コンパイラ講義ノート

国島丈生

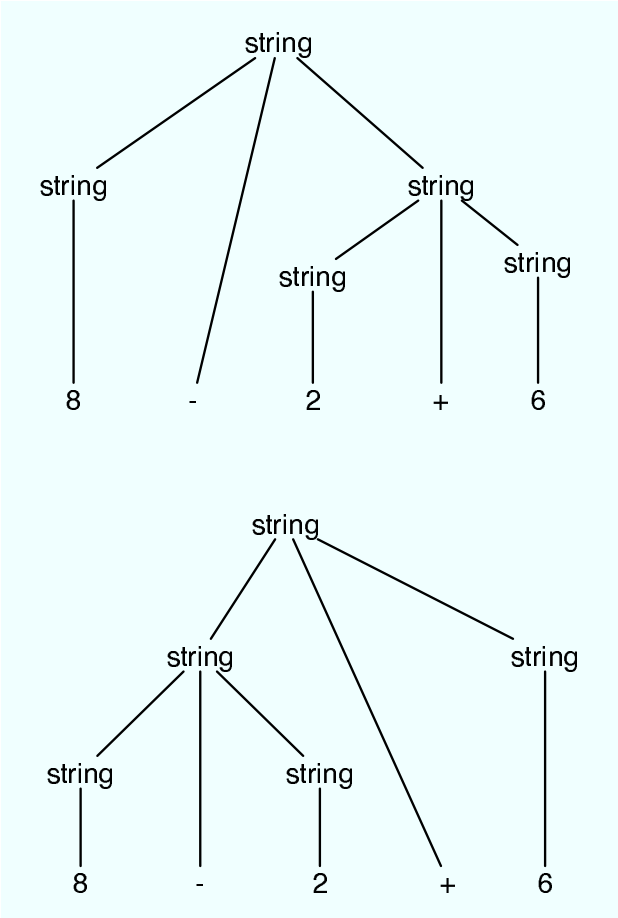
2005-06-01

練習問題の解答

1. 文脈自由文法

$$string \rightarrow string + string | string - string | 0|1|2|3|4|5|6|7|8|9$$

について、トークン列 $8-2+6$ の解析木を示せ。可能なすべての場合を示し、自然な式の意味になるものはどれか示せ。



$+, -$ は左結合の演算子であり、括弧がなければ式の左から計算していくのが正しい意味になる。したがって、下側の解析木のほうが自然である。このように、文法によっては、同じ記号列に対して解析木が 2 通り以上考えられる場合がある。このような文法を**曖昧である**という。

2. 文脈自由文法 $S \to 0S1|01$ はどのような言語を表すか。
   0 が n 個 $(n \geq 1)$ 並んだ後に 1 が n 個並んだような文字列すべてからなる言語。$0, 1$ をそれぞれ $\{, \}$ に置き換えてみると、これは対応関係のとれた括弧の入れ子を表す。この言語は正則言語ではない例として有名である。つまり、この言語は正則表現では書けず、またこの言語を受理する有限オートマトンも作れない。

1 文脈自由文法

文脈自由文法の定義は前回述べた。

例 1 次の文法は、簡単な算術式を定義するものである。

$$\begin{aligned} expr &\to expr\ op\ expr \mid (\ expr\ ) \mid -\ expr \mid \mathbf{id} \\ op &\to + \mid - \mid * \mid / \end{aligned}$$

以降の説明では、次のように記法を決めておくことにする。

1. 前のほうの英小文字 $(a, b, c, \cdots)$、演算子記号 $(+, -, \cdots)$、括弧やコンマなどの区切り記号、数字、太字の記号列は終端記号を表す。
2. 前のほうの英大文字 $(A, B, C, \cdots)$、S (開始記号)、英小文字のイタリック体の名前 ($expr$, $stmt$ など) は非終端記号を表す。
3. 後ろのほうの英大文字 $(X, Y, Z, \cdots)$ は非終端記号または終端記号を表す。
4. 後ろのほうの英小文字 $(x, y, z, \cdots)$ は終端記号列を表す。
5. 小文字のギリシャ文字 $(\alpha, \beta, \gamma, \cdots)$ は、記号列 (終端記号と非終端記号が混ざっていてよい) を表す。
6. 最初に現れる生成規則の左辺を開始記号とする。

