コンパイラ

湯淺太一

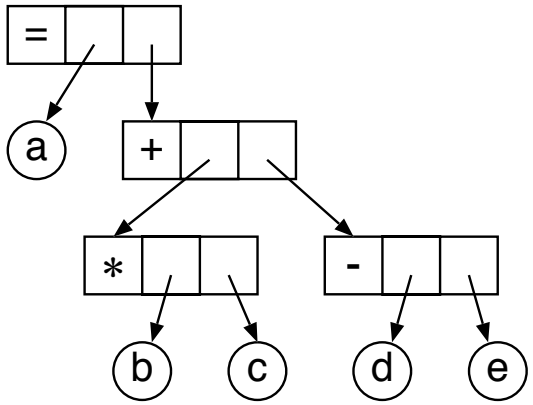
第４章

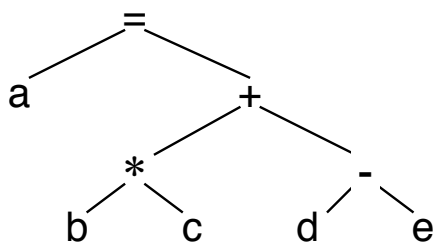
構　文　解　析

構文木とその表現

N組 (N-tuple)

```
a = b * c + (d - e)
```





`tuple`：N組へのポインタの型

`token`：トークンへのポインタの型

`tuple`型は，`token`型を含む.

N組の生成

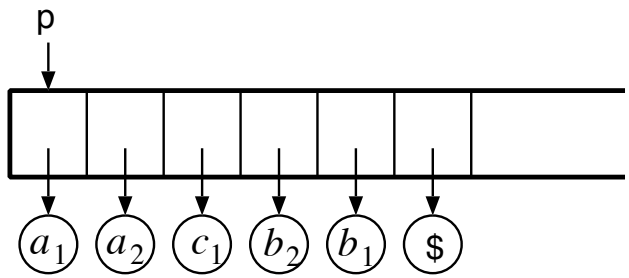
```
x1 = make_tuple("*", b, c);
x2 = make_tuple("-", d, e);
x3 = make_tuple("+", x1, x2);
x4 = make_tuple("=", a, x3);
```

トークンの種類判定

```
int is_a(x)
```

再帰的下向き (recursive descent) 構文解析法

解析関数 (parsing function)



$A \to aAb \mid c$

```
tuple parse_A() {
  token *prev = p, x, z; tuple y;
  if (is_a(x = *p++) && (y = parse_A())
                     && is_b(z = *p++))
    return make_tuple("A", x, y, z);
  p = prev;
  if (is_c(x = *p++)) return x;
  p = prev;
  return NULL;
}
```

$は特殊な終端記号と見なす.

生成規則は再帰的
→ 再帰的な解析関数群

入力トークン列全体の解析

```
tuple parse() {
  tuple x = parse_A();
  if (x && is_eof(*p))
    return x;
  else
    syntax_error();
}
```

再帰的下向き構文解析法の問題点

- 左再帰 (left recursion)
- バックトラック (backtracking)

左再帰

文法が左再帰を含む → 解析関数の再帰呼出しが止らない.

G1の生成規則：

$$E \to E + T \mid T$$

直接の左再帰 (immediate left recursion)

```
tuple parse_E() {
  token *prev = p; tuple x, y;
  if ((x = parse_E()) && is_plus(*p++)
                      && (y = parse_T()))
    return make_tuple("+", x, y);
  p = prev;
  if (x = parse_T()) return x;
  p = prev;
  return NULL;
}
```

間接的な左再帰

$$A \to Ba \mid c$$
$$B \to Ab \mid d$$

C 言語における左再帰

identifier-list:
　identifier
　identifier-list , *identifier*

右再帰に置き換え可能.

identifier-list:
　identifier
　identifier , *identifier-list*

単純に右再帰に置き換えられない例

additive-expression:
　multiplicative-expression
　additive-expression + *multiplicative-expression*
　additive-expression - *multiplicative-expression*

左結合的が右結合的になってしまう.

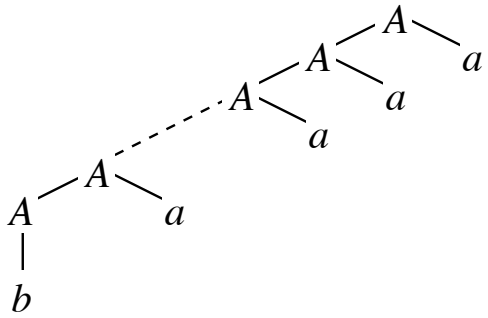
$$\texttt{a - b + c} = \texttt{(a - b) + c} \neq \texttt{a - (b + c)} = \texttt{a - b - c}$$

C 言語の文法には，間接的な左再帰は存在しない.

