正規右辺属性文法と1パス再帰降下属性評価器の生成†

丁 亜希†† 渡辺美樹†††
中田育男†††† 佐々政孝††††

正規右辺文法，すなわち構文規則の右辺に正規表現の許される文脈自由文法をもとにした属性文法，いわゆる正規右辺属性文法が考えられる．ただし構文規則の中の選択構造や繰り返し構造に対応した属性評価規則の表現法が問題である．これらの点について，我々は新しい観点から意味規則にも正規表現を導入することを考案し，それに基づく読解性の高い意味規則表記法を編み出した．この表記法は属性生起に対する属性評価を意味規則中の正規表現を用いて記述する所に特徴がある．このような表記法を採用した正規右辺属性文法から1パス再帰降下属性評価器を生成するため，入力される正規右辺属性文法から意味処理つき文法木をつくり，その文法木から属性評価器を生成する方法を開発し，生成系を試作した．

1. はじめに

属性文法は文脈自由文法に意味規則を付与したもの，すなわち，文脈自由文法の文法記号に属性を，構文規則に属性値の評価規則を付けたものである．構文木上で，与えられた意味規則に従って属性値を評価したり，属性値をチェックしたりするプログラムを属性評価器という．属性文法のクラス分けおよび各クラスでの属性評価の方法については各種研究されている[6),9)]．

Bochmann は構文木上で左から右へ，上から下への1パス遍歴 (One-pass traversal) で属性が評価できるような属性文法を定義した[1)]．これが実用上重要なクラス，いわゆるL属性文法である．これは1パスで属性が評価できるので，構文木を作らず，構文解析しながら効率的に属性評価が行える．再帰降下パーサを使い，属性評価用のプログラムを各パーサ手続きに埋め込むと，効率的な1パス再帰降下属性評価器を作ることができる．

正規右辺文法は，構文規則の右辺に正規表現の許される文脈自由文法である．正規表現の使用により，構文規則の記述がコンパクトになり，再帰的な構造を繰り返し構造に書き換えることができ，記述の便利さと読解性がよくなる．

この正規右辺文法をもとにした属性文法，いわゆる正規右辺属性文法が考えられる．ただし問題がいくつかある．一つは構文規則の中に選択構造があるから，それに応じて意味規則の中にも選択的な規則が必要になることである．もう一つは構文規則の繰り返し構造に対応した繰り返しごとの属性評価規則や，それらの結果をまとめた属性値などの表現法の問題である．

それを解決する一つの方法は意味規則を構文規則に混在させて選択的な評価を表現するものである[2),5),13)]．そこでは繰り返しごとの評価規則には

attr_name:=...attr_name...

のような記述を用いている．しかし，これでは意味と構文の記述が混在するため読解性は劣るし，評価規則は単一代入文であるという属性文法の特性が失われ，属性文法とは言えないものになってしまう．Jullig らでは評価規則のいくつかの典型的なパターンを考え，それにあわせた表記法を提案している[3)]．この方法では属性文法の特性は保たれており，決められたパターンにあった規則はきれいに書ける．ただし，それ以外のものの表現ができないのが問題である．

我々は意味規則にも正規表現を導入することによって，読解性の高い意味規則表記法を考案し，それに従って1パス再帰降下属性評価器の生成系を開発した．生成系の概略を**図1**に示す．そこで，ジェネレータは入力される正規右辺属性文法から意味処理つき文法木をつくり，その文法木から1パス再帰降下属性評価器を生成する．字句解析部は別途（例えば，Lex により）作られる．属性評価関数はユーザ定義の関数である．字句解析部，属性評価器と属性評価関数がコンパイラを構成する．

以下，第2章で我々が提案する正規右辺属性文法お

† Regular Right Part Attribute Grammars and Generation of One-pass Recursive Descent Attribute Evaluators by YAXI DING (Doctoral Program in Engineering, University of Tsukuba), YOSHIKI WATANABE (Fuji Xerox), IKUO NAKATA (Institute of Information Sciences and Electronics, University of Tsukuba) and MASATAKA SASSA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学工学研究科
††† 富士ゼロックス
†††† 筑波大学電子・情報工学系

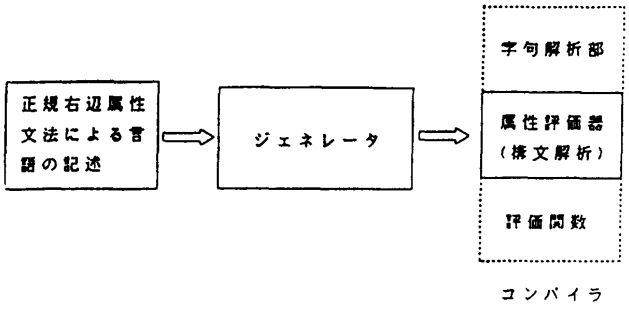

図 1 生成系の構成
Fig. 1 The generation system for regular right part attribute grammars.

よびその表記法を述べ，第3章で意味処理つき文法木から再帰降下属性評価器を生成する方法を説明し，第4章で属性評価規則から意味処理文を生成する方法を述べることにする．

2. 正規右辺属性文法

2.1 正規右辺文法

正規右辺文法 G は四つ組 (V_N, V_T, S, P) である．V_N, V_T はそれぞれ非終端記号の集合，終端記号の集合である．$V=V_N \cup V_T$ とする．$S \in V_N$ は出発記号である．P は右辺に正規表現の許される構文規則の集合である．構文規則 $p \in P$ は，

$A : \alpha ;$

で表すことにする．ここで，$A \in V_N$ である．α は V 上の正規表現であり，次のように再帰的に定義される．ここでは通常の正規表現を少し拡張している．

