コンパイラ講義ノート

国島丈生

2005-06-08

練習問題の解答

1. 次の文法を考える。

$$S \to A1B$$
$$A \to 0A|\epsilon$$
$$B \to 0B|1B|\epsilon$$

この文法に対し、文字列 00101 の最左導出、最右導出を示せ。

$$S \Rightarrow A1B \Rightarrow 0A1B \Rightarrow 00A1B \Rightarrow 001B \Rightarrow 0010B \Rightarrow 00101B \Rightarrow 00101$$
$$S \Rightarrow A1B \Rightarrow A10B \Rightarrow A101B \Rightarrow A101 \Rightarrow 0A101 \Rightarrow 00A101 \Rightarrow 00101$$

2. 算術式に対する文法

$$E \to E + T|T$$
$$T \to T * F|F$$
$$F \to (E)|\mathbf{id}$$

を、左再帰でないように変形せよ。

$$E \to TE'$$
$$E' \to +TE'|\epsilon$$
$$T \to FT'$$
$$T' \to *FT'|\epsilon$$
$$F \to (E)|\mathbf{id}$$

1 下向き構文解析

下向き構文解析は解析木を根から葉に向かって構築していく構文解析手法であり、入力記号列の最左導出を見つけること、と考えてよい。

構文解析の視点からみると、一般的な文脈自由文法には次のような問題点がある。

1. **左再帰**:左再帰文法から再帰下降構文解析ルーチンを作ると、停止しない再帰関数ができてしまう。(前々回資料、前回資料 1.5 節参照)

2. **後戻り**:文法によっては、解析に失敗したときに記号列をいくらか戻し、処理をやり直さなければならないことがある。後戻りは一般に処理コストが高く、なるべく避けるのが望ましい。

例 1 文法 $S \to aBd, B \to b|bc$ に対し、再帰下降構文解析を試みる。この文法に対する構文解析プログラムは次のようになる。

```
void S() { match(a); B(); match(d); }
void B() { match(b); または { match(b); match(c); } }
```

ここで match() は前々回の資料で示した関数である。また、「または」と書いた部分は「match(b) が失敗したら match(b); match(c);」という意味である[a]。
この文法により記号列 $abcd$ を構文解析する過程は次のようになる。

1. S() を呼び出す。
2. S() の中から B() を呼び出す。
3. B() の中から match(b) を呼び出す。この結果、文 abd が得られるが、これは $abcd$ と一致しないため、構文解析は失敗であり、入力記号 b を読む前の状態に後戻りし、B() に戻る。
4. B() の中から match(b); match(c); を呼び出す。この結果、文 $abcd$ が得られ、構文解析は成功する。

この文法では、左端の括り出しを行うことで後戻りが回避できる。つまり文法を $S \to aBd, B \to b(c|\epsilon)$ と書き直す。すると構文解析プログラムは次のようになり、後戻りが発生しない。

```
void S() { match(a); B(); match(d); }
void B() { match(b); { if (lookahead == 'c') match(c); } }
```

[a] 後で述べる通り、通常はこのような後戻りを含むプログラムは作らないため、分かりやすさを重視して、あえて日本語で示した。

例 2 文法 $S \to aBc, B \to b(c|\epsilon)$ について、記号列 abc を再帰構文解析してみよう。

1. S() を呼び出す。
2. S() の中で B() を呼び出す。
3. B() の中で match(b) を呼び出し、次いで match(c) を呼び出す。ここまでは成功である。
4. S() に戻り、match(c) を呼び出す。しかし、このとき入力記号は尽きているので、構文解析は失敗であり、3 に後戻りし、ϵ に相当する処理を試みる。つまり、何もせず、S() に戻る。
5. match(c) を呼び出す。この結果、構文解析は成功である。

この後戻りは、左端の括り出しでは解決できない。

2 LL(1) 文法

後戻りのない再帰構文解析 (**予測型構文解析**) を実現するため、文脈自由文法にいくらか制限を加えてみる。これを **LL(1) 文法**といい、多くのプログラミング言語の文法はこの範囲に収まることが知られている。

2.1 FIRST と FOLLOW

まず FIRST と FOLLOW という 2 つの関数を導入しておこう。

任意の記号列 α に対し、α から導出される記号列の中で先頭に現れる終端記号の集合を FIRST(α) とする。ただし $S \stackrel{*}{\Rightarrow} \epsilon$ の場合は ϵ も FIRST(α) に含める。

非終端記号 A に対し、文形式の中で、A のすぐ後ろに現れることのある終端記号の集合を FOLLOW(A) とする。A が文形式の一番後ろになることがある場合には、記号列の末尾を表す特殊な終端記号 \$ を FOLLOW(A) に含める。

例 3

$$
\begin{aligned}
E &\to TE' \\
E' &\to +TE'|\epsilon \\
T &\to FT' \\
T' &\to *FT'|\epsilon \\
F &\to (E)|\mathbf{id}
\end{aligned}
$$

この文法に対し、FIRST, FOLLOW の一部を求めると次のようになる。

$$
\begin{aligned}
&\mathsf{FIRST}(E) = \mathsf{FIRST}(T) = \mathsf{FIRST}(F) = \{(, \mathbf{id}\} \\
&\mathsf{FIRST}(E') = \{+, \epsilon\} \\
&\mathsf{FIRST}(T') = \{*, \epsilon\} \\
&\mathsf{FOLLOW}(E) = \mathsf{FOLLOW}(E') = \{), \$\} \\
&\mathsf{FOLLOW}(T) = \mathsf{FOLLOW}(T') = \{+,), \$\} \\
&\mathsf{FOLLOW}(F) = \{+, *,), \$\}
\end{aligned}
$$

