では、出力の構文や属性はすべて入力の属性として表現される。このため出力の構造は入力ほど明確に記述されていない。そこで例えばコード生成部を形式的に記述する場合の様に出力構造も入力構造と同程度に形式化する必要がある場合には1）の方法では不十分である。しかし本稿の方法では入力と出力の構造は全く同等に属性文法によって記述されるのでこの様な場合にも適している。

b）入力構造と出力構造を直接対応させる必要がない。

2）の方法ではテンプレートを作るために入力構造と出力構造が直接対応している必要がある。　そこで例えば3．で述べた例の様に入力の構文規則と出力の構文規則の間に直接の対応関係がない場合には2）の方法を用いる事はできない。　しかし、この方法では出力構造を入力から直接定義するのではなく、入力の属性値から間接的に定義しているのでこの様な場合にも適用できる。

c）変更が容易である。

1）、2）の方法では入出力の対応を直接記述している。このため、入力や出力の構造の一部は変更された場合は、その構造だけでなく対応関係を記述している部分も変更する必要がある。しかしこの方法では入出力間の対応関係は直接記述されていないので、変更された構造に関する記述のみを書き換えればよく、変更が容易である。例えば3．の例で5進小数に出力する様に変更された場合は、3進小数の構造の記述（と各構文規則のselecterとdecomposer）を5進小数の構造の記述に変更するだけでよい。

5．代数的仕様記述に基づく変換

4で述べた様に、本稿の方法は従来の方法に比べていくつかの点ですぐれているが次の問題点がある。

1）変換が冗長になる事がある。

この方法では、まず入力全体の属性値を求め次にその値に基づいて出力構造を決定している。このため、入力の部分構造と出力の部分構造が直接対応している場合は、全体の属性値を求めてから対応させる事になり変換が冗長になる。

2）記述が冗長になる事がある。

この方法では出力の属性記述から改めてselecter やdecomposer を作成する必要がある。このため出力の各構文要素の属性間の関係が明らかな場合は記述が冗長になる。

例　(infix notation→postfix notation)

◎入力構造

```
exp1 : exp2  OP  exp3
  exp1.val = cons(exp2.val, OP.val
                  , exp3.val);
exp  : NUMBER
     exp.val = NUMBER.val
```

◎出力構造

```
exp1': exp2' exp3' OP'
  exp1'.val=cons(exp2'.val, OP'.val
                 , exp3'.val);
exp' : NUMBER'
  exp'.val = NUMBER'.val
```

この例では入力の第一の構文規則と出力の第一の構文規則が対応し、入力の第二の構文規則と出力の第二の構文規則が対応する。そこで入力の構文規則を認識した時点で対応する出力構造を決定できる。このため本来は入力全体の属性値(exp.val) を計算する必要はないが、4．で述べた方法では入力全体の属性値を求めなければならない。

また入出力の構文規則同士が対応しているので本来はselecterを作成する必要はないが、この方法では入力全体の属性値から出力構造を求めるのでselecterを作成する必要がある。

これらの問題点は、入力の意味規則と出力の意味規則の対応関係を、前もって（コンパイル時に）決定できない事に原因がある。前もって対応関係を決定できれば　入力構造から直接（その属性値を計算せずに）出力構造を決定できるので、selecterやdecomposerも不要になる。対応関係を決定するためには、入力の意味規則と出力のdecomposerの内容をあらかじめ決定できればよい。こうすれば　入力の各構文規則の右辺の要素の属性値に対して（入力の意味規則と出力のdecomposerを解析する事により）対応する出力の構造とその属性値を　前もって決定できる。しかし一般の場合は属性や、その間の関係を表わす意味規則は十分形式化されていないので、意味規則や（その逆変換である）decomposerの内容をあらかじめ決定するのは困難である。

このため、ここでは属性を代数的仕様で表わす事を考える。代数的仕様は抽象データ型を発展させたものである。代数的仕様では各データ型に対して、それを操作する演算子と演算子間の関係が定義されている。この型の値は、これらの演算子によってのみ操作されるので演算子をノードとする木構造で表現できる。これにより属性の値や意味規則を木構造とその変換という形で形式的に扱う事ができ意味規則による属性値の間の対応関係を実行する以前に解析できる。