(1) ε, $X \in V$ は正規表現である．

以下，α, α_1, α_2 を正規表現とする．

(2) $\alpha_1\alpha_2$ も正規表現である．これを連結構造 (conc) と呼ぶ．

(3) $(\alpha_1|\alpha_2)$ も正規表現である．これを選択構造 (alt) と呼ぶ．曖昧でないときは，括弧を使わなくてもよい．

(4) $[\alpha]$, $\{\alpha\}$, $\{\alpha\}+$, $\{\alpha//d\}$ も正規表現である $(d \in V_T)$．これらをそれぞれオプション (opt), 繰り返し (rep), シーケンス (seq), リスト (list) と呼ぶ．リストは d を区切り記号として α を並べたものである．

(5) (1)-(4)項以外のものは正規表現ではない．

上記の記号 ( ), |, [ ], { }, +, // などは正規表現のメタ記号という．また，正規表現 α の値を value(α) で記す．例えば，$\alpha = aA|b\{B\}$ であるとき，

$\text{value}(\alpha) = \{aA, b, bB, bBB, \ldots\}$

である．

構文規則の意味するところを対応する BNF 記法で示すと次のとおりである．

正規右辺文法の記法	相当する BNF の記法
$A : u\|v ;$	$A \to u ;\ A \to v ;$
$A : u[v]w ;$	$A \to uBw ;$
	$B \to v ;\ B \to \varepsilon ;$
$A : u\{v\}w ;$	$A \to uBw ;$
	$B \to vB ;\ B \to \varepsilon ;$
$A : u\{v\}+w ;$	$A \to uBw ;$
	$B \to vB ;\ B \to v ;$
$A : u\{v//d\}w ;$	$A \to uvBw ;\ B \to \varepsilon ;$
	$B \to dvB ;$

2.2 正規右辺属性文法と意味規則表記法

我々の提案する正規右辺属性文法は正規右辺文法を属性を使えるようにできるだけ自然に拡張したものである．正規右辺文法の文法記号 $X \in V$ に属性の集合 $A(X)=S(X) \cup I(X)$ が付けられる．$S(X)$ と $I(X)$ はそれぞれ合成属性の集合と相続属性の集合である．X が開始記号であるとき，あるいは $X \in V_T$ であるとき，$I(X)=\phi$ とする．

構文規則の特定の場所に現れる X の属性 $a \in A(X)$ を属性出現 (appearance) という．正規表現 α において，$v \in$ value(α) に現れる文法記号 X の属性 a を属性生起 (occurrence) という．構文規則ごとに属性評価規則が付いている．これは，構文規則が $X_0 : \alpha$; で，X が α の中に現れる文法記号とすれば，$S(X_0)$ と $I(X)$ 中の属性出現の値を計算する規則である．$S(X_0)$ と $I(X)$ 中の属性出現を Output 属性といい，$I(X_0)$ と $S(X)$ 中の属性出現を Input 属性という．

正規表現の導入前の属性文法については，属性出現と属性生起を区別する必要がないが，我々の正規右辺属性文法においては，これらは異なる概念である．これは正規右辺属性文法の属性評価規則表記法の問題点を明確にするため必要である．例えば，$A : [a]B$; の構文規則に対して，相続属性出現 B. inh を入力 a があったかどうかに従ってそれぞれ評価関数 $f1$ か $f2$ かで評価したいとする．そのとき，二つの属性生起があり得る．a に出会った時の B. inh と，a に出会わなかった時の B. inh である．正規右辺属性文法の表記法では，属性評価規則中属性出現だけでなく，場合によって属性生起の評価方法を指定しなければならない．そのため，我々は属性評価規則にも正規表現を導入する方法[12]を提案する．

属性評価規則は，属性出現，評価関数，定数および

演算子の集合（それをΣと記す）上の正規表現によって記述される．

まず，一つの例を見ながら我々の表記法を説明する．

例 2.1：次は加算と減算からなる「式」の値を求める文法である．

exp：$ID_1\{_1(_2\text{ADD}|\text{SUB})ID_2\}$；

%attr

exp. val:=ID_1. val$\{_1(_2+|-)ID_2$. val$\}$；

この生成規則で，小文字で始まるのは非終端記号で，大文字で始まるのは終端記号である．属性評価規則は%attr で始まる．ID. val は識別子の値を表す合成属性である．exp. val は式の値を表す合成属性である．ID を読み込むごとにその ID. val を加算，あるいは減算して，最終の結果を属性出現 exp. val の値とする．

属性評価規則には，構文規則に含まれるメタ記号(,),[,],{,} などに対応して同じ記号を用いることができる．その対応関係を示すためにこれらの記号に$\{_n$ や $(_n$ などの形の添字をつける．例えば，この例では属性評価規則の中の $\{_1\ldots\}$ は構文規則の中の $\{_1\ldots\}$ に対応するものである．入力がその繰り返しを m 回用いて受理されるとき，属性評価は属性評価規則の中の対応する繰り返しを m 回用いて得られたものによって評価される．属性評価規則中の $(_2+|-)$ は構文規則の中の $(_2$ ADD|SUB) に対応するものである．入力が ADD のときは属性評価規則に加算演算子＋を用いて，入力が SUB のときは属性評価規則に減算演算子－を用いて，それぞれ属性評価を行う．例えば，入力が

ID_1 ADD ID_2^1 SUB ID_2^2

であるときは，評価規則を

exp. val:=ID_1. val$+ID_2^1$. val$-ID_2^2$. val；

と解釈する．上つきの添字は繰り返しの順番を示すために用いた記号である．

次に，２種類の正規表現の形の属性評価規則を構文規則と対応させて説明する．

[属性評価規則パターンＩ]

構文規則 $X_0:\alpha$；に対して，属性評価規則は

attr_name:=β；

のような形で記述できる．

attr_name はこの構文規則の中の Output 属性出現である．α は V 上の正規表現である．β はΣ上の正規表現である．ただし，β はその中のメタ記号の使い方およびその値の決め方についていくつかの制限があるものである．β に用いられるメタ記号は α に含まれるメタ記号に対応したものであり，メタ記号 (,[,{ などの後に $_n$ のような添字でその対応関係を示す．n は正整数であり，異なるメタ記号には異なる整数値をつけることにする．同じ整数値のついたメタ記号は再帰的に使用できない．すなわち，

