正規右辺文法の再帰降下パーサの効率のよい生成法†

丁 亜希†† 中田育男†††

正規右辺文法は構文規則の右辺の記述に文法記号の正規表現を許した文脈自由文法である．正規表現を使用することにより，構文規則の記述がコンパクトになり，文法の再帰的な構造を繰り返し構造に書き換えることができる．それにより，記述の便利さと読解性がよくなるだけでなく，効率のよい再帰降下パーサを作ることができる．本論文は正規右辺文法に対する再帰降下パーサの効率のよい生成法を示す．この方法では，入力文法の構造に対応する文法木を作り，その文法木から直接に再帰降下パーサを生成する．パーサ生成の過程中，必要な時だけ，必要となる FIRST 集合や FOLLOW 集合しか計算しない．それを実現するために，文法木のノード間の FIRST/FOLLOW 集合の計算上の依存関係を動的に解析する．また，同じ値を持つ集合は共通集合とされ，空間的，時間的な効率が上げられる．本方法で実現した再帰降下パーサ生成系は，入力される文法が ELL(1) 文法であるとき，確かに効率的である．本方法を Pascal の文法に適用した結果，FIRST/FOLLOW 集合計算にかかった時間が従来の方法と比べて 4〜9 倍ぐらい短くなった．

1. はじめに

正規右辺文法とは文脈自由文法の構文規則の右辺の記述に文法記号の正規表現を許したものであり，拡張文脈自由文法とも言われる．正規右辺文法は一つの記号の先読みにより LL 構文解析できるとき ELL(1) 文法と言われる．

再帰降下構文解析は直観的に分かりやすく，その効率も比較的よい[6]．一方，再帰降下構文解析できる文法のクラスは比較的小さく，特に左再帰的な文法構造を許さないという問題がある．しかし，正規右辺文法を使うと，その左再帰的な文法構造が簡単に繰り返し構造に書き換えられる場合が多い．また，一般に繰り返し構造の方が再帰的な構造より効率のよい再帰降下パーサが作られるので，正規右辺文法の再帰降下パーサは実用上有用である．

今まで，ELL(1) パーサの自動生成に関する研究がいろいろなされている[1)-3)]．生成したパーサだけでなく，パーサ生成の効率にも注目されている．特に，FIRST 集合および FOLLOW 集合の計算について効率よく求める方法が注目されているようである[3)]．

Heckmann[3)] が定点繰り返し (fixed-point iteration) を不要とするアルゴリズムを提案した．彼は入力の正規右辺文法の構造に対応する文法木を作り，木のノード上で FIRST 集合および FOLLOW 集合を定義した．その方法では木のノードが多いので集合の数も多かった．だが，実際にパーサの生成に使われる集合はその一部でしかなかった．

一方，正規右辺文法ではないが，文献 4) が BNF 記法の文脈自由文法に対して各生成規則の lookahead 集合を計算するために「要求駆動」(demand-driven) という方式を採用した．

我々も要求駆動の考えから出発して，正規右辺文法に対して文法木から直接にパーサを生成し，パーサ生成の過程中必要な時にだけ，必要となる FIRST 集合および FOLLOW 集合だけを計算する方法を考案した．それを実現するために，文法木のノード間の FIRST/FOLLOW 集合の計算上の依存関係を動的に解析する．また，同じ値を持つ集合を共通集合とし，空間的，時間的な効率を上げる方法を考案した．

2. 正規右辺文法に関する定義

定義 2.1：正規右辺文法 G は四つ組 (V_N, V_T, S, P) で定義される．V_N, V_T, S, P はそれぞれ非終端記号の集合，終端記号の集合，出発記号，右辺に正規表現の許される構文規則の集合である． ■

以後は特に断らない限り，$A, B \in V_N$, $a, b \in V_T$, $x, y, z \in V_T^*$, $V = V_T \cup V_N$, $u, v, w \in V^*$ を使うことにする．

定義 2.2：正規表現は次のように再帰的に定義される：

1. ε, a, A は正規表現である．

以下，$e, e_1, e_2, \cdots, e_n$ などを正規表現とする．

2. $e_1 e_2$ も正規表現である．これを連結構造 (conc) と呼ぶ．

† Efficient Generation of Recursive Descent Parsers for Regular Right Part Grammars by YAXI DING (Doctoral Program in Engineering, University of Tsukuba) and IKUO NAKATA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系

