正規右辺属性文法の一提案

中田育男† 山下義行†

プログラム言語の構文規則は，文脈自由文法で記述するより，生成規則の右辺に正規表現を許す正規右辺文法（regular right part grammar）を使った方が，一般に簡潔で分かりやすいものになる．しかし，意味規則については，文脈自由文法についての属性文法にあたるものを正規右辺文法に拡張した，いわゆる正規右辺属性文法（regular right part attribute grammar）を定義するのは簡単ではない．その理由は，たとえば生成規則の右辺に0回以上の繰り返しを意味する正規表現がある場合は，それに対応する記号列がプログラム中に現れる回数が一定でなく，そのそれぞれに対応した意味規則を記述するのが困難であるからである．正規右辺文法に対する意味規則として従来提案されている記述法には，典型的なパターンを定めたもの，意味規則にも正規表現を導入したもの，生成規則中に属性評価規則を埋め込むものなどがあるが，いずれの方法にも欠点があった．本論文では，それらの欠点のない，文脈自由文法に対する属性文法の自然な拡張としての正規右辺属性文法の提案をする．それは，繰り返し型や選択型の正規表現の属性名と属性評価式の表現法を与えるものであり，繰り返し型の正規表現に対してはその属性値を漸化式の形で与えるものである．また，そのような正規右辺属性文法に対して構文解析と同時に属性評価をする方法の概略を述べる．

A Proposal of Regular Right Part Attribute Grammars

IKUO NAKATA † and YOSHIYUKI YAMASHITA †

A regular right part grammar (RRPG), or an extended context-free grammar, is a context-free grammar in which regular expressions of grammar symbols are allowed in the right-hand sides of productions. RRPGs are useful for representing the syntax of programming languages naturally and concisely. However it is not easy to define a regular right part attribute grammar as an extension of an attribute grammar over an ordinary context-free grammar. The main problem is the difficulty to express semantic rules for a regular expression because it can produce indefinite number of syntactic constructs. There have been several proposals to overcome this difficulty, for example, the restriction of patterns of attribute propagations, the introduction of semantic regular expressions, and semantic rules embedded in syntactic rules. They have not been completely successful. In this paper we propose a new notation for a regular right part attribute grammar which is a natural extension of an attribute grammar. The attribute values for repeated occurrences of a syntactic construct are expressed by a recurrence formula. The outlines of attribute evaluation methods for an L-attribute or an LR-attribute grammar are also explained.

1. はじめに

プログラム言語の構文規則は，文脈自由文法で記述するより，生成規則の右辺に正規表現を許す正規右辺文法（regular right part grammar）を使った方が，一般に簡潔で分かりやすいものになる．しかし，意味規則については，文脈自由文法についての属性文法にあたるものを正規右辺文法に拡張した，いわゆる正規右辺属性文法(regular right part attribute grammar)を定義するのは簡単ではない．その理由は，たとえば

$$A \rightarrow \alpha\{\beta\}\gamma$$

なる生成規則について(ここで，$\{\beta\}$ は β の0回以上の繰り返しを意味する正規表現とする)，A に還元される終端記号列の中には β に対応する記号列が複数個現れ得るのに対して，そのそれぞれに対応した意味規則を記述するのが困難であるからである．

正規右辺文法に対する意味規則として，従来提案されている記述法には，(1) 複数個の β での属性値の伝播の典型的なパターンを決めてそのパターンに対する記述法を定めたもの[1),2)]，(2) 構文規則における正規表現と対応させて意味規則にも正規表現を導入したもの[3)]，(3) 生成規則中に属性評価規則を埋め込むもの[4)~6)]，などがあるが，(1)は分かりやすいがパターン