1.1 導出

文法の言語の定義方法にはいくつかあるが、その 1 つに「生成規則を書き換え規則と考える」というものがある。つまり、開始記号 S を $S-$ 生成規則の右辺の記号列 α で置き換え、さらに α に含まれる非終端記号を、その記号の生成規則の右辺で置き換える。この操作を、記号列が終端記号のみになるまで繰り返す。このようにして得られる終端記号列すべての集合を、その文法の**言語**という。

例 2 文法 $E \to E+E|E*E|(E)|-E|\mathbf{id}$ において、E を右辺の記号列、例えば $-E$ で置き換えることができる。このとき、E から $-E$ を**導出する**といい、$E \Rightarrow -E$ と表す。□

もう少し形式的に書くと次のようになる。生成規則 $A \to \gamma$ を記号列 $\alpha A\beta$ に適用して書き換えを行うと $\alpha\gamma\beta$ となる。このとき $\alpha A\beta \Rightarrow \alpha\gamma\beta$ と表す。$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ のとき、$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ と書く。文法 G の開始記号を S とするとき、G の言語 $L(G) = \{w|S \overset{*}{\Rightarrow} w\}$ である。

$S \overset{*}{\Rightarrow} \alpha$ のとき、α を**文形式**という。とくに α が終端記号のみからなるとき、**文**という。

1.2 最左導出、最右導出

例 2 の文法において、記号列 $-(\mathbf{id}+\mathbf{id})$ を導出する方法はいくつかある。

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

つまり、文形式の中でどの非終端記号を置き換えるかによって、導出の様子が変わってくる。これらの導出の中で、必ず文形式の一番左の非終端記号を置き換えるような導出を**最左導出**、一番右の非終端記号を置き換えるような導出を**最右導出**という。

1.3 解析木と導出

解析木は、導出の様子を (順番を無視して) 図示したものと考えられる。したがって、どの解析木も必ず 1 つの最左導出 (または最右導出) と対応づけられる。

1.4 曖昧な文法

1 つの文に対して 2 つ以上の解析木が作れる文法を**曖昧である**という。つまり、曖昧な文法では 1 つの文に対して最左導出 (または最右導出) が 2 つ以上存在する。構文解析では 2 つ以上の解析木ができてしまうのは困るので、曖昧でない文法に変換したり、ある種の規則を設けて複数の解析木から 1 つを選ぶようにしたりする。

曖昧さを解消する方法は、あまり体系的にはなっていない。例えば、前回の練習問題 (1) の文法は曖昧であるが、前回の講義中に示した次の文法に変形すれば曖昧さは解消できる。

$$\begin{aligned} list &\to list + digit \\ list &\to list - digit \\ list &\to digit \\ digit &\to 0|1|2|3|4|5|6|7|8|9 \end{aligned}$$