3. $(e_1|e_2)$ も正規表現である．これを選択構造 (alt) と呼ぶ．曖昧ではないとき，括弧を使わなくてもよい．

4. $[e]$, $\{e\}$ も正規表現である．これらをそれぞれオプション (opt)，繰り返し (rep) と呼ぶ．

5. 1-4 項以外のものは正規表現ではない． ■

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
$A\rightarrow u\|v$;	$A\rightarrow u$;
	$A\rightarrow v$;
$A\rightarrow u\{v\}w$;	$A\rightarrow uBw$;
	$B\rightarrow vB$;
	$B\rightarrow\varepsilon$;
$A\rightarrow u[v]w$;	$A\rightarrow uBw$;
	$B\rightarrow v$;
	$B\rightarrow\varepsilon$;

正規表現 e から導出できる $u\in V^*$ の集合を $R(e)$ と記述する．すなわち，$R(e)=\{u|e\overset{*}{\Rightarrow}u,\ u\in V^*\}$ である．正規表現 e に対して，FIRST 集合と FOLLOW 集合を次のように定義する．

定義 2.3：

$$\text{FIRST}(e)=\{a|w\overset{*}{\Rightarrow}au, w\in R(e)\subseteq V^*;\ \text{または，}\ w\overset{*}{\Rightarrow}a, a=\varepsilon\}$$ ■

定義 2.4：

$$\text{FOLLOW}(e)=\{a|S\overset{*}{\Rightarrow}uwav,\ w\in R(e)\subseteq V^*,\ a\in V_T\cup\{\$\}\}$$

ここで，$ は入力の終わりを示す記号とする． ■

定義 2.5：以下の二つの条件（いわゆる ELL(1) 条件）を満たす文法が ELL(1) 文法である．

① 構文規則の右辺に現れる正規表現 $e_1|e_2|\cdots|e_n$ に対して

$$\forall i,k\ \text{FIRST}(e_i)\cap\text{FIRST}(e_k)=\phi\quad i\neq k$$

かつ，$\varepsilon\in\text{FIRST}(e_i)$ のとき

$$\text{FIRST}(e_1|e_2|\cdots|e_n)\cap\text{FOLLOW}(e_i)=\phi$$

② 構文規則の右辺に現れる正規表現 $\{e\}$ に対して

$$\text{FIRST}(e)\cap\text{FOLLOW}(\{e\})=\phi$$ ■

ところで，オプション $[e]$ が $(e|\varepsilon)$ を意味するので，ELL(1) 条件の①によって $\text{FIRST}(e)\cap\text{FOLLOW}([e])=\phi$ であるが，ここで，そのオプション構造を以下のように拡張する．すなわち，$a\in\text{FIRST}(e)\cap\text{FOLLOW}([e])$ のとき，入力記号の a を $\text{FOLLOW}([e])$ のものとせず，e の先頭と考える．このような約束によって，一部のもともと LL(1) ではない言語に対して再帰降下構文解析法で解析可能になる．次の文法 G 1

#p1 : st→if|assign|ε

#p2 : if→IF cond THEN st [ELSE st]

#p3 : assign→ID=exp

は曖昧であり，

IF cond THEN IF cond THEN st ELSE st

の ELSE st はどの IF に対応するものか分からない．そのことは終端記号 ELSE∈FIRST(ELSE st)∩FOLLOW([ELSE st]) にも表れているが，構文解析中入力記号 ELSE に出会ったときこの ELSE を FIRST (ELSE st) のものと考えれば，Pascal 言語のようにこの ELSE はすぐ前の THEN と対応することになるわけである．上の例は本稿の ELL(1) 条件の定義により ELL(1) 文法である．

以下，パーサ生成系に入力される文法 G は次の条件を満足しているものとする．

1) G は ELL(1) 文法である．これは G に対して生成系が ELL(1) 条件を調べないという意味ではなく，非 ELL(1) の場合パーサの生成を中止するということである．同じ言語を記述するのに文法を ELL(1) 条件を満たすように書き直す方法がいろいろあるが，生成系はそれをしない．

2) 無意味な構文規則がない．これは二つの意味をもつ．一つは $\text{FIRST}(e)=\phi$ のような正規表現 e がない．このような e からは終端記号列を導出できないからである．もう一つは構文規則の右辺に出現しない開始記号以外の非終端記号がない．これは完全に必要ではない非終端記号である．

3. 文法木と再帰降下パーサ生成

入力文法の内部表現として，文献 3) のように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て一つの文法木を作る．

前節の文法 G1 から次の**図 1** のような三つの文法木が得られる．

文法木の各ノード t に以下の情報が入っていると

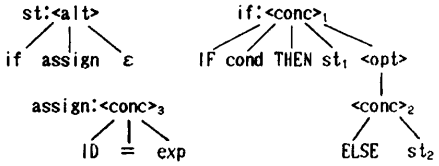

図 1 G1 の文法木
Fig. 1 Grammar trees of G1.