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

に制限がある，(2) は必ずしも分かりやすい表現とならない，(3) は分かりやすいが表現が手順的になり，単一代入性から外れるところがある，といった欠点がある．

属性文法の例題としてよく使われる2進小数の属性文法(**図1**)を使ってこのことを説明する．ここでは属性評価規則は「<」と「>」で囲むことにする．この文法を正規右辺文法で書けば，たとえば次のようになる．

$F \rightarrow .\{B\}$

$B \rightarrow 0|1$

これに (2) の方式で意味規則を付けると，たとえば

$F \rightarrow .\{B\}$ $\langle F.\mathrm{val}=0\{+B.\mathrm{val}\}\rangle$

となる．この $\{+B.\mathrm{val}\}$ は構文規則の $\{B\}$ に対応する意味規則での正規表現である．この表現はこの例では見やすいと言えるが，意味規則で使える正規表現には制限があるし，$\{B\}$ という繰り返しや，$(A|B)$ という選択が一つの右辺の中で何か所かに現れると，それらを区別する記述が煩雑になり，見やすくなくなるという問題がある．

その問題を解決するためには，構文規則の中に意味規則を埋め込むことが考えられるが，(3)の方法では，たとえば

$F \rightarrow .\langle p:=0;v:=0\rangle\{\langle p:=p+1;B.\mathrm{pos}=p\rangle B \langle v:=v+B.\mathrm{val}\rangle\}\langle F.\mathrm{val}=v\rangle$

となる．p や v は局所変数である．構文要素と意味規則との対応は分かりやすくなり，記述の自由度も高いが，単一代入性が失われたり，意味規則が長くなると構文規則が読みにくくなる問題がある．

(1) の文献 1) の方法も構文規則の中に意味規則を埋め込むものであるが，それによれば，F.val を求める意味規則はたとえば（記法は本論文にあわせて少し変更してある）

$F\langle \mathrm{sum}\{\mathrm{val}\}\rangle \rightarrow .\{B\langle \mathrm{val}\rangle\}$

となる．ここで，{val} は各 B の val を一つのリストにまとめたものを意味する．sum はリストの各要素の値の合計を求める関数である．この論文では，このリストにまとめる表記法を含めて5種類の表記法が提案されている．その範囲で表現できるものは簡潔に表現できるが，たとえば，今の例の B.pos のように値が変化していくものは一つの生成規則の中では表現できないという問題がある．

$F \rightarrow . L$	<F.val=L.val; L.pos=1>
$L \rightarrow \varepsilon$	<L.val=0>
$L_0 \rightarrow B\ L_1$	<L_0.val=B.val+L_1.val; L_1.pos=L_0.pos+1; B.pos=L_0.pos>
$B \rightarrow 0$	<B.val=0>
$B \rightarrow 1$	<B.val=$2^{-B.pos}$>

図1 2進小数の属性文法
Fig. 1 An attribute grammar of binary fractions.

本論文では，上記のような欠点のない，文脈自由文法に対する属性文法の自然な拡張としての正規右辺属性文法の提案をし，それに対して構文解析と同時に属性評価をする方法の概略を述べる．その表記法は，意味規則が長くなってもあまり読みにくくならないように生成規則と意味規則を分離するものであり，分離しても両者の対応が明確になるように生成規則中に属性名を付記するようにしたものである．また，正規表現の繰り返しに対応する意味規則は漸化式で表現する．

2. 正規右辺属性文法

正規右辺文法における生成規則

$A \rightarrow \alpha\{\beta\}\gamma$

の $\{\beta\}$ は，たとえば，

$A \rightarrow \alpha B\gamma$

$B \rightarrow \varepsilon$

$B \rightarrow B\beta$

の略記法と考えられるから，この B の相続属性と合成属性にあたるものが $\{\beta\}$ に付随すると考えるのは自然である．ところで，属性文法では非終端記号については，たとえば