$(_n\ldots|\ldots(_n\ldots|\ldots)\ldots)$

という形の使用は許さない．ただし，

$(_n\ a|b)\ldots(_n\ c|d)$

のような形で２か所以上に現れてもよい．

β	αの中でβに対応する部分	vの中で左の欄に対応する部分の形	$\lambda(\beta)$
a	α	α	a
e_1e_2	α	α	$\lambda(e_1)\lambda(e_2)$
$(_ne_1\,|\,e_2)$	$(_np\,|\,q)$	p	$\lambda(e_1)$
		q	$\lambda(e_2)$
$[_ne]$	$[_np]$	p	$\lambda(e)$
		ε	ε
$[_ne_1\,\|\|\,e_2]$	$[_np]$	p	$\lambda(e_1)$
		ε	$\lambda(e_2)$
$\{_ne\}$	$\{_np\}$	ε	ε
		p	$\lambda(e)$
		$p^1p^2\ldots$	$\lambda(e^1)\lambda(e^2)\ldots$

図 2 与えられた $v\in$value(α) に対応する β の意味 $\lambda(\beta)$
Fig. 2 The meanings $\lambda(\beta)$ of β for the given $v\in$value(α).

記述上簡単のために構文規則中の rep, seq, list に対して，属性評価規則では別々のメタ記号を用いず，すべて $\{_n$ の形で示すことにする．

β の意味（それを $\lambda(\beta)$ と書く）は対応する α と入力 $v\in$value(α) から図 2 のように再帰的に定義する．図２中，$a\in\Sigma^*$ であり，$e1$, $e2$, e および p, q はそれぞれ属性評価規則と構文規則中に用いられる正規表現である．v ($\in$value(α)) に関する欄中の p などは，β 対応する α の部分に関して，それに対応する v の部分が p の形（すなわち value(p) の要素）をしていることを意味する．文法が曖昧でない場合，そういう p の形をする v の部分は一意的に決められる．

$\lambda(\beta)$ の中の文法記号としては構文規則の左辺にある非終端記号 X_0 および v に出現した文法記号しか出現しないとする．したがって，$\lambda(\beta)$ 中の属性名は v 中の属性生起を意味する．ただし，繰り返しに対して，e の中に p の中の文法記号 X に対する属性出現がある場合，$\lambda(e^i)$ 中の対応する属性生起は p^i 中の X に対する属性生起を意味する．

以下，VALUE(β)= {v に対応する $\lambda(\beta)|v\in$value(α)} と定義する．

$\lambda(\beta)$ はすべて評価可能な式の形になるものでなければならない．特に，繰り返し構造に対して，評価規

則中の繰り返し構造が $e_0\{_n e\}e_1$ の形とすれば，$\lambda(e_0)\lambda(e^1)\lambda(e^2)\ldots\lambda(e_1)$ は評価可能な式の形になるものでなければならないから，e は次の2種類のどちらかの形でなければならない．

1) $w\in$VALUE(e) は二項演算子で始まる記号の列である．w の最後は二項演算子ではない．そのとき e を演算子前置型という．$e_0\neq\varepsilon$ である．$e_1\neq\varepsilon$ ならば，$w_1\in$VALUE(e_1) も二項演算子で始まる文法記号の列である．例2.1の $(+|-)ID_2$.val は演算子前置型である．

2) $w\in$VALUE(e) は二項演算子で終わる記号の列である．w の最初は二項演算子ではない．そのとき e を演算子後置型という．$e_1\neq\varepsilon$ である．$e_0\neq\varepsilon$ ならば，$w_0\in$VALUE(e_0) も二項演算子で終わる文法記号の列である．例えば，$a+\{_n b-\}c$ の中の $b-$ は演算子後置型である．

なお，実行効率のよい1パス属性評価器を作るためには，評価規則中の繰り返し構造 $e_0\{_n e\}e_1$ の中の e は繰り返しごとに計算を進めることができるような形でなければならない．例えば，右結合のパワー演算子 $\uparrow$ を含む評価規則 $\{_n a\uparrow\}b$ に対して計算は繰り返しから抜け出した場所まで実行できない．また，評価規則 $a\{_n+b|*c\}$ も繰り返しごとに計算できない例である．

属性評価の典型的なパターンの一つに縫い糸型[10]と呼ばれるものがある．それは次の例2.2に示されるようなものである．それを表現するために次の形を導入する．

[属性評価規則パターンII]

属性評価規則には

$\beta_1\{_n$=:attr_name$_1$; $\beta_2\}$ =:attr_name$_2$

のような形の記述ができる．

$\beta1$, $\beta2$ は前記のΣ上の正規表現である．=: は右定義記号という．a=:b とは，a の値で b の値を定義するという意味である．ここで，メタ記号 $\{_n\}$ も上述の Σ上の正規表現に用いられるメタ記号 $\{_n\}$ と同じ意味とする．

例 2.2: 次は PL/0 言語[4]の記述中の変数宣言部分である．

```
vardeclpart : VAR{vardecl//','}';';
%attr
vardeclpart. I_env
    {=:vardecl. I_env ; vardecl. S_env}
    =:vardeclpart. S_env ;
```

I_env と S_env はそれぞれ環境（コンパイラの記号表にあたるもの）を表す相続属性と合成属性を表す．入力 VAR vardecl[1], vardecl[2] に対して属性評価規則の意味は

```
vardeclpart. I_env=:vardecl[1]. I_env ;
vardecl[1]. S_env=:vardecl[2]. I_env ;
vardecl[2]. S_env=:vardeclpart. S_env ;
```

となる．

以上，属性生起に対する属性評価は属性評価規則の中の正規表現を用いて計算できる．α と β の対応関係および $\lambda(\beta)$ の定義は直観的にわかりやすいものであるから，この条件を満足するように属性評価規則を書くことは比較的容易であろう．