もう 1 つ例を挙げよう。

$$stmt \to \mathbf{if}\ (expr)\ stmt$$

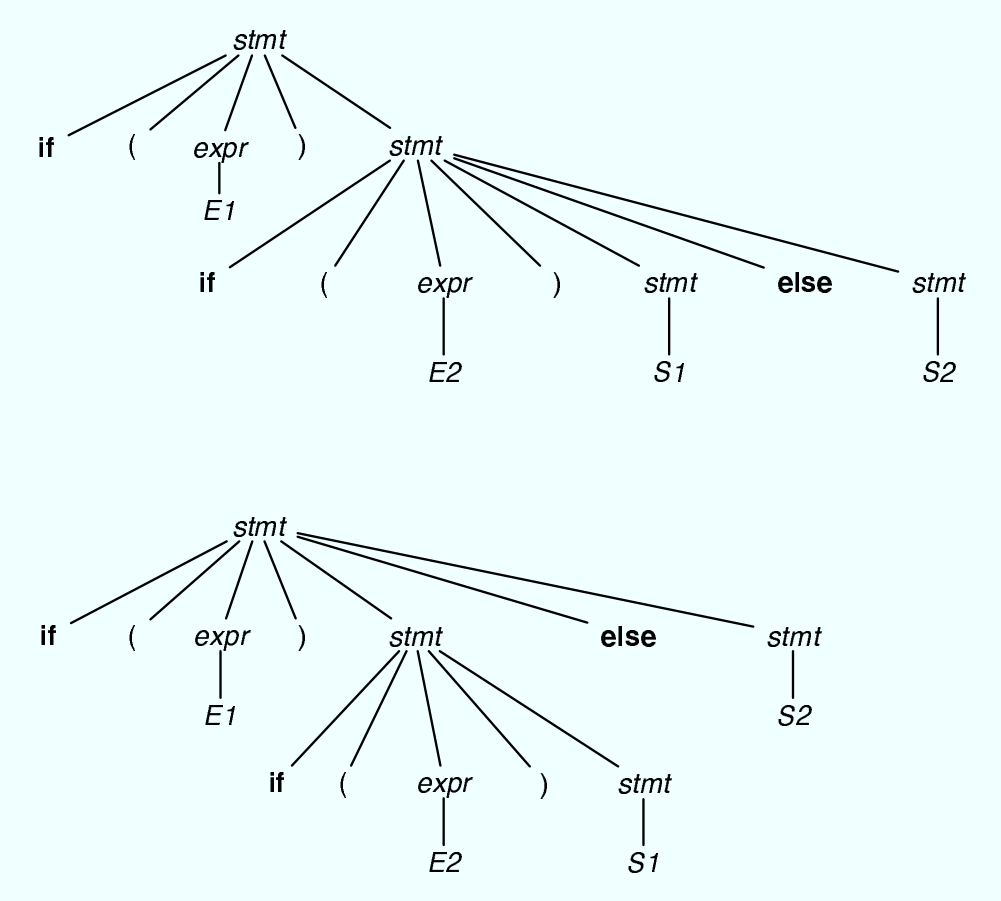


図 1　曖昧な文法に対する 2 つの解析木

$$\mid \textbf{if}\ (expr)\ stmt\ \textbf{else}\ stmt$$
$$\mid \cdots$$

上に示したのは if 文を表す文法である。しかし、図 1 で分かるように、この文法は次の文に対して 2 つの解析木を生成するので、曖昧である。

$$\textbf{if}\ (E_1)\ S_1\ \textbf{else if}\ (E_2)\ S_2\ \textbf{else}\ S_3$$

このような条件文の文法では普通、「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」という規則を設け、曖昧性を解消する。つまり図 1 の解析木のうち、上のほうを採用する。次のように文法を書き換えることで、曖昧さが解消できる。

$$\begin{aligned} stmt &\rightarrow matched\_stmt \\ &\mid unmatched\_stmt \\ matched\_stmt &\rightarrow \textbf{if}\ (expr)\ matched\_stmt\ \textbf{else}\ matched\_stmt \\ &\mid \cdots \end{aligned}$$

$$
\begin{aligned}
\mathit{unmatched\_stmt} \rightarrow\ & \textbf{if}\ (\mathit{expr})\ \mathit{stmt} \\
\mid\ & \textbf{if}\ (\mathit{expr})\ \mathit{matched\_stmt}\ \textbf{else}\ \mathit{unmatched\_stmt}
\end{aligned}
$$

1.5 左再帰

$A \stackrel{*}{\Rightarrow} A\alpha$ という導出が存在するとき、この文法は**左再帰**であるという。予測型構文解析法では非終端記号それぞれに (再帰) 関数を定義し、生成規則の通りに関数を呼び出すことで構文解析を進めていく。したがって、左再帰の文法では停止しない再帰呼び出しが起こってしまう。つまり、左再帰の文法は、予測型構文解析では扱えない[*1]。