A.val

の形で属性を表現するが，$\{\beta\}$ については A に当たる名前がないので，表現しにくい．そこで，$\{\beta\}$ の相続属性名 i と合成属性名 s を次の形で書くことにする．

$$A \rightarrow \alpha\{\beta\}^{i}_{s}\gamma \qquad (1)$$

一般には，相続属性や合成属性は複数個ありうるから

$$A \rightarrow \alpha\{\beta\}^{i_1,i_2,\cdots,i_m}_{s_1,s_2,\cdots,s_n}\gamma \qquad (2)$$

の形になるが，以後は簡単のため，誤解のない限り，そのとき注目している属性の名前だけを式（1）の形に表記することにする．

選択型の正規表現についても同様に考えられる．たとえば，

$A \rightarrow \alpha(\beta_1|\beta_2|...|\beta_n)\gamma$

なる生成規則において，選択型の $(\beta_1|\beta_2|...|\beta_n)$ の属性を

$$A \rightarrow \alpha(\beta_1|\beta_2|...|\beta_n)^{i}_{s}\gamma \qquad (3)$$

の形で書くことにする．

なお，これらの生成規則において，α，β，γ は，それ自身正規表現であり，繰り返し型や選択型の表現を含むことがありうる．

ところで，式(1)における β は複数回(0回を含む)出現し得るものを表しているから，β が n 回出現した

ときには，式（1）とそれに付随する意味規則で

$$A\rightarrow\alpha(\beta)_{s^1}^{i^1}\cdots(\beta)_{s^n}^{i^n}\gamma \qquad (4)$$

に相当する意味規則が表現できている必要がある．ここで，i^j, s^j は β の j 回目の出現に対応する i, s の値を表す．

通常の属性文法においては，意味規則は一般に次の形である．

$$x=f(y_1, y_2, \cdots, y_m)$$

ここで，x は，対応する生成規則の左辺の非終端記号 A の合成属性か，右辺の非終端記号の相続属性であり，y_i は，左辺の非終端記号 A の相続属性か，右辺の非終端記号の合成属性である．我々の場合は，生成規則の右辺に正規表現があるから，その属性に関する意味規則も，その生成規則に対応する意味規則の中で書ける必要がある．すなわち，上記の x や y_i として，繰り返し型や選択型の正規表現の合成属性や相続属性を許す必要がある．ただし，繰り返し型や選択型の正規表現の属性については以下のような制限を付けることにする．

（a） 繰り返し型や選択型の正規表現の中の合成属性は，その正規表現の外の属性評価には使えない．

（b） 繰り返し型の正規表現の合成属性や相続属性を求める評価関数は，漸化式型のものに限られる．

（c） 選択型の正規表現の合成属性を求める評価関数は，選択型のものに限られる．

ここで，正規表現の「中」，「外」は，その正規表現の構成要素であるかないかを意味する．

（a）の制限は，たとえば，式（1）の γ の中の非終端記号の属性を求める関数の引数として β の中の非終端記号の合成属性が現れてはならないということである．この場合，β に対応する記号列は複数回出現しうるから，その引数値が複数個の出現の中のどの値であるかが特定できないからである．選択型についても同様に，複数の可能性の中のどれが出現するか特定できないからである．いずれの場合も，正規表現の中の合成属性はその正規表現自身の合成属性（式（1）の場合の s）に反映して，その結果をその外で参照するのが自然である．

（b）の制限は，次のように考えられる．たとえば，式（4）の $\{\beta\}$ の合成属性は $s^0, s^1, s^2, \ldots$ と繰り返し計算されるが，繰り返し計算法の多くは，初期値の計算法と，一つ前の値から次の値を求める関数で与えることができる．それが漸化式である．それを，通常の文脈自由文法の属性文法から導き出すには次のように考えればよい．

一般に，$A\rightarrow\alpha\{\beta\}\gamma$ という生成規則を使った場合は，

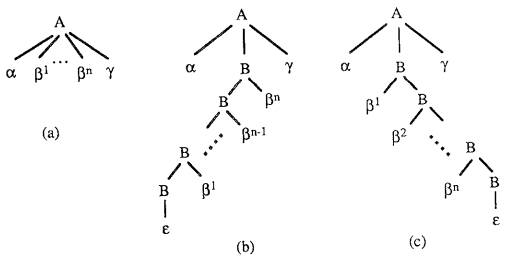


図2 $A\rightarrow\alpha\{\beta\}\gamma$ による導出木と文脈自由文法への変換
Fig. 2 A derivation tree of $A\rightarrow\alpha\{\beta\}\gamma$ and its transformations to context-free form.