する．t. class が正規表現に対応してノードの種類を示すものであり，その値は {ε, term, non, alt, conc, rep, opt} 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．逆に t の親を t. parent とする．t が文法木のルートであるとき，その木に対応する構文規則の左辺の非終端記号を $L(t)$ と記し，構文規則の右辺に出現しているすべての $L(t)$ のノードを $t.p_1$, $t.p_2$, $\cdots t.p_m$ と記す．また，t が非終端記号のノードであるとき，$t.r$ でその非終端記号に対応する文法木のルートを示すことにする．t. first と t. follow がそれぞれ FIRST 集合と FOLLOW 集合へのポインタである．例えば，ノード t に対して FIRST(t) が FIRST 集合そのものを意味し，t. first が FIRST(t) へのポインタを意味する．

文法木の上で FIRST 集合が次のように再帰的に定義できる．

ノード t の class が

ε のとき　FIRST(t)={ε}

term のとき FIRST(t)={$M(t)$}

nonのとき　FIRST(t)=FIRST($t.r$)

conc のとき

FIRST(t)=(FIRST($t.c_1$)−{ε})∪⋯

∪(FIRST($t.c_{k-1}$)−{ε})∪FIRST($t.c_k$)

ここで k は ε∈FIRST($t.c_i$)　$i=1,\cdots,k-1$

ε∉FIRST($t.c_k$) なる k ($k\leqq n$)

あるいは $k=n$ (そのとき ε=FIRST(t) となることもある)

alt のとき

FIRST(t)=FIRST($t.c_1$)∪⋯∪FIRST($t.c_n$)

opt のとき　FIRST(t)={ε}∪FIRST($t.c$)

rep のとき　FIRST(t)={ε}∪FIRST($t.c$) ■

また，FOLLOW 集合が次のように再帰的に定義できる．

ノード t の class が

conc のとき FOLLOW($t.c_n$)=FOLLOW(t)

$k=1,\cdots,n-1$ に対して

ε∉FIRST($t.c_{k+1}$) の場合

FOLLOW($t.c_k$)=FIRST($t.c_{k+1}$)

ε∈FIRST($t.c_{k+1}$) の場合

FOLLOW($t.c_k$)=

{FIRST($t.c_{k+1}$)−{ε}}∪FOLLOW($t.c_{k+1}$)

alt のとき　FOLLOW($t.c_i$)=FOLLOW(t)

$i=1,\cdots,n$

opt のとき　FOLLOW($t.c$)=FOLLOW(t)

rep のとき　FOLLOW($t.c$)=FOLLOW(t)

t が文法木のルートであるとき

$L(t)$ が開始記号であれば

FOLLOW(t)=

{$} ∪FOLLOW($t.p_1$)∪⋯∪FOLLOW($t.p_m$)■

$L(t)$ が開始記号でなければ

FOLLOW(t)=FOLLOW($t.p_1$)∪⋯∪FOLLOW($t.p_m$) ■

再帰降下パーサの生成法は基本的に文献 1) に準じるが，文法木のノードに対して以下に示されるルールがある．そのルールに従って簡単に文法木からパーサを生成することができる．

パーサ生成ルール

生成規則 $A\rightarrow e$ に対して，文法木のルートを t としたとき，手続き proc_A( ) BEGIN $\Gamma(t)$ END を生成する．ここで，$\Gamma(t)$ は木ノード t から生成すべきパーサのパタンを示すものであり，**表1**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．

生成ルールの中で FIRST 集合を使うのは 5, 6, 7 番目のルールだけである．ELL(1) 条件を調べることは5と7番目のルールの所で行えばよい．5番目のルールでは ELL(1) 条件を調べるために ε∈FIRST

表 1 パーサ生成のパタン
Table 1 Patterns for parser generation.