もっとも簡単な左再帰の文法は、$A \rightarrow A\alpha$ という形の生成規則を持つものである。ここでは、この形の生成規則を含む左再帰文法を、左再帰ではない文法に変換する方法のみ示す。左再帰の文法の A-生成規則は、一般に

$$A \rightarrow A\alpha_1 | A\alpha_2 | \cdots | A\alpha_m | \beta_1 | \beta_2 | \cdots | \beta_n$$

と書ける。ここで β_i は A 以外の記号で始まる記号列である。このとき、この A-生成規則を次のように変形する。

$$
\begin{aligned}
A &\rightarrow \beta_1 A' | \beta_2 A' | \cdots | \beta_n A' \\
A' &\rightarrow \alpha_1 A' | \alpha_2 A' | \cdots | \alpha_m A' | \epsilon
\end{aligned}
$$

この文法は、変形前のものと同じ言語を生成し、かつ左再帰を含まない。

1.6 左端の括り出し

先に挙げた文法の一部について、再び考える。

$$
\begin{aligned}
\mathit{unmatched\_stmt} \rightarrow\ & \textbf{if}\ (\mathit{expr})\ \mathit{stmt} \\
\mid\ & \textbf{if}\ (\mathit{expr})\ \mathit{matched\_stmt}\ \textbf{else}\ \mathit{unmatched\_stmt}
\end{aligned}
$$

トークン **if** を読み込んだとき、どちらの生成規則を用いて *unmatched_stmt* を書き換えればよいかはまだ判断できない。ずっとトークン列を読んでいって、**else** が出てきて初めて、下の生成規則を適用すればよいということが分かる。このとき、処理の逆戻り (バックトラック) が発生する可能性が高く、効率が悪くなる。

このような場合には、次のように左端を括り出すと、処理の逆戻りが発生しなくなる。

$$
\begin{aligned}
\mathit{unmatched\_stmt} &\rightarrow \textbf{if}\ (\mathit{expr})\ \mathit{unmatched\_stmt}' \\
\mathit{unmatched\_stmt}' &\rightarrow \mathit{stmt} \\
&\quad \mid\ \mathit{matched\_stmt}\ \textbf{else}\ \mathit{unmatched\_stmt}
\end{aligned}
$$

[*1] 実は、一般に下向き構文解析法はすべて、左再帰の文法を扱うことができない。

1.7 予測型構文解析

文法 G について予測型構文解析を行うには、G が次のような制限を満たしていなければならない。

1. G は曖昧でなく、かつ左再帰ではない。
2. 生成規則 $A \to \alpha_1|\alpha_2|\cdots|\alpha_n$ について、現在の入力記号 a のみからどの生成規則で A を書き換えればよいか、一意に決定できなければならない。

左端の括り出しとは、2 番目の制限を満たすように文法を変形することに他ならない。ただし、左端の括り出しだけでは不十分な場合があり、もう少し精密に議論しなければならない。この部分は次週以降に譲る。

2 文脈自由文法の限界

言語 $L = \{wcw|w \in (a|b)^*\}$ は「c の前後に同じ文字列 w が出現するような言語」であり、プログラミング言語での「変数を使うときには、それより前に宣言をしておかなければならない」という決まりを抽象化したものと言える。ところが、実は L は文脈自由文法では表現できない。つまり、変数を使う前に宣言されているか検査するのは、構文解析フェーズでは行うことができない。実際、この検査は、意味解析フェーズで行われている。

このように、プログラミング言語の制限には構文解析フェーズで検査できないものがあり、それらは意味解析フェーズで検査されることが多い。

練習問題

1. 次の文法を考える。

$$
\begin{aligned}
S &\to A1B \\
A &\to 0A|\epsilon \\
B &\to 0B|1B|\epsilon
\end{aligned}
$$

   この文法に対し、文字列 00101 の最左導出、最右導出を示せ。

2. 算術式に対する文法

$$
\begin{aligned}
E &\to E + T|T \\
T &\to T * F|F \\
F &\to (E)|\mathbf{id}
\end{aligned}
$$

   を、左再帰でないように変形せよ。