それから導出される木は図2の(a)の形とするのが自然である．しかし，文脈自由文法では直接それを表現することはできず，図2の(b)か(c)の形で表現することになる（図の β^i は β の i 回目の出現を表す）．(a)での各 β の合成属性がその一つ左の β の合成属性から計算されていくのを文脈自由文法の上で表現するには，(b)に変換するのが適している（直感的には，合成属性は下から求められるものであるから，β の列が左下から右上の方向に並んでいる方がよい）．そこで，もとの生成規則を

$$A\rightarrow\alpha B\gamma$$
$$B\rightarrow\varepsilon$$
$$B\rightarrow B\beta \qquad (5)$$

と変換して表現することにする．この B の合成属性 s の評価規則は

$$A\rightarrow\alpha B\gamma \quad \{B.is=s_0;\ldots\}$$
$$B\rightarrow\varepsilon \quad \{B.s=B.is;\ldots\}$$
$$B_0\rightarrow B_1\beta \quad \{B_0.s=g(B_1.s,a_1,\ldots,a_k);B_1.is=B_0.is;\ldots\}$$

の形であるとしても，一般性は失わない．ただし，$a_1,\ldots, a_k$ は β の中の非終端記号の合成属性か B_0 の相続属性である．B の合成属性 s の評価規則は，次の漸化式型の関数で表現できる．

$$s=\mathbf{rec}(g, s_0, a_1, \ldots, a_k) \qquad (6)$$

この関数の意味は次のように与えることにする．ここで，s^i は β が i 回出現したときの s の値とする．

$$s^0=s_0$$
$$s^1=g(s^0, a_1, \ldots, a_k)$$
$$\cdots$$
$$s^i=g(s^{i-1}, a_1, \ldots, a_k)$$

ただし，s^i を評価するときの a_j の値は β の i 回目の出現に対応する値である．

この関数をそのまま $\{\beta\}$ の合成属性 s の評価規則にも使うこととする．ただし，上記の非終端記号 B を使った表現では a_j として A や a から得られる属性値を直接使うことはできず，いったん B の相続属性の値に

置き換えて使う必要があるが，$A\rightarrow\alpha\{\beta\}\gamma$ の場合は A や α が見えているのであるから，$\{\beta\}$ の属性の評価規則の中ではそれを直接使うことを許すことができる．これにより，値を受け渡すだけの属性名の導入が不要になり，意味規則の表現が簡潔になる．

$\{\beta\}$ の相続属性についても同様に考えられるが，β の k 回目の相続属性 i^k から $k+1$ 回目の相続属性 i^{k+1} を求める漸化式は，図1の(c)の形で考えるのが適している（直感的には，相続属性は上から求められるものであるから，β の列が左上から右下に並んでいる方がよい．(b)の形でもできないわけではないが，属性の受け渡しの記述が複雑になる．このように場合に応じて構文規則を書き換えても問題のないことは，LL構文解析のために(b)を(c)や(a)に書き換えれることにも示されている[7]．この場合は，$A\rightarrow\alpha\{\beta\}\gamma$ を生成規則

$A\rightarrow\alpha B\gamma$

$B\rightarrow\beta B$

$B\rightarrow\varepsilon$

に変換することになる．その場合，B の相続属性 i の評価規則は，一般に

$A\rightarrow\alpha B\gamma$ $\{B.i=i_0;...\}$

$B_0\rightarrow\beta B_1$ $\{B_1.i=h(B_0.i, a_1, ..., a_k);...\}$

$B\rightarrow\varepsilon$

の形と考えられ，合成属性と同様に漸化式型の関数 $i=\mathbf{rec}(h, i_0, a_1, ..., a_k)$ で表現でき，その意味は以下のとおりである．

$i^1=i_0$

$i^2=h(i^1, a_1, ..., a_k)$

...