3. 意味処理つき文法木

1パス再帰降下属性評価器を作成するには，再帰降下パーサに意味処理部分を付け加えればよい．つまり，再帰降下パーサの手続き中の適当な場所に構文規則につけられた意味規則から変換された意味処理文の列を埋め込んで，非終端記号の属性をその非終端記号に対応する構文解析手続きの引数とすればよい．

意味処理つき正規表現からオートマトンへの変換方法として中田のもの[8]があるが，ここではそれと違って，まず入力された正規右辺属性文法から意味処理つき文法木へ変換し，次にそれから1パス再帰降下属性評価器を生成する方法[11]を提案する．オートマトンより文法木の方が再帰降下パーサを生成しやすいと考えられるからである．

3.1 意味処理つき文法木と再帰降下評価器の生成

入力された文法の内部表現として，各非終端記号に対して，その非終端記号が左辺に出現する構文規則をまとめた一つの文法木というものを作る（例2.1の文法木を図3に示す）．

文法木の各ノード t には以下の情報が入っているとする．t.class は正規表現に対応してノードの種類

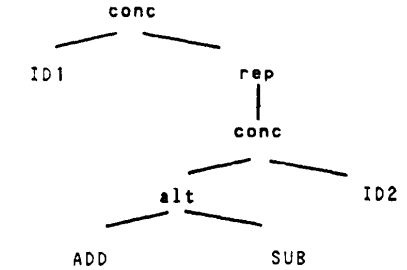

図3 例2.1の文法木
Fig. 3 The grammar tree of EX. 2.1.

評価手続きのパターンΓ(t)

```
1.  ε      t.pre;    t.post;
2.  term   t.pre;
           IF ( sym==M(t) ) THEN getsym(); ELSE error();
           t.post;
3.  non    t.pre;
           CALL proc_M(t)(a1,a2,...,am;b1,b2,...,bn);
           /* a1,a2,...,amはM(t)の相続属性である.
              b1,b2,...,bnはM(t)の合成属性である */
           t.post;
4.  conc   t.pre;
           Γ(t.c1);Γ(t.c2)...;Γ(t.cn);
           t.post;
5.  alt    t.pre;
           CASE sym IN
               FIRST(t.c1) : Γ(t.c1);
               FIRST(t.c2) : Γ(t.c2);
                        ...
               FIRST(t.cn) : Γ(t.cn);
             もしε∉FIRST(t)ならば次の一行
              を加える
              otherwise    : error();
           ESAC
           t.post;
6.  opt    t.pre;
           IF sym IN FIRST(t.c)
              THEN Γ(t.c);
           t.post;
7.  rep    t.pre;
           WHILE sym IN FIRST(t.c) DO
             BEGIN  Γ(t.c)  END;
           t.post;
8.  seq    t.pre;
           DO  Γ(t.c);
           UNTIL sym NOT_IN FIRST(t);
           t.post;
9.  list   t.pre;
           LOOP:  Γ(t.c);
                  IF sym==Separator THEN
                   BEGIN getsym();
                         GOTO LOOP  END;
           t.post;
```

図 4 再帰降下属性評価器生成のパターン
Fig. 4 Patterns for recursive descent attribute evaluators.

を示すものであり，その値は {ε, term, non, alt, conc, rep, opt, seq, list} の中の一つである．t.class が non (非終端記号) または term (終端記号) である場合，その文法記号を $M(t)$ と記す．ノード t に子供がいる時，左から右の順に $t.c_1, t.c_2, \cdots t.c_n$ のように記す．

次に，意味規則から変換された意味処理文の列を文法木のノードに付随させて，意味処理つき文法木を構成する．問題はこの意味処理文の列をどのようにして生成するかである．それは4章で論じるとして，ここではそれが生成できた後の処理について述べる．

一つのノード t に二つの処理文列 t.pre と t.post を付けることにする．t.pre (あるいは t.post) は t をルートとしている部分木から生成される構文解析譜の実行の前 (あるいは後) で実行される意味処理文の列である．

再帰降下属性評価器のパーサ部分の生成法は基本的に通常の方法[7]に従う．属性評価の部分，つまり意味処理文をパーサに埋め込むには，意味処理つき文法木のノードに対して図4に示す属性評価器の生成ルールを使えばよい．これは通常の再帰降下パーサの生成ルールに t.pre と t.post を前後に付加する部分を付け加えただけのものである．そのルールの意味を以下に説明する．

生成規則 $A : \alpha$; に対して，意味処理つき文法木のルートを t としたとき，手続き

proc_$A(a_1, a_2, \ldots, a_m$; var $b_1, b_2, \ldots, b_n)$
BEGIN $\Gamma(t)$ END

を生成する．ここで，$a_1, a_2, \ldots, a_m$ は A の相続属性，$b_1, b_2, \ldots, b_n$ は A の合成属性である．$\Gamma(t)$ は木ノード t から生成すべき評価器のパタンを示すものであり，図4のとおりである．そこで sym はパーサが先読みしてある「次のシンボル」を示すものとする．getsym ()は次のシンボルを sym に読み込む手続きである．

意味処理文列の間の「実行順序」を表すには，記号↦を使う．それは次のように，文法木に対する深さ優先，左から右への遍歴の順序に従って決まる．すなわち，t.class が

ε, non, term のとき　t.pre↦t.post
conc のとき　t.pre↦子供1の順序↦子供2の順序↦…↦子供 n の順序↦t.post
rep, opt, seq, list のとき
t.pre↦子供の順序↦t.post
alt のとき
t.pre↦子供1の順序↦t.post
t.pre↦子供2の順序↦t.post
……