属性を代数的仕様によって表現した時の変換アルゴリズムの概要は次の様になる。（簡単のため属性はすべて合成属性であり、入力データはトップダウン解析が可能だとする）

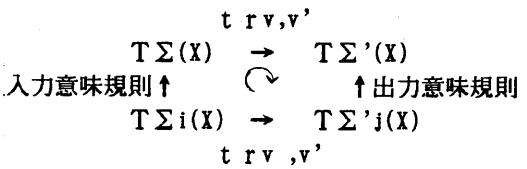
1）入出力の各構文の意味規則に対するselecter, decomposerを求める。

今の場合各属性の値は木構造で表わす事ができる。ここでは、各属性値を以下に述べる代表元（その属性のソートのΣ(X)項 ）の集合で表わし、selecterは代表元と構文規則、decomposerは代表元間の対応関係として表現する。

ソートｓの代表元とは次の性質を持つ
Σs(X)項の集合{t1(x)，　,tn(x)}の元である。
ソートｓの任意の値v に対して、
t(X)のσによる代入結果が vに等しい
代表元t(x)と代入σが一意に定まる。

2）入出力の各構造同士の対応関係を求める。

既に決定している入出力の構文記号（の属性値）間の対応関係から、その記号を左辺とする構文規則同士の対応と、それらの規則の右辺の構文記号（の属性値）間の対応関係を下図を満たす様に求める。



但し

t r v,v' ：左辺の属性値間の対応規則
TΣ(X) ：入力左辺属性の代表元の集合
TΣ'(X) ：出力左辺属性の代表元の集合
TΣi(X) ：入力右辺属性の代表元の集合
TΣ'j(X)：出力右辺属性の代表元の集合
t r v ,v'：右辺の属性値間の対応規則

図　入出力属性の対応関係

3）2）の対応関係に基づきparserを作成する。

仮定により入力データ構造はトップダウン解析が可能だから、このparserでは入力をトップダウンに構文解析を行ない、各構文記号を認識する毎に、2）で決定した対応関係に基づき対応する出力構造を決定する。

この方法では次の点が問題になる。

a）各意味規則のdecomposerを代表元間の対応として、常に表現できるか

これは常に成立するわけではない。これが成立するための意味規則や各ソートの操作に対する書き換え規則に関する十分条件はいくつか考えられるが、ここでは簡単のためにこの条件の成立は仮定する。

b）入出力構造の対応から下部構造の対応関係が常に決定できるか

下部構造（構文記号）同士が常に１：１に、順序も込めて対応するわけではない。しかし対応しない場合は２）の段階で判定できる。そこでparserとしては、入出力の構造が直接対応しなくなれば　それ以降は入力の解析と属性値の計算のみを行ない、入力の属性値が求まった後で　その値から（出力のselecterとdecomposerによって）出力構造を求める様にする。対応関係が一意に決定できない場合も同様の処理を行なう。

6．アルゴリズム

ここでは5．で述べた変換方法を　具体的に述べる。なお代数的仕様記述に関する基本的な定義は省略する（[4]を参照）。

定義

ソートｓの代表元とは次の性質を持つ
Σs(X)項の集合 {t1(x), ,tn(x)}の元である。
for all v(ソートｓの値(を表わす木構造))
代表元ti(x), 代入σ　s.t.
ti(X)のσによる代入結果が vに等しい。

Note

ｔi(X)をソートｓの代表元とする時、
ｔi(X)から代入σによって得られる値をｔσ、
ｔi(X)から代入によって得られるｓの値全体の集合を[ｔi]で表わす。

定義

ｒ　　：構文規則
Tr(X)　：ｒの左辺の属性値の代表元の集合
ｆ　　：ｒに対応する意味規則
ｒi　　：構文規則の右辺の i番目の構文記号
Tri(X)　：ｒi の属性値の代表元の集合
とする時
ｒのselecter(Ｓｅｌ), decomposer(Ｄｅｃi)
Ｓｅｌ　：　Tr(X) → {T、F}
Ｄｅｃi：　Tr(X) →　Tri(X)
とは次の条件を満たす関数である。
for all ｔ in Tr(X)
Ｓｅｌ(ｔ) ＝ T
iff ｆの結果が[ｔ]に含まれる様なｒによる構文構造が存在する。
Ｄｅｃi(ｔ) ＝ ti
iff　for all σ(代入)
ｆ(t1σ,　,tnσ) ＝ tσ