$i^k=h(i^{k-1}, a_1, ..., a_k)$

ただし，i^k を評価するときの a_j の値は β の $k-1$ 回目の出現に対応する値であり，i^k は β の k 回目の出現に対してその相続属性として使われる属性値である．

なお，式（6）の縮退した形として

$s=s_0$ （および $i=i_0$）

も許すこととする．これは，

$s^0=s^1=...=s^n=s_0$ （および $i^1=...=i^n=i_0$）

を意味するものとする．

また，s の評価関数を直感的に分かりやすく表現するために式（6）の略記法として

$s\leftarrow g([s_0], a_1, ..., a_k)$

という記法を許すこととする（下記の図4参照）．

（c）の選択型の関数は，

$s=\mathbf{alt}(e_1, e_2, ..., e_n)$

の形であり，これが $(\beta_1|\beta_2|...|\beta_n)$ の合成属性 s の評価

```
F → .{(0|1)dval}pos_val    < dval=alt(0,2^pos);
                              pos=rec(add1,1); val=rec(add,0,dval);
                              F.val=val >
                             where add(v,d)=v+d
                                   add1(p)=p+1
```

図3 2進小数の正規右辺属性文法（その1）

Fig. 3 A regular right part attribute grammar of binary fractions (1).

```
F → .{(0|1)dval}pos_val    < dval=alt(0,2^pos);
                              pos←[1]+1; val←[0]+dval;
                              F.val=val >
```

図4 2進小数の正規右辺属性文法（その2）

Fig. 4 A regular right part attribute grammar of binary fractions (2).

```
prog → {dcl}env{stmnt}valid  < env←[φ]∪dcl.def; valid←[true]∧(stmnt.use⊂env);
                               prog.valid=valid >
```

図5 宣言と使用の関係の正規右辺属性文法

Fig. 5 A regular right part attribute grammar of def-use relations.

関数であるとしたとき，β_i が出現したときの s の値を s^i とすると，その値は以下のようにして求められる．

$s^i=e_i$

ただし，e_i の表現の中に β_j $(j\neq i)$ の中の非終端記号の属性が現れてはならない．

上に述べた表記法を使えば，最初にあげた2進小数の属性文法（図1）は**図3**，または略記法を使って**図4**のように簡潔に表現することができる．

図5は，いくつかの宣言（dcl）と文（stmnt）からなるプログラム（prog）で，文の中で使われている変数名の集合（stmnt.use）は各宣言で宣言された変数名の集合（dcl.def）の和の部分集合でなければならないことを示す属性文法である．図5で，∪，⊂，∧ はそれぞれ，集合和，集合の包含，論理積を意味するものとする．

3. 正規右辺L属性文法の属性評価

正規右辺文法のLL構文解析の方法は良く知られている．たとえば，

$A\rightarrow\alpha\{\beta\}\gamma$

については，

```
procedure parse_A
  {parse_α;
  while next_token∈First(β) do
      parse_β;
  parse_γ;}
```

とすればよい．これに属性評価をつけ加えるには，一般には属性値を構文解析手続きのパラメタとしてつけ

加えればよい．我々の場合，このような繰り返し型の正規表現については，

```
procedure parse_A
  {parse_α；
   {β} のすべての合成属性 s について
       s：＝s^0；
   while next_token∈First(β) do
       {β} のすべての相続属性 i について
           i：＝h(i, a1, ..., ak)；
           ただし，1回目だけは　i：＝i^1；
        parse_β；
       {β} のすべての合成属性 s について
           s：＝g(s, a1, ..., ak)；
   parse_γ；
  }
```