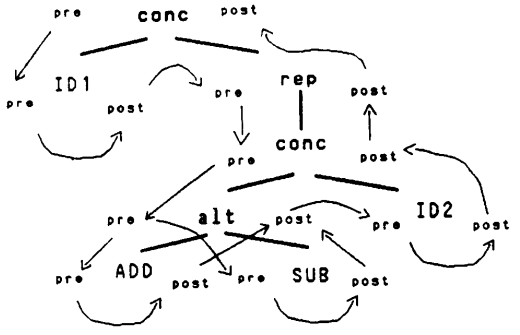

図 5 例 2.1 の文法木上の処理文列の実行順序
Fig. 5 Evaluating order of semantic statements on the tree of EX. 2.1.

t. pre↦子供 n の順序↦t. post

である．ここで，子供の順序とはその子供をなす部分木に付けられている意味処理文列の実行順序を意味する．

$\delta_i \mapsto \ldots \mapsto \delta_k$ であれば $\delta_i \overset{+}{\mapsto} \delta_k$ と記す．$\delta_i = \delta_k$，あるいは $\delta_i \overset{+}{\mapsto} \delta_k$ であるとき $\delta_i \overset{*}{\mapsto} \delta_k$ と記す．$\delta_i \overset{+}{\mapsto} \delta_k$ も $\delta_k \overset{+}{\mapsto} \delta_i$ も成立しないとき $\delta_i \langle\rangle \delta_k$ と記す．

例2.1の文法の文法木に付けられる意味処理文列の順序を**図5**に示す．

3.2 構文解析フラグ

構文規則に正規表現が使えるので，構文解析の道が何通りかあり得ることになる．そこで，その道（通った場所）を憶えるため，内部実現としてフラグを使うことにする．フラグは構文規則の正規表現の構造によるものであり，正規表現の特定の構成部分に対して一つのフラグを用意すればよい．

例えば，$({}_n\alpha_1|\alpha_2|\ldots|\alpha_s)$ に対して flag_n を用意し，それの値が代入される場所は

$\mathrm{flag}_n := i$, α_i の pre, $i = 1, 2, \ldots, s$

である．

$[{}_n\alpha_1|\alpha_2]$ は $[{}_n({}_m\alpha_1|\alpha_2)]$ を略したものである．例えば，$[{}_n\alpha_1|\alpha_2|\ldots|\alpha_s]$ に対して，一つのフラグ flag_n を用意すればよい．

$\mathrm{flag}_n := 0$, opt の pre,

$\mathrm{flag}_n := i$, α_i の pre, $i = 1, 2, \ldots, s$

繰り返し構造に対する flag の値は繰り返しごとに1を加算される．たとえば，$\{{}_n\alpha\}$ に対して，

$\mathrm{flag}_n := 0$, rep の pre;

$\mathrm{flag}_n := \mathrm{flag}_n + 1$, α の pre,

$\mathrm{flag}_n := \infty$ rep の post,

とすればよい．ただし，$\mathrm{flag}_n = 0$ が「繰り返しに入る前」と意味するので，rep の親が conc であるときは $\mathrm{flag}_n := 0$ をこの conc の pre で実行する．

4. 意味処理文の生成

この章では属性評価規則から意味処理文の生成法および与えられた属性評価規則による属性の評価が構文解析と同時に1パスで行われるかどうかのチェック法を述べる．

4.1 評価規則の展開

2章で述べたように属性評価規則には

パターンⅠ

attr_name:=β;

パターンⅡ

$\beta_1\{{}_n$:=attr_name$_1$; $\beta_2\}$ =:attr_name$_2$

という2種類の形がある．ここで attr_name は Output 属性出現である．$\beta, \beta_1, \beta_2,$ は評価規則中の正規表現である．評価器のパーサ部分に埋め込まれる意味処理文を生成するため，このような正規表現を用いる評価規則を

attr_name:=w;

の形に展開する必要がある．w は正規表現のメタ記号を含まない実行できる式である．また，評価規則は前記の構文解析フラグ値によってそれを評価すべきかどうかが制御される．その制御に関するフラグは一つとは限らない．π を構文解析フラグに関する条件の集合とする．正規表現のメタ記号を含む規則を展開する過程中に，その π は作り出される．さらに，展開される評価規則に関して，遅くともあるところまでに実行しなければならないという場所がある．その場所を $\bar{\delta}$ で記す．

評価規則を展開するため以下に示す展開手続き $T(\beta, \pi, \bar{\delta}, \gamma)$ を利用する．ここで，β は評価規則中の正規表現とする．γ は β で評価される属性出現である．

属性評価規則のパターンⅠに対しては，最初に

$T(\beta, \phi, \bar{\delta}, \mathrm{attr\_name})$

を用いて評価規則を展開していく．π の初期値は空集合である．T は β を展開しながら構文解析フラグに関する情報を π に追加する．attr_name が相続属性であるとき，$\bar{\delta}$ は attr_name の付随する非終端記号のノードの pre である．attr_name が合成属性であるとき，$\bar{\delta}$ は現在対象とする構文規則の文法木ルートの post である．

パターンⅡに対しては，

$T(\beta_1, \{\mathrm{flag}_n = 0\}, \bar{\delta}_1, \mathrm{attr\_name}_1)$

$T(\beta_2, \{0 < \mathrm{flag}_n < \infty\}, \bar{\delta}_2, \mathrm{attr\_name}_1)$

$T(\mathrm{attr\_name}_1, \{\mathrm{flag}_n = \infty\}, \bar{\delta}_3, \mathrm{attr\_name}_2)$

を用いて評価規則を展開していく．ここで $\bar{\delta}_1$ は rep_n の pre であり，$\bar{\delta}_2$ は rep_n の子供の post であり，$\bar{\delta}_3$ は上記のように attr_name$_2$ が相続/合成属性であるかによって決まる．すなわち，繰り返しに入る前に attr_name$_1$ に β_1 から得られた値を代入し，その以後の繰り返しごとの attr_name$_1$ に β_2 から得られた値を代入する．繰り返し終了後，最後に得られた attr_name$_1$ の値を用いて attr_name$_2$ を計算する．（正確には attr_name$_1$ は繰り返しごとに別の属性生起を意味するのであるが，実行上はこのように同じ場所を共用してもよい）

