

第3章 下向き構文解析 (教科書 p.50)

3.1 下向き構文解析の特徴

演算子順位法は if ~ else などの制御構造の部分には使いにくい

LR 法は 人手での作成に向かない

アイデア

左端のトークン (if, whileなど) を見て生成規則を選ぶ ()

例 教科書 p.14 図 1.4 の Statement (改)

```
Statement → if ( ConditionExp ) Statement ElsePart
           | { StatementSeq }
           | Id = Expression ;
           | Id ( ExpressionList );
           | ...
```

各非終端記号について次の疑似コードで示すような (再帰的な) 関数を定義する

```
int Statement(void) {
    switch (次のトークン) {
    case IF: {
        IF を消費;
        '(' " ";
        c = ConditionExp();
        ')' " ";
        s = Statement();
        e = ElsePart();
        return c, s, e を使った式
    }
    case '{': {
        '{' を消費;
        s = StatementSeq();
        '}' を消費;
        return s を使った式;
    }
    /* : */
    }
}
```

3.2 再帰下降構文解析 (recursive decent parsing)

- の一種 ... 幹から葉へ解析木ができていく
- があるとまずい

$$\begin{aligned} \text{expr} &\rightarrow \text{const} \\ &| \underline{\text{expr}} * \text{const} \end{aligned}$$

$$\begin{aligned} \text{StatementSeq} &\rightarrow \varepsilon \\ &| \underline{\text{StatementSeq}} \text{ Statement} \end{aligned}$$

下線部のところが、一番左端の再帰的出現である。これをプログラムにしようとすると、

```
int StatementSeq(void) {
    switch (次のトークン) {
        case IF: case WHILE: ... {
            ss = StatementSeq(); /* ← */
            sl = Statement();
            return ...;
        }

        /* : */

    }
}
```

← のところで入力が変わらないので、止まらなくなる。

必要な準備

- 先頭の共通部分をくくりだす

$$S \rightarrow AB \mid AC$$

は

$$\begin{aligned} S &\rightarrow AT \\ T &\rightarrow B \mid C \end{aligned}$$

に書き換える

- _____ する
- BNF の各右辺 α に対して _____ を求める
($First(\alpha)$ は α の _____)
- $A \xRightarrow{*} \varepsilon$ となりうる非終端記号 A に対して _____ を求める
($Follow(A)$ は A の _____)

再帰下降構文解析のプログラムの作り方

各非終端記号に対して関数を定義する

- $N \rightarrow X_{11}X_{12}\dots X_{1n_1} \mid \dots \mid X_{m1}X_{m2}\dots X_{mn_m}$ に対して、次のトークンがどの $First(X_{i1}X_{i2}\dots X_{ini})$ に属するかによって分岐する ($X_{i1}X_{i2}\dots X_{ini} \xRightarrow{*} \varepsilon$ となる場合は $Follow(N)$ も考慮する)

- II. 右辺の $X_{i1}X_{i2}\dots X_{ini}$ に対して $X_{i1}(); X_{i2}(); \dots X_{ini}();$ のように
 続けて関数を呼出す (ただし、 X_{ij} が終端記号のときは単にトークンを消費する)
 そのあと _____ を繰り返して書き換える (効率のため)

3.3 アルゴリズム (左再帰の除去) (教 p.57)

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n$$

とする (α_i, β_j は構文記号列で β_j の先頭の記号は A ではない。 β_j の先頭以外には A は出現しても構わない)

↓

先頭が $\beta_1 \sim \beta_n$ でそのあとに $\alpha_1 \sim \alpha_m$ が 0 回以上繰り返すというかたちになる

↓

次のように書き換えることができる

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \varepsilon \end{aligned}$$

注:

- 最後の ε を忘れない
- 間接的な左再帰 (教科書 p.54) があるともう少しややこしくなるが、そのような場合でも除去可能であることが知られている

→

問 3.3.1

$$L \rightarrow L ; C \mid C$$

の左再帰を除去せよ。 (L, C は非終端記号、 $;$ は終端記号である。)

First と Follow の求め方の例 (詳しい説明は教科書 pp.60-61)

例 5.3

$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow + T E' \mid \varepsilon \\ T &\rightarrow F T' \\ T' &\rightarrow * F T' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

$$\begin{aligned}
First(TE') &= First(T) = First(FT') = First(F) \\
&\leftarrow First(F) \text{に } \varepsilon \text{ が入っていないので } T' \text{ や } E' \text{ は考慮しなくてよ} \\
First(FT') &= First(F) = \{ (, \mathbf{id} \} \\
Follow(E') &= Follow(E) \cup Follow(E') \\
Follow(E) &= \{), \$ \} \leftarrow \text{開始記号の } Follow \text{ には } \$ \text{ (入力の終) を追加する} \\
Follow(T') &= Follow(T) \cup Follow(T') \\
Follow(T) &= First(E') \setminus \{ \varepsilon \} \cup Follow(E) \cup Follow(E') \\
&= \{ +,), \$ \} \\
First(E') &= \{ +, \varepsilon \} \leftarrow \varepsilon \text{ になりうる場合、} \varepsilon \text{ を加える} \\
(First(T') \text{ と } Follow(F) \text{ は求める必要はない})
\end{aligned}$$

3.4 予測型構文解析表 (教科書 表5.1 (p.63))

- $First$ と $Follow$ の結果をまとめて表にまとめたもの
- 構文解析すべき非終端記号 A と入力の先頭の終端記号 a に対して _____ を示す

LL(1) 文法

予測型構文解析表のエントリーに重複がない文法のことを _____ という

- エントリーに重複があると構文解析中に _____ が必要となる (通常のプログラミング言語では記述しにくい ... Prolog の出番?)
- LL(1) は Left-to-Right Leftmost derivation (1) に由来する。Leftmost (最左導出) はあとで説明する

表 5.1 教科書 p.63

	id	*	+	()	\$
$E \rightarrow$	i			i		
$E' \rightarrow$			ii		iii	iii
$T \rightarrow$	iv			iv		
$T' \rightarrow$		v	vi		vi	vi
$F \rightarrow$	vii			viii		

- i $\therefore First(TE') = \{ (, \mathbf{id} \}$
- ii $\therefore First(+TE') = \{ + \}$
- iii $\therefore Follow(E') = \{), \$ \}$
- iv $\therefore First(FT') = First(F) = \{ (, \mathbf{id} \}$
- v $\therefore First(*FT') = \{ * \}$
- vi $\therefore Follow(T') = \{ +,), \$ \}$
- vii $\therefore First(\mathbf{id}) = \{ \mathbf{id} \}$
- viii $\therefore First((E)) = \{ (\}$

入力例 $x + y * z$ ただし、 x, y, z は **id** に属するトークンである。

予測	入力	動作

—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—
—	—		—

結果

導出列 _____

木

-
- 最左導出 ... _____
 - 下向き構文解析 ... 先に根に近い節ができる
-