とすればよい．

4．正規右辺 LR 属性文法の属性評価

正規右辺文法の LR 構文解析では，ある生成規則で還元するとき，還元される記号列の長さが一定でないので，解析スタックの中の記号列をどこまで還元すべきかを決定するのが難しいという問題があり，それを解決するための種々の方法が提案されているが，簡潔でかつ効率も良いものはまだないと言っても過言ではない[8),9)]．しかし，上記のような正規右辺属性文法を考え，LR 構文解析と同時に属性評価をすることを考えれば，たとえば

$A\rightarrow\alpha\{\beta\}\gamma$

なる生成規則に従って還元するのは，まとめて還元するのでなく，β が1回現れる度にその1回分の β だけを還元し，属性評価をするのが自然である．その場合，β が1回現れたときも n 回現れたときもそれが一つの非終端記号（今ここではそれを B と書くことにする）に還元されると考えれば，上の生成規則による還元は

$A\rightarrow\alpha B\gamma$

による還元となる．選択型の正規表現

$A\rightarrow\alpha(\beta_1|\beta_2|...|\beta_n)\gamma$

についても，β_i は一つの非終端記号 C に還元されると考えれば，A への還元は

$A\rightarrow\alpha C\gamma$

による還元となる．すべての繰り返し型および選択型の正規表現についてそのようにすれば，還元の長さが決まらないという問題はなくなる．すなわち，上記の正規右辺属性文法に関する LR 構文解析の問題は，すべての繰り返し型および選択型の正規表現を異なる非終端記号で置き換えて得られる文脈自由文法の LR 構文解析の問題に帰着される．

属性評価に関しても，同様に文脈自由文法の属性評価の問題に置き換えることは可能である．ただし，次のことを考慮する必要がある．

上記の例で $\{\beta\}$ を B に置き換えた場合は，$\{\beta\}$ の中で使われている $\{\beta\}$ の外の属性はすべて B に対する相続属性として与える必要がある．そのようにして得られた属性文法が LR 属性であるためには

（1） L 属性であること

（2） 構文解析の任意の時点で，その時点で必要になるかも知れないすべての相続属性が求めうること

が必要であり，構文解析と同時に属性評価をする場合には（2）の相続属性を評価していく必要がある．たとえば，上記の生成規則に関して，

$A\rightarrow\alpha\bullet B\gamma$

という LR 項を含んだ LR 状態では，B のすべての相続属性（$\{\beta\}$ について宣言されている相続属性，および，$\{\beta\}$の中で使われている$\{\beta\}$の外の属性のすべて）と γ の先頭に現れうるすべての非終端記号の相続属性を求める必要がある．一般的には，その LR 状態に含まれるすべての LR 項について同様のことをする必要があるが，ここではその一つの LR 項について考察する．元の生成規則で考えれば，

$A\rightarrow\alpha\bullet\{\beta\}\gamma$

を含んだ LR 状態では，$\{\beta\}$ について宣言されている相続属性，および，β と γ の先頭に現れうるすべての非終端記号の相続属性だけを求めればよい．

$\{\beta\}$ の合成属性については

$A\rightarrow\alpha\{\beta\}\gamma$ （7）

の構文解析・属性評価はあたかも

$A\rightarrow\alpha B\gamma$ （8）

$B\rightarrow\varepsilon$ （9）

$B\rightarrow B\beta$ （10）

であるかのように行えばよい．$\{\beta\}$ の合成属性 s については式（9）で還元する際に s^0 の評価をし，式（10）で還元する度に s^i の評価をすればよい．