以下に，β の各々の形に対する展開手続き $T(\beta, \pi, \bar{\delta}, \gamma)$ の動きを示す．

a） 正規表現のメタ記号を含まない規則 β に対する展開

$T(\beta, \pi, \bar{\delta}, \gamma) \Rightarrow (\gamma := \beta, \pi, \bar{\delta})$

これは評価規則変換の最終の形である．$\gamma := \beta$ の実際の実行場所は $\pi, \bar{\delta}$ および β の中の属性出現によって決められる．その決める方法は次節で説明する．

b） 選択項に対する展開

$T(\beta_0({}_n\beta_1 | \beta_2 | \ldots | \beta_m)\beta_{m+1}, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\beta_0\beta_i\beta_{m+1}, \pi \cup \{\mathrm{flag}_n = i\}, \bar{\delta}, \gamma)$

ここで，$i = 1, 2, \ldots, m$ である．また，β_0, β_{m+1} にメタ記号 $({}_n)$ が存在する場合それを同時に展開する．例えば，

$T({}_na | b)({}_nc | d), \pi, \bar{\delta}, \gamma)$ を

$T(ac, \pi \cup \{\mathrm{flag}_n = 1\}, \bar{\delta}, \gamma)$

$T(bd, \pi \cup \{\mathrm{flag}_n = 2\}, \bar{\delta}, \gamma)$

に展開する．

c） オプションに対する展開

$T(\beta_0[{}_n\beta_1 | \beta_2 | \ldots | \beta_m]\beta_{m+1}, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\beta_0\beta_{m+1}, \pi \cup \{\mathrm{flag}_n = 0\}, \bar{\delta}, \gamma)$

$T(\beta_0\beta_i\beta_{m+1}, \pi \cup \{\mathrm{flag}_n = i\}, \bar{\delta}, \gamma)$

または

$T(\beta_0[{}_n\beta_1 | \ldots | \beta_m \| \beta'']\beta_{m+1}, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\beta_0\beta'\beta_{m+1}, \pi \cup \{\mathrm{flag}_n = 0\}, \bar{\delta}, \gamma)$

$T(\beta_0\beta_i\beta_{m+1}, \pi \cup \{\mathrm{flag}_n = i\}, \bar{\delta}, \gamma)$

$i = 1, 2, \ldots, m$ である．β_0, β_{m+1} にメタ記号 $[{}_n]$ が存在する場合もそれを同時に展開する．

d） 繰り返し部分に対する展開

繰り返し部分 $\beta_0\{{}_n\beta_1\}\beta_2$ の値は β_0 から得られる値と繰り返しごとの β_1 から得られる値と β_2 から得られる値からなる．

繰り返し中の計算の中間結果を γ に一時的に退避する．β_0 にはメタ記号 $\{{}_n\}$ が存在しないと仮定する．

1） β_1 が演算子前置型であるとき

$T(\beta_0\{{}_n\beta_1\}\beta_2, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\beta_0, \pi \cup \{\mathrm{flag}_n = 0\}, \bar{\delta}_1, \gamma)$

$T(\gamma\beta_1, \pi \cup \{0 < \mathrm{flag}_n < \infty\}, \bar{\delta}_2, \gamma)$

$T(\gamma\beta_2, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

$\bar{\delta}_1$, $\bar{\delta}_2$ はそれぞれこの繰り返しノードの pre, そのノードの子供の post とする．$\bar{\delta}_1 \overset{*}{\rightarrow} \bar{\delta}_2 \overset{*}{\rightarrow} \bar{\delta}$ である．

β_2 にもメタ記号 $\{{}_n\}$ が存在するとき，β_2 を $OP\alpha$ の形となるように β_2 から最初の二項演算子の部分を分離する必要がある．OP は二項演算子のみからなる正規表現である．そこで，上記の最終行

$T(\gamma\beta_2, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

を

$T(\alpha, \pi, \bar{\delta}, tmp)$

$T(\gamma\ OP\ tmp, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

で入れ替える．ここで，tmp は新規増設の一時変数である．α の値を計算して tmp に一時的に退避する．繰り返しから抜け出した後にその tmp の値を用いて γ を計算する．ここで注意したいのは，α を $\alpha_0\{{}_n\alpha_1\}\alpha_2$ とすれば，繰り返しの中で α_1 に関する計算と β_1 の計算は並行して行われ，各中間結果はそれぞれ一時的に tmp と γ に退避されることである（実行場所は次節参照）．

2） β_1 が演算子後置型であるとき，$\beta_1 = \alpha_1\ OP_1$ と書ける．$\beta_0 \neq \varepsilon$ ならば $\beta_0 = \alpha_0\ OP_0$ と書ける．そうすると，

$T(\alpha_0\ OP_0\{{}_n\alpha_1\ OP_1\}\beta_2, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\alpha_0, \pi \cup \{\mathrm{flag}_n = 0\}, \bar{\delta}_1, \gamma)$

$T(\gamma\ OP_0\ \alpha_1, \pi \cup \{\mathrm{flag}_n = 1\}, \bar{\delta}_2, \gamma)$

$T(\gamma\ OP_1\ \alpha_1, \pi \cup \{1 < \mathrm{flag}_n < \infty\}, \bar{\delta}_2, \gamma)$

$T(\gamma\ OP_1\ \beta_2, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

または，$\beta_0 = \varepsilon$ のとき

$T(\{{}_n\alpha_1\ OP_1\}\beta_2, \pi, \bar{\delta}, \gamma)$

$\Rightarrow$

$T(\alpha_1, \pi \cup \{\mathrm{flag}_n = 1\}, \bar{\delta}_2, \gamma)$

$T(\gamma\ OP_1\ \alpha_1, \pi \cup \{1 < \mathrm{flag}_n < \infty\}, \bar{\delta}_2, \gamma)$