t. class	パーサのパタン $\Gamma(t)$
1. ε	空文; 何も生成しない.
2. term	IF sym=$M(t)$ THEN getsym( ) ELSE error( );
3. non	CALL proc_$M(t)$( );
4. conc	$\Gamma(t.c_1)$; $\Gamma(t.c_2)$; ⋯; $\Gamma(t.c_n)$;
5. alt	CASE OF sym IN FIRST($t.c_1$): $\Gamma(t.c_1)$; FIRST($t.c_2$): $\Gamma(t.c_2)$; ⋯ FIRST($t.c_n$): $\Gamma(t.c_n)$; もし ε∉FIRST(t) ならば次の一行を加える OTHERWISE: error( ); ESAC
6. opt	IF sym IN FIRST($t.c$) THEN $\Gamma(t.c)$;
7. rep	WHILE sym IN FIRST($t.c$) DO BEGIN $\Gamma(t.c)$ END;

(t) のときだけ FOLLOW(t) が必要である．

したがって，パーサ生成の途中で alt，opt と rep の文法木のノードに出会った時必要に応じてそれぞれの FIRST 集合または FOLLOW 集合を計算すればよいのである．それらのノードに出会ったとき必要な FIRST/FOLLOW 集合は既に計算してあるかもしれないので，その場合はそれをそのまま使えばよい．まだ計算していないときだけ，集合を計算する手続きを呼び出して計算する．

4. 集合の計算

前節で文法木から再帰降下パーサを直接に作りながら必要になったとき FIRST 集合と FOLLOW 集合を計算するということを述べたが，本節ではそれらの集合の計算方法について説明する．集合計算の要求が発生したときの文法木のノードから出発してそのノードにつけられている FIRST 集合や FOLLOW 集合を正しくかつ効率的に計算できるかどうかが問題である．

それを解決する前に，まず文献 3) の FIRST 集合と FOLLOW 集合の計算方法の概要を説明する．

4.1 Heckmann の計算法

計算の順序は次の三つのステップからなっている．

step 1 どの FIRST 集合が ε を含むのかを判断する．

step 2 文法木のルートに関する FIRST 計算式を作る．
文法木のルートにつけられる FIRST 集合を計算する．
その他の文法木のノードにつけられる FIRST 集合を計算する．

step 3 文法木のルートに関する FOLLOW 計算式を作る．
文法木のルートにつけられる FOLLOW 集合を計算する．
その他の文法木のノードにつけられる FOLLOW 集合を計算する．

文法木のルートに関する FIRST/FOLLOW 計算式は下記のとおりである．

$$f(r)=G(r)\cup(\cup f(r'));$$
ただし $r'\in H(r)$;

ここで，f は FIRST 集合または FOLLOW 集合である（実際には step 1 の情報があるので FIRST 集合の代わりに FI′=FIRST-{ε} を計算する）．r，r' は文法木のルートであり，$H(r)$ は $f(r)$ を計算するために必要なルートの集合である．すなわち，$f(r)$ は $r'\in H(r)$ のそれに依存する．$f(r')$ は $f(r)$ より先に計算されなければならない．その依存関係は step 1 の情報を使って求められるが，サイクルになっているかも知れないので，これは文献 5) の深さ優先探索のアルゴリズムを利用して解決する．$G(r)$ は $f(r)$ を計算するとき参照される特定の集合である．それは f が FIRST 集合を示すとき，いくつかの終端記号のノード t の $\{M(t)\}$ からなるものであり，f が FOLLOW 集合を示すとき，いくつかの依存する FIRST 集合の和である．ルートにつけられる集合が既に計算されていることを前提としたら，その他のノードにつけられる集合の計算は一度行えば済む．したがって，定点繰り返しを避けて効率的に FIRST 集合と FOLLOW 集合を計算することができる．

4.2 要求駆動の計算法

ところで，上記の計算方法の中では文法木の各々のノード間の依存関係を求める方法が我々の要求駆動の計算に対しても有用であるが，その計算の順序は次の理由で不都合である．その理由の一つは集合の計算要求が出るノードは必ずしも文法木のルートに限らない．もう一つは必要な集合であるかどうかにかかわらず，あらかじめ文法木のルート間の依存関係を計算するのが無駄になるかもしれない．