$A\rightarrow\alpha(\beta_1|\beta_2|...|\beta_n)\gamma$

についても同様に考えればよい．（$A\rightarrow\alpha\{\beta\}\gamma$ の場合より簡単である．）

上に述べたことの具体的な手順として，構文解析スタックと属性スタックを使う方法の概略を述べる．

通常，基底文法が LR 文法で，LR 属性である属性文法について，LR 構文解析をしながら解析木を作らずに属性評価をする処理系では，次の三つのスタックを

使う．

（1） 構文解析スタック：LR状態番号を入れる（文法記号は入れない）

（2） 相続属性スタック：必要になる可能性のある相続属性を入れる

（3） 合成属性スタック：還元時に合成属性を入れる

構文解析スタックに状態番号 I_m が入っているとき，相続属性スタック上でそれに対応する所には，その時の先読み記号のもとで状態 I_m から読み始める可能性のあるすべての非終端記号の相続属性を入れる（LR属性文法の定義では，先読み記号によってLR状態を細分化したものを部分状態という[10]）．合成属性スタック上で状態 I_m に対応する所には，状態 I_m から読み始めた記号列を一つの非終端記号に還元したとき，その非終端記号の合成属性を入れる．ここで，一つの非終端記号に複数の属性が付随する場合は，それらをまとめたものを入れる．

その方法を正規右辺属性文法に適用してみよう．まず，

$A \rightarrow \alpha\{\beta\}\gamma$

について考える．

$A \rightarrow \alpha\bullet\{\beta\}\gamma$

を含んだLR状態（それを I_m とする）に到達したとき，I_m を構文解析スタックに積み，その時の先読み記号 x が $x \in \mathrm{First}(\beta)$ であるとき，$\{\beta\}$ のすべての相続属性 i について i^1 を相続属性スタックの対応する場所に積む．次に，$\{\beta\}$のすべての合成属性 s について s^0 を評価して合成属性スタックの対応する場所に積む．そこから状態遷移して

$A \rightarrow \alpha\{\beta\bullet\}\gamma$

を含む状態で β を還元することになったら，先の I_m に対応する場所にある s^{j-1} の値と β の中の非終端記号などの属性値を使って s^j の値を求めてそれで s^{j-1} を置き換え，その時の先読み記号 x が $x \in \mathrm{First}(\beta)$ であるとき i^j などの値を使って i^{j+1} の値を求めてそれで i^j を置き換え，すべてのスタックを先の I_m の場所までポップアップする．β の中にまた繰り返し型などがあっても，それにも同じ手順を施すことによって，β を還元するときの長さは一定であるから，以上のことは容易に実現できる．

文献11）には，生成規則の右辺に

$\langle\langle e_1 \| e_2 \| \ldots \| e_n \rangle\rangle$

の形（permutation phraseという）を許し，これが，e_1 から e_n までのものが任意の順序で現れてもよいことを示すことにする，という提案があり，それを含んだ文法のLL構文解析の方法が述べられている．

この問題は構文解析の問題ではあるが，その構文解析のためには何らかの属性評価が必要になる．本論文のような構文・意味解析の方法を採れば，必要な属性評価をしながらその構文解析が可能になる．たとえば，

$A \rightarrow \alpha\langle\langle e_1 \| e_2 \| \ldots \| e_n \rangle\rangle\gamma$

については，あたかも

$A \rightarrow \alpha\{e_1 | e_2 | \ldots | e_n\}\gamma$

であるかごとくに考えて，上記の

$A \rightarrow \alpha\{\beta\}\gamma$

の場合と同じような処理をする．これによって，各 e_i が任意回数現れる形を解析することができる．ただし，今度の場合は各 e_i が1度だけ出現することをチェックする必要がある．そのために

$A \rightarrow \alpha\bullet\{e_1 | e_2 | \ldots | e_n\}\gamma$

を含んだLR状態（その状態が，α を読み終わったときに到達する状態であり，その後，ある e_i の還元をしたときに戻る状態である）に対応する合成属性スタックの場所にどの e_i の還元が済んだかを覚えておくことにする．それによって，おなじ e_i が2度還元されたか（2度現れたか），1度も還元されなかったかなどのエラーチェックをすることができる．