定義

ｒ　　：構文規則
Ｄｅｃi：ｒのdecomposer　　とする時
Var(r,i)＝Ｄｅｃi中の変数のソートの集合

定義

ｒ　：入力構文規則
ｒ’：出力構文規則　とする時
Cor(r,r',i)
＝{j| Var(r,i)⊆Var(r',j)
Var(r,k)⊈Var(r',j)　(all k<i)}
Eff(r,r',i)
＝{j| Var(r,i)∩Var(r',j)≠φ}

定義

ｖ　　：入力の非終端記号
Tv(X)　：ｖの属性値の代表元の集合
ｖ’　　：入力の非終端記号
Tv'(X)：ｖ’の属性値の代表元の集合
とする時
ｔｒv,v’：Tv(X)→Tv'(X) は
次の方式で定義される対応規則である。
1）ｔｒs,s’ ＝ id（恒等変換）
但し s,s’は入出力の開始記号
2）ｔｒv,v’ が定義されている場合に
for all ｔ in Tv(X)
s.t.　Ｓｅｌ(ｔ) ＝ T
Ｓｅｌ(ｔｒv,v'(ｔ)) ＝ T
を満たす規則ｒ，ｒ’が存在する。
この時
ｊ ＝ min Cor(ｒ,ｒ’,ｉ)　とすると
ｔｒr_i,r_j' がまだ定義されていなければ
ｔｒr_i,r_j' を　Ｄｅｃi(t)に
Ｄｅｃj(ｔｒv,v'(ｆ(t)))を
対応させるものと定義する。

定義

parserは　以下で定義される副プログラムｐを順次呼出す事により定義される。
ｐ　(ｖ, ｖ’, ｏｕｔP, ｖａｌP)
ｖ　:入力構文記号
ｖ’:出力構文記号 ＋ {UNDEF}
ｏｕｔP　:ｖ’の構造を出力するかを示す
ｖａｌP　:ｖ の属性値を返すかを示す
{
1）ｖが非終端記号の場合
ｖの構造に対応する構文規則をｒとする。
1-a)ｖ’≠ UNDEF、ｔｒv,v’が定義され、
for all ｔ in Tv(X)
s.t.　Ｓｅｌ(ｔ) ＝ T
Ｓｅｌ(ｔｒv,v'(ｔ)) ＝ Tとなる
ｖ’ の構文規則ｒ’が存在する時
ｒ　の右辺の記号をｒ1，　,ｒj
ｒ’の右辺の記号をｒ’1，,ｒ’k
とおく。

for (i=1; i<=j; ++i)
{ P(ri,o,oP, vP);
但し
o = UNDEF
(if Cor(r,r',i)==φ || oP==F)
= r'm (m=min Cor(r,r',i))
oP=T
(if Cor(r,r',i)!=φ &&
r'm以前の記号はすべて出力済)
=F (otherwise)
vP=T (if #Eff(r,r',i)>1)
=F (otherwise)
oP==Fの場合に、Cor(r,r',i) の元に対応する記号で、出力していないものがあれば規則に現われる順に出力する。
}
outP==Tの場合に、出力していないr' の記号があればvの属性値を計算し、trv,v'によりv' の属性値に変換し、それから (selecter,decomposerにより) v' に対応する出力構造を出力する。
valP==Tの場合はvの属性値を返す。

1-b)それ以外の場合
r の右辺の記号をr1, ,rj とおく。
for (i=1; i<=j; ++i)
P(ri,UNDEF,F, T);
各riの属性値から、vの属性値を計算して返す。

2) vが終端記号の場合
vの属性値を返す。

}

まとめ

属性文法で構造が記述されたデータ間の変換プログラムを生成する新しい方法を述べた。この方法では入出力データの構造を対等に記述できるので、コンパイラ (特にフロントエンド) は従来の方法よりも自然に記述できる。今後はこの属性が代数的仕様で記述されている場合のアルゴリズムの改良およびこの方式の実現を行ないたい。

参考文献

[1]P.M.Lewis, D.J.Rosenkrantz, R.E.Sterns:
"Attributed Translations",
J.Comput.Syst.Sci, Vol.9, pp279-307 (1974)

[2]渡辺:
意味情報の推論的照合による言語変換方式の提案,
情報処理学会ソフトウェア基礎論研究会,
16-9, (1986)

[3]片山:属性文法型計算モデル,
情報処理, Vol.24, No.2, pp.147-155 (1983)

[4]稲垣:抽象データ型の概念と仕様記述法
情報処理, Vol.27, No.2, pp.120-128 (1986)