FIRST(t) を計算するために，あるノード q の FIRST(q) の値を参照する場合がある．ここで，その参照関係を $t\downarrow q$ で記す．具体的に任意のノード t に対して，

t. class が

non のとき $t\downarrow t.r$;

conc のとき $t\downarrow t.c_1$; $t\downarrow t.c_2$; …; $t\downarrow t.c_k$;
ただし，$\varepsilon\in$FIRST($t.c_i$) $i=1,\cdots,k-1$
$\varepsilon\notin$FIRST($t.c_k$)
あるいは $k=n$

alt のとき $t\downarrow t.c_1$; $t\downarrow t.c_2$; …; $t\downarrow t.c_n$

opt のとき $t\downarrow t.c$;

rep のとき $t\downarrow t.c$;

である．ε と term の場合 FIRST(t) がそれぞれ {ε} と $\{M(t)\}$ になるので，t は他のノードに依存しない．

関係 $\downarrow$ の推移的閉包を $\downarrow^+$ で記す．すなわち，$t\downarrow^n q$ $(n>0)$ のとき $t\downarrow^+ q$ である．

入力される文法が ELL(1) であれば，$t\downarrow^+ t$ のようなノードが存在していないはずである．$t\downarrow^+ t$ があれば，文法の構文規則に左再帰的な構造が存在するので

ある．

$t \downarrow q$ のとき，FIRST(q) がまだ計算されてない場合，$t \mathrel{I} q$ で記し，関係 I の推移的閉包を $\mathrel{I}^+$ で記す．

```
手続き genfirst(t):
1  IF t.first=MARK THEN STOP;
   /* サイクルがある，したがって ELL(1) 文法でない */
2  t.first := MARK;
3  t I p なるすべての p について，FIRST(p) を先に計算するために genfirst(p) を呼び出す;
4  第3節の定義に従って FIRST(t) を計算する．t.first はその集合を指すようにする．ただし，t.first は t がεであるとき集合 {ε} を，t が term であるとき集合 {M(t)} をそれぞれ指すようにする．t が non であるとき，t.first := t.r.first，t が conc であり ε∉FIRST(t.c1) であるとき t.first := t.c1.first とする;
```

図 2 FIRST(t) を計算するアルゴリズム
Fig. 2 An algorithm to evaluate FIRST(t).

```
手続き genfollow(t):
 1  lowlink := t.number := counter;
 2  counter := counter+1;
 3  IF t I p のような p が存在しない THEN
 4      第3節の定義に従って FOLLOW(t) を計算する．t.follow はその集合を指
 5      すようにする．ただし，その集合がすでに計算されているある一つの集合に
 6      等しいとき t.follow はその集合を指すだけで済ませる;
 7  ELSE
 8    BEGIN
 9      t.follow := MARK;
10      PUSH(t);
11      FOR t I p なる各 p DO
12          IF (p.follow=nil) THEN
13              lowlink := MIN(lowlink, genfollow(p));
14          ELSE
15              IF p.follow=MARK かつ p.number<t.number THEN
16                  lowlink := MIN(lowlink, p.number);
17      IF lowlink=t.number THEN
18          IF stack の top=t THEN
19            BEGIN
20              POP(q);
21              4行目～6行目のように t.follow を設定する;
22            END
23          ELSE
24            BEGIN
25                新集合（初期値は空集合）をつくり，それをポインタ newset で
26                指す;
27                REPEAT
28                    POP(q); q.follow := newset;
29                    第3節の定義に従って FOLLOW(q) の計算に参照される集
30                    合（今回の REPEAT 文によって計算される FOLLOW 集
31                    合を除く）を新集合に加える;
32                UNTIL q=t;
33            END
34    END
35  return (lowlink);
```

図 3 FOLLOW(t) を計算するアルゴリズム
Fig. 3 An algorithm to evaluate FOLLOW(t).

ノード t の場所で FIRST(t) の計算要求が出されたとき $t \mathrel{I} q$ の FIRST(q) の計算手続きを先に呼び出せればよい．この I 関係は FIRST 集合計算の都合によって動的に決められる．$t \mathrel{I}^+ t$ のような関係を発見するにはマークを使う．FIRST 集合を計算する手続きを**図 2** に示す．

各ノードの first の初期値を nil とする．あるノード t の FIRST(t) の計算が必要になったとき genfirst(t) が呼ばれる．計算の進行に従って集合を作り，ポインタを立てる．集合のコピーを避けるため，計算される集合の値が参照先の集合の値と一致する場合ポインタが共通の集合を指すようにする．