5. おわりに

属性文法の自然な拡張としての正規右辺属性文法を提案し，L属性，LR属性の場合の属性評価方法の概略を述べた．今後の課題は，属性評価器生成系の開発などを通してその実用性を評価することである．

参考文献

1) Jullig, R. K. and DeRemer, F.: Regular Right-Part Attribute Grammars, *SIGPLAN NOTICES*, Vol. 19, No. 6, pp. 171-178 (1984).

2) Kastens, U., Hutt, B. and Zimmermann, E.: *GAG: A Practical Compiler Generator*, LNCS, Vol. 141, Springer (1982).

3) 丁 亜希，渡辺美樹，中田育男，佐々政孝：正規右辺属性文法と1パス再帰降下属性評価器の生成，情報処理学会論文誌，Vol. 30, No. 2, pp. 204-212 (1989).

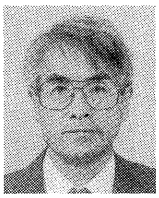
4) 山之上卓，安在弘幸：属性付構文指示翻訳系の生成系MYLANG，情報処理学会論文誌，Vol. 26, No. 1, pp. 195-204 (1985).

5) Elsworth, E. F. and Parkers, M. A. B.: Automated Compiler Construction based on Top-down Syntax Analysis and Attribute Evaluation, *SIGPLAN NOTICES*, Vol. 25, No. 8, pp. 37-42 (1990).

6) 中川裕之，金谷英信，星野英之，山下義行，中田育男：拡張1パス型属性文法によるコンパイラ生成

系の実現, 情報処理学会論文誌, Vol. 36, No. 4, pp. 902-912 (1995).
7) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers — Principles, Techniques, and Tools*, Addison-Wesley (1986).
8) 佐々, 中田：正規右辺文法の LR パーサの簡単な実現法, 情報処理学会論文誌, Vol. 27, No. 1, pp. 124-127 (1986).
9) Zhang, Y. and Nakata, I.: Generation of Path Directed LALR(k) Parsers for Regular Right Part Grammars, *J. Inf. Process.*, Vol. 14, No. 3, pp. 325-334 (1991).
10) Sassa, M., Ishizuka, H. and Nakata, I.: A Contribution to LR-attributed Grammars, *J. Inf. Process.*, Vol. 8, pp. 196-206 (1985).
11) Cameron, R. D.: Extending Context-Free Grammars with Permutation Phrases, *ACM LOPLAS*, Vol. 2, Nos. 1-4, pp. 85-94 (1993).

(平成 6 年 11 月 17 日受付)
(平成 7 年 3 月 13 日採録)



中田 育男（正会員）

1935 年生. 1958 年東京大学理学部数学科卒業. 1960 年同大学大学院修士課程修了. 1960～79 年（株）日立製作所中央研究所, 同システム開発研究所勤務. 1979 年 4 月より筑波大学電子・情報工学系教授. 理学博士. プログラム言語, 言語処理系, ソフトウェア工学などに興味を持っている. 著書「コンパイラ」(産業図書),「基礎 FORTRAN」(岩波書店). ソフトウェア科学会, 電子情報通信学会, ACM, IEEE 各会員.

山下 義行（正会員）

1959 年生. 1982 年大阪大学理学部物理学科卒業. 同年日立マイクロコンピュータ・エンジニアリング（株）入社. 1986 年退社. 1987 年筑波大学大学院博士課程電子・情報工学専攻入学. 1989 年退学, 同年東京大学大型計算機センター助手. 1992 年より筑波大学電子・情報工学系講師. プログラミング言語, コンピュータ・グラフィックスの研究に従事. 日本ソフトウェア科学会会員.