2.2 LL(1) 文法

文法 G の生成規則のうち、$A \to \alpha|\beta$ の形のものについて常に次の性質が成立するとき、G を **LL(1) 文法**という。

1. α と β は同じ終端記号で始まる記号列を導出することがない。

$$(\mathsf{FIRST}(\alpha) - \{\epsilon\}) \cap (\mathsf{FIRST}(\beta) - \{\epsilon\}) = \emptyset$$

2. α と β のうち、ϵ を導出するのはたかだか一方だけである。
3. $\beta \stackrel{*}{\Rightarrow} \epsilon$ のとき、α からは FOLLOW(A) に含まれる終端記号で始まる記号列は導出されない。つまり

$$\beta \stackrel{*}{\Rightarrow} \epsilon\text{のとき } \mathsf{FIRST}(\alpha) \cap \mathsf{FOLLOW}(A) = \emptyset$$

$\alpha \stackrel{*}{\Rightarrow} \epsilon$ のときも同様。

例 4 例 3 の文法では

$$\mathsf{FIRST}(+TE') = \{+\},\ \mathsf{FOLLOW}(E') = \{), \$\}$$
$$\mathsf{FIRST}(*FT') = \{*\},\ \mathsf{FOLLOW}(T') = \{+,), \$\}$$
$$\mathsf{FIRST}((E)) = \{(\},\ \mathsf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$$

したがってこの文法は LL(1) である。

与えられた文脈自由文法から (1) 曖昧性を取り除き (2) 左再帰を取り除き (3) 左端の括り出しを行い、LL(1) 文法になれば、前々回に示した方法で予測型構文解析プログラムが必ず作れる。

2.3 FIRST の計算方法

1. 終端記号 a について $\mathsf{FIRST}(a) = \{a\}$
2. 生成規則 $X \to a\alpha$ について、a を $\mathsf{FIRST}(X)$ に加える。
3. 生成規則 $X \to \epsilon$ について、ϵ を $\mathsf{FIRST}(X)$ に加える。
4. 生成規則 $X \to Y_1 Y_2 \cdots Y_n$ について
   - Y_1 が ϵ を導出しないならば $\mathsf{FIRST}(Y_1)$ を $\mathsf{FIRST}(X)$ に加える
   - $Y_1 \stackrel{*}{\Rightarrow} \epsilon$ であれば、$\mathsf{FIRST}(Y_1) - \{\epsilon\}$ を $\mathsf{FIRST}(X)$ に加える。さらに Y_2 が ϵ を導出しないならば、$\mathsf{FIRST}(Y_2)$ を $\mathsf{FIRST}(X)$ に加え、$Y_2 \stackrel{*}{\Rightarrow} \epsilon$ ならば $\mathsf{FIRST}(Y_2) - \{\epsilon\}$ を $\mathsf{FIRST}(X)$ に加える。(以下同様)
   - $Y_1 Y_2 \cdots Y_n \stackrel{*}{\Rightarrow} \epsilon$ ならば、ϵ を $\mathsf{FIRST}(X)$ に加える
5. $\mathsf{FIRST}(X_1 X_2 \cdots X_n)$ は 4 と同様に計算

2.4 FOLLOW の計算方法

1. $\mathsf{FOLLOW}(S)$ に \$ を加える。
2. どの FOLLOW にも記号が追加されなくなるまで、以下を繰り返す。
   - $A \to \alpha B \beta$ について、$\mathsf{FIRST}(\beta) - \{\epsilon\}$ を $\mathsf{FOLLOW}(B)$ に加える。
   - $A \to \alpha B \beta$、かつ $\mathsf{FIRST}(\beta)$ が ϵ を含んでいるとき、$\mathsf{FOLLOW}(A)$ を $\mathsf{FOLLOW}(B)$ に加える。
   - $A \to \alpha B$ のとき、$\mathsf{FOLLOW}(A)$ を $\mathsf{FOLLOW}(B)$ に加える。

3 LL(1) でない文法の構文解析

LL(1) でない文法に対する一般的な構文解析手法は、本講義の範囲を越えるので省略する。ただし、ある種の LL(1) でない文法では、生成規則の適用順に優先度を設けることで予測型構文解析が行える場合がある。

例 5 C 言語の if 文を一般化した次の文法を考える。

$$S \to iES|iESeS|a$$
$$E \to b$$

これに対し左端の括り出しを行うと、次のような文法が得られる。

$$S \to iESS'|a$$
$$S' \to eS|\epsilon$$
$$E \to b$$

これでもまだ LL(1) 文法ではない。実はこの文法は、どう変形しても LL(1) にはならないことがわかっている。しかし、$S' \to eS$ を $S' \to \epsilon$ より優先して適用することにすれば、予測型構文解析が行える。S' に対応する関数は次のようになる。

```
void S_dash() { match(e); S(); }
```

上で示した優先度は、前回述べた規則「各 else は前のほうにある未対応の then 文のうち、もっとも近いものと対応する」と同じ働きをしている。

練習問題

1. 次の文法が LL(1) 文法であることを示せ。

$$S \to AaAb|BbBa$$
$$A \to \epsilon$$
$$B \to \epsilon$$