$T(\gamma\ OP_1\ \beta_2, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

である．β_2 にもメタ記号 $\{{}_n\}$ が存在するときは上記の最終行

$T(\gamma\ OP_1\ \beta_2, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

を

$T(\beta_2, \pi, \bar{\delta}, tmp)$

$T(\gamma\ OP_1\ tmp, \pi \cup \{\mathrm{flag}_n = \infty\}, \bar{\delta}, \gamma)$

で入れ替える．

以上に述べた展開過程中，属性評価規則に用いられるメタ記号が構文規則中のそれを対応するかどうか，または同じ整数値を付いたメタ記号なら再帰的に使われるかどうかをチェックする．

例 2.1 の属性評価規則から展開された規則を以下に

示す．ここで，$\bar{\delta}_1$, $\bar{\delta}_2$ はそれぞれ繰り返しノードの pre とこの繰り返しノードの子供（すなわち conc）の post である．

(exp. val:=ID_1. val, {flag$_1$=0}, $\bar{\delta}_1$)
(exp. val:=exp. val+ID_2. val,
{0<flag$_n$<∞, flag$_2$=1}, $\bar{\delta}_2$)
(exp. val:=exp. val−ID_2. val,
{0<flag$_n$<∞, flag$_2$=2}, $\bar{\delta}_2$)

4.2 展開された規則の実行場所

前に述べたとおり，展開された属性評価規則は

$(p_0:=f(p_1, p_2, \ldots, p_n), \pi, \bar{\delta})$

の形をしている．p_0は属性出現，あるいは評価規則の展開中新設した一時変数 *tmp* である．f は p_0 の値を計算する式である．p_k $(0<k<=n)$ は f に現れる属性出現である．π は構文解析フラグに関する条件の集合である．$\pi=\{\pi_1, \pi_2, \ldots, \pi_m\}$ とする．$\bar{\delta}$ はこの展開された規則をそこまでに実行しなければならない場所である．

$p_0:=f(p_1, p_2, \ldots, p_n)$ を実際に実行する場所を δ とする．以下にその実行場所 δ を決める方法について述べる．

δ では，次の条件が必要である．

1) p_k $(0<k<=n)$ の値が既に計算されている．
2) π_i $(0<i<=m)$ がすべて満足される．
3) $\delta \overset{*}{\rightarrow} \bar{\delta}$ でなければならない．

p_k の値が得られる場所を $\Delta(p_k)$ で示す．π_i の成り立つとする場所（あるいは，成り立つかどうかが判断できる場所）を $\Delta(\pi_i)$ で示す．すると，上記の三つの条件は，

1) $\Delta(p_k) \overset{*}{\rightarrow} \delta$, $(k=1, 2, \ldots, n)$
2) $\Delta(\pi_i) \overset{*}{\rightarrow} \delta$, $(i=1, 2, \ldots, m)$
3) $\delta \overset{*}{\rightarrow} \bar{\delta}$

であることと等価である．

$\Delta(p_k)$ は次のように決められる．

$p_k=p_0$ であるとき，この $\Delta(p_k)$ を無視してもよい．このとき，p_k は繰り返し中の一時的に退避された値である．繰り返しに関する情報が π に入っているので，π が満足されるときは，その一時退避された p_k の値も保証できる．

p_k が Input 属性であるとき，$\Delta(p_k)$ は文法の構造によって決まる．つまり p_k が相続属性であるとき，$\Delta(p_k)$ は文法木ルートの pre である．p_k が合成属性であるとき，$\Delta(p_k)$ はこの属性の付随される文法記号のノードの post である．

p_k が Output 属性であるとき，この p_k を計算する意味規則が現在対象としている構文規則に付随してあるはずである．p_k が *tmp* であるとき，この *tmp* の値を計算する規則が展開されたはずである．すなわち，展開された規則

$(p_k:=f'(\ldots), \pi', \bar{\delta}')$

が存在している．1パス属性評価ができるためには，p_k の計算の場所が p_0 の計算の場所より前になければならない．いま，処理中の

$(p_0:=f(p_1, p_2, \ldots, p_n), \pi, \bar{\delta})$

は

$(p_k:=f'(\ldots), \pi', \bar{\delta}')$

に依存している．p_k を計算する規則を先に処理して $\Delta(p_k)$ が求められる．依存関係にサイクルがあってはならないが，それを発見するのには，展開された規則ごとにマークを使用すればよい．

$\Delta(\pi_i)$ については次のように決められる．π_i には以下のケース

1) flag$_n$=s (s は正整数である)
2) 0<flag$_n$<∞
3) 1<flag$_n$<∞
4) flag$_n$=∞

がある．

ケース1)に対しては，flag$_n$ がオプション (opt) に関するフラグであるとき $\Delta(\pi_i)$ はその opt の post とする．flag$_n$ がそれ以外のフラグであるとき $\Delta(\pi_i)$ は 3.2 節で述べたとおり flag$_n$:=s あるいは flag$_n$:=flag$_n$+1 (flag$_n$ が繰り返しに関するフラグの場合) をセットする場所とする．ケース2)，3)に対しては，$\Delta(\pi_i)$ は繰り返しノードの pre とする．ケース4)に対しては，$\Delta(\pi_i)$ はこの繰り返しノードの post とする．

また誤った意味記述を検出するため，この n 個の $\Delta(p_k)$ と m 個の $\Delta(\pi_i)$ について，お互いに◇ではないことをチェックする．

上記の条件を満たす場所の中で一番早い場所を評価場所 δ とすればよい．

4.3 意味処理文の組立

評価場所 δ が決まったら，次の意味処理文

IF(π_1 AND π_2 AND ... AND π_m)
THEN $p_0:=f(p_1, p_2, \ldots, p_n)$;