一方，FOLLOW(t) を計算するため，あるノード q の FOLLOW(q) の値を参照する場合もある．それを $t \uparrow q$ で表す．関係 ↑ の推移的閉包を $\uparrow^+$ で記す．ノード t に対して，t が文法木のルートであるとき，$t \uparrow t.p_1$; $t \uparrow t.p_2$; …; $t \uparrow t.p_m$ である．また，t.class が

conc のとき $t.c_n \uparrow t$;
　$k=1,2,\cdots,n-1$ に対して
　$\varepsilon \in$FIRST($t.c_{k+1}$) ならば
　　$t.c_k \uparrow t.c_{k+1}$
alt のとき $t.c_i \uparrow t$; $i=1,\cdots,n$
opt のときあるいは rep のとき $t.c \uparrow t$;

である．残念ながら，入力される文法が ELL(1) であっても，$t \uparrow^+ t$ のノードが存在するかもしれない．このようなサイクル中のノードは明らかに同じ値の FOLLOW 集合を持つべきである．なぜならば，$t \uparrow^+ q$ のとき，FOLLOW(t)⊆FOLLOW(q) であるし，$q \uparrow^+ t$ のときも，FOLLOW(q)⊆FOLLOW(t) であるから，$t \uparrow^+ q$ と同時に $q \uparrow^+ t$ も成り立つとき FOLLOW(t)=FOLLOW(q) である．したがって，同一サイクルにあるノードに対して共通の FOLLOW 集合を持たせるようにすれば，集合間のコピーや集合和の運算は避けられる．

$t \uparrow q$ のとき，FOLLOW(q) がまだ計算されてない場合，$t \uparrow q$ で記し，関係 $\uparrow$ の推移的閉包を $\uparrow^+$ で記す．関係 $\uparrow$ は動的に計算できるが，サイクルの発見にはグラフの強連結成分を見つければよい．そのために文献5）の深さ優先探索アルゴリズムを利用する．そのため一つの counter と各ノードにつけられる整数変数 number と lowlink が必要である．counter は初期値が0であり，新しいノードを探索するごとに1増やす．number はこのノードの探索された順番を示す番号である．lowlink は強連結成分（サイクル）の根を見つけるため定義される変数である．深さ優先探索アルゴリズムの詳細は文献 5) を参照されたい．ただし，ここではノード上につけられる情報を減らすため，lowlink を探索手続きの返す値とする．各ノードの FOLLOW の初期値は nil とする．FOLLOW 集合を計算する手続きは**図3**に示す．

4〜6 行目では，参照されるべき集合が既に用意されてしまったので FOLLOW(t) を計算してよい．$t \uparrow p$ であるのは $t \uparrow p$ でありながら p.follow が nil あるいは MARK であることにより分かる．17 行目では，lowlink$\neq t$.number となる場合，t を含むサイクルがまだ全部捜し出されてないので，lowlink の値だけを返してよい．t はまだ stack にあり，FOLLOW(t) は t を含むサイクル中のノードが全部探索されたとき計算される．20〜21 行目では，t (=q) は他のノードとサイクルにならないので 4〜6 行目と同じように処理すればよい．25〜32 行目では同一サイクル中のノード（stack の中に t と t の上にあるノード）に共通集合を持たせるようにする．REPEAT 文の中で不必要な集合和演算を避けるため，今回の genfollow(t) によって計算される FOLLOW 集合を新集合に追加しない．それは follow=newset (POP したばかりのノード）であるかあるいは follow=MARK（この REPEAT 文の終わる前に必ず POP されるノード）であることによって分かる．