を組み立て，文法木の場所 δ につければよい．ただし，π_i が繰り返しに関するフラグの flag$_n$=0 あるいはケース2)，4)であるとき，それを IF 文の条件部分に入れる必要がない．それらは評価場所を決定する際参考とする情報であるが，δ の決め方により，場所 δ

ではそれらの条件は必ず成り立つ．ケース3)の代わりには $flag_n \neq 1$ を IF 文の条件部分に入れればよい．場所 δ では $flag_n > 0$ および $flag_n < \infty$ が必ず成り立つからである．以上により，繰り返しに関するフラグについて $flag_n := \infty$ をセットする必要はなくなる．

3.2 節の構文フラグのセットをいつでも行うのでなく，実際に必要になるものだけにすることも考えられる．

例えば，評価器の中では繰り返しフラグ $flag_n$ に対して，現在対象としている構文規則の中のすべての意味規則に $flag_n \neq 1$ の条件を用いてない場合，$flag_n$ のセットする必要がなくなる．

例 2.1 の属性評価規則から生成した意味処理文を以下に示す．

```
exp.val:=ID1.val;      /* δ=ID1 の post */
flag2:=1;              /*   ADD の pre */
flag2:=2;              /*   SUB の pre */
IF(flag2=1)exp.val:=exp.val+ID2.val;
                       /* δ=ID2 の post */
IF(flag2=2)exp.val:=exp.val-ID2.val;
                       /* δ=ID2 の post */
```

5. おわりに

以上，正規右辺属性文法の表記法およびそれに基づく1パス再帰降下属性評価器の生成法を述べた．

その属性評価器の生成系を使って PL/0 言語のコンパイラを試作した．正規右辺属性文法による記述では，非終端記号が 15 個，構文規則が 21 個，意味規則が 73 個である．それと対比して，普通の属性文法による記述では，非終端記号が 20 個，構文規則が 45 個，意味規則が 119 個である．また，三つの手続きからなる 37 行のプログラムを対象として，生成されたコンパイラの実行時間を測ったところそれぞれ 2.53 秒と 3.0 秒であった．正規右辺属性文法を用いて生成された評価器は構文解析フラグの扱いに若干のオーバヘッドがあるが，生成された評価（構文解析）手続きの数が普通より少なく，手続きの呼び出しにかかる時間が少ないので効率がよい．

以上の経験から，ここで提案した正規右辺属性文法による手続き型言語の記述が簡潔的で分かりやすいとともに効率もよいことが確かめられた．

謝辞 本研究を行うにあたり，多くのご助言をいただきました筑波大学プログラミング言語研究室の諸氏に深く感謝いたします．

参考文献

1) Bochmann, G. V.: Semantic Evaluation from Left to Right, *Comm. ACM*, Vol. 19, No. 2, pp. 55-62 (1976).
2) Bochmann, G. V. and Ward, P.: Compiler Writing System for Attribute Grammars, *Comput. J.*, Vol. 21, No. 2, pp. 144-148 (1978).
3) Jullig, R. K. and DeRemer, F.: Regular Right-Part Attribute Grammars, *SIGPLAN Notices*, Vol. 19, No. 6, pp. 171-178 (1978).
4) 片山卓也(訳)：アルゴリズム＋データ構造＝プログラム，p. 414, 日本コンピュータ協会 (1979).
5) Lewi, J., De Vlaminck, K., Huens, J. and Steegmans, E.: *A Programming Methodology in Compiler Construction Part 1: concepts*, p. 308, North-Holland Publ. Co., Amsterdam (1982).
6) Lorho, B. (ed.): *Methods and Tools for Compiler Construction*, p. 398, Cambridge Univ. Press, Cambridge (1984).
7) 中田育男：コンパイラ，p. 278, 産業図書，東京 (1981).
8) 中田育男，佐々政孝：意味規則つき正規表現とデータ構造直結型プログラムへの応用，コンピュータソフトウェア，Vol. 3, No. 1, pp. 47-56 (1986).
9) 佐々政孝：属性文法，コンピュータソフトウェア，Vol. 3, No. 4, pp. 73-91 (1986).
10) 佐々政孝，石塚治志，野口尚子，中田育男：1パス型属性文法による意味記述手法，第 31 回情報処理学会全国大会論文集，pp. 357-358 (1985).
11) 丁 亜希，中田育男，佐々政孝：正規右辺属性文法の再帰的下向き評価器，第 32 回情報処理学会全国大会論文集，pp. 361-362 (1986).
12) 渡辺美樹，中田育男：正規右辺属性文法の表記法及びその評価法，第 36 回情報処理学会全国大会論文集，pp. 823-824 (1988).
13) 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).

(昭和 63 年 8 月 26 日受付)
(昭和 63 年 11 月 14 日採録)

丁 亜希（正会員）

1955年生．1982年中国長沙工学院電子計算機系卒業．1986年筑波大学大学院修士課程修了．現在，同大学院博士課程工学研究科に在学中．プログラミング言語，属性文法，コンパイラの生成系の研究に興味を持っている．日本ソフトウェア科学会会員．

渡辺 美樹（正会員）

1963年生．1986年筑波大学第三学群情報学類卒業．1988年同大学院修士課程理工学研究科修了．同年富士ゼロックス(株)入社，システム技術センターシステムソフトウェア開発部所属．日本ソフトウェア科学会会員．

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年同大学院修士課程修了．1960～1979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)．日本ソフトウェア科学会，電子電報通信学会，ACM，IEEE 各会員．

佐々 政孝（正会員）

1948年生．1970年東京大学理学部物理学科卒業．1974年同理学系研究科博士課程中退，東京工業大学理学部情報科学科助手となる．1981年筑波大学電子情報工学系講師，1986年より同助教授．理学博士．プログラミング言語，属性文法，コンパイラ，コンパイラ生成系，プログラミング支援系に興味を持っている．1981年本学会論文賞受賞．ソフトウェア基礎論研究会幹事．ソフトウェア科学会，ACM，IEEE 各会員．