以下に第3節の図1（$G1$ の文法木）を例としてこのアルゴリズムによる FOLLOW 集合計算の過程を簡単に説明する．

パーサ生成の途中，最初は st の 〈alt〉 の所で FIRST 集合の計算要求が発生する．その後，ε が FIRST 集合に現れるので，〈alt〉 の FOLLOW 集合計算の要求が出て，手続き genfollow(〈alt〉) に入る．$L(t)=st$ で，最初は $t \uparrow st_1$ と $t \uparrow st_2$ であるので，genfollow(st_1) と genfollow(st_2) を順次呼び出す．genfollow(st_1) の中で genfollow(〈opt〉) を，genfollow(〈opt〉) の中で genfollow(〈conc〉$_1$) を呼び出し，そこから genfollow(if) を呼び出す．また，genfollow(st_2) の中で genfollow(〈conc〉$_2$) を呼び出す．その結果 if, 〈conc〉$_1$, 〈opt〉, st_1 の lowlink は 0, 〈conc〉$_2$, st_2 の lowlink は2となる．genfollow(〈alt〉) に戻ってきたら，stack の中には下からの順で 〈alt〉, st_1, 〈opt〉, 〈conc〉$_1$, if, st_2, 〈conc〉$_2$ が入っている．すなわち，これらの FOLLOW 集合の依存関係はサイクルをなす．今度は，図3の 25 行目に入り，これらのノードの follow が一つの新集合を指すようにする．集合に実際に加える演算は二か所だけで行われる．すなわち，st_1 を POP した際 FIRST (〈opt〉)-{ε} = {ELSE} を新集合に加え，〈alt〉 を POP した際 {$} を新集合に加える．結局，17 個のノードを持つ $G1$ の文法木に対して，7個のノードに付けられる FOLLOW 集合だけが計算された．また，FOLLOW 集合の実体は {ELSE, $} 一つしかない．

5. 評価実験

上述の方法で SUN 3 上C言語で再帰降下パーサの生成系を実現した．それと比べるために，文献 3) の方法および定点繰り返しを要する方法でも生成系を作った．そして，Pascal 言語の文法を入力としてそれぞれの生成効率を調べてみた．それは非終端記号が 36 個，終端記号が 69 個，文法木のノードが 466 個の中規模の文法であった．一つの FIRST/FOLLOW 集合は四つの unsigned int 型の整変数から構成し，各終端記号をその中の 1 bit と対応させた．これが1であるときこの集合に属するとし，0であるときこの集合に属しないとする．一つの int が 32 bit であるから，128 個までの終端記号を持つ文法が扱える．結果を**表2**に示す．

その中の集合計算時間は FIRST/FOLLOW 集合の計算時間であり，本方法がその他の方法に比べて 4〜9 倍速くなっている．パーサの生成にかかったそれ以外の時間は三つの方法とも同じくほぼ1秒であった．

表2 三つの方法の結果の比較
Table 2 Results of the three methods.

	FIRST 集合の数	FOLLOW 集合の数	集合計算時間 (秒)
要求駆動	130	117	0.06
文献 3)	226	462	0.26
定点循環	226	466	0.52

したがって，集合の計算にかかった時間がパーサ全体の生成にかかった時間に対してそれぞれ 5.7%，20.6%，34.2% を占めていることが分かる．なお，試作したパーサ生成系の大きさはソースではそれぞれ 1127 行，1339 行，1147 行であり，コンパイルした後ではそれぞれ 40 KB，49 KB，40 KB であった．

6. おわりに

正規右辺文法の再帰降下パーサの効率よい生成法を述べた．これは要求駆動を発想としたものである．入力の文法が ELL(1) 文法であるとき確かに効率的である．実際に ELL(1) 文法の構造は非常に分かりやすいので，ELL(1) 条件を満たす文法を書くのは難しいことではないと思われる．なお，本方法での FIRST 集合の計算法を FOLLOW 集合計算法のように直せば，LR 文法のように左再帰的な構造の許される文法の FIRST 集合計算にも利用できる．

本報告には構文誤りの回復問題を触れなかった．誤り回復の方法がいろいろあって，その中に FOLLOW 集合を利用する方法もある．そのような方法を採用するときも要求駆動により必要な FOLLOW 集合を計算すればよい．

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電子・情報工学系助教授佐々政孝先生と同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) 中田育男：コンパイラ，p. 278，産業図書，東京 (1981).
2) Lewi, J., de Vlaminck, K., Huens, J. and Steemans, E.: *A Programming Methodology in Compiler Construction Part 1: concept*, p. 308, North-Holland Publ. Co., Amsterdam (1982).
3) Heckmann, R.: An Efficient ELL(1)-Parser Generator, *Acta Inf.*, Vol. 23, pp. 127-148 (1986).
4) Dwyer, B.: Improving Gough's LL(1) Look-ahead Sets Generator, *SIGPLAN Notices*, Vol. 20, No. 11, pp. 27-29 (1985).
5) Tarjan, R.: Depth-first Search and Linear Graph Algorithms, *SIAM J. Comput.*, Vol. 1, pp. 146-160 (1972).
6) Gerardy, R.: Experiment Comparison of Some Parsing Methods, *SIGPLAN Notices*, Vol. 22, No. 8, pp. 79-88 (1987).

(昭和 63 年 4 月 18 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．

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動務．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，電子情報通信学会，ACM，IEEE 各会員．