「直接の左再帰」の除去

$A \to Aa \mid b$

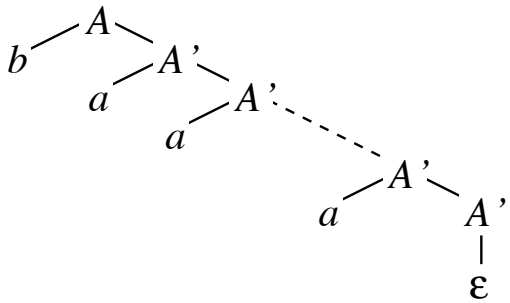
$$A \underset{\text{lm}}{\Longrightarrow} Aa \underset{\text{lm}}{\Longrightarrow} Aaa \underset{\text{lm}}{\Longrightarrow} Aaaa \underset{\text{lm}}{\Longrightarrow} \cdots \underset{\text{lm}}{\Longrightarrow} Aa^n \underset{\text{lm}}{\Longrightarrow} ba^n$$



$A \to bA'$

$A' \to aA' \mid \varepsilon$

$$A \underset{\text{rm}}{\Longrightarrow} bA' \underset{\text{rm}}{\Longrightarrow} baA' \underset{\text{rm}}{\Longrightarrow} baaA' \underset{\text{rm}}{\Longrightarrow} \cdots \underset{\text{rm}}{\Longrightarrow} ba^nA' \underset{\text{rm}}{\Longrightarrow} ba^n$$




```
tuple parse_A() {
  token *prev = p, y;
  if (is_b(y = *p++)) return parse_A1(y);
  p = prev;
  return NULL;
}
tuple parse_A1(tuple x) {
  token *prev = p, y;
  if (is_a(y = *p++))
    return parse_A1(make_tuple("A", x, y));
  p = prev;
  return x;
}
```

一般の場合

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \cdots \mid \beta_m$$

ここで

$\beta_j \in V^*$，β_j は A で始まらない．

$\alpha_i \in V^+$

$$A \to \beta_1 A' \mid \cdots \mid \beta_m A'$$
$$A' \to \alpha_1 A' \mid \cdots \mid \alpha_n A' \mid \varepsilon$$

例：G1の生成規則

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (\ E\ ) \mid i$$

から左再帰を除去した文法G3

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to *FT' \mid \varepsilon$$
$$F \to (\ E\ ) \mid i$$

バックトラック

複数の選択肢があるとき，その一つを試してみて，
失敗だったらもとの状態に戻してやり直す．

例

$A \to aBc$

$B \to b \mid bd$

```
tuple parse_A() {
  token *prev = p, x, z; tuple y;
  if (is_a(x = *p++) && (y = parse_B())
                     && is_c(z = *p++))
    return make_tuple("A", x, y, z);
  p = prev;
  return NULL;
}
tuple parse_B() {
  token *prev = p, x, y;
  if (is_b(x = *p++)) return x;
  p = prev;
  if (is_b(x = *p++) && is_d(y = *p++))
    return make_tuple("B", x, y);
  p = prev;
  return NULL;
}
```

正しく動作しない（例：$abdc$）$\to$ 一般的には複雑な機構が必要

くくり出し (factoring)によるバックトラックの回避

数学では「$ax + ay$のaをくくり出して$a(x + y)$」

$B \to b \mid bd$のbをくくり出す.

$B \to b\,(d \mid \varepsilon)$

```
tuple parse_B() {
  token *prev = p, x;
  if (is_b(x = *p++)) {
    token *prev1 = p, y;
    if (is_d(y = *p++))
      return make_tuple("B", x, y);
    p = prev1;
    return x;
  }
  p = prev;
  return NULL;
}
```

生成規則が終端記号で始まらない場合は適用が困難.

$A \to aB\,c \quad B \to C \mid D \quad C \to \cdots \quad D \to \cdots$

バックトラックのもう一つの大きな問題：

適切なエラーメッセージ出力が難しい

LL(k)構文解析法

k個のトークンを先読みし，どの生成規則を適用するかを決定する

$B \to C \mid D$の解析関数で，
次の入力トークンxを先読みし，
1. もし$C \overset{+}{\Rightarrow} x \cdots$なら`parse_C`を呼び出す.
2. もし$D \overset{+}{\Rightarrow} x \cdots$なら`parse_D`を呼び出す.
3. どちらでもなければ構文エラー

LL(1)文法

文法 $G = \langle V_N, V_T, P, S \rangle$

First(α)：記号列 α の first 集合

$$\mathrm{First}(\alpha) = \{a \mid a \in V_T,\ \alpha \overset{*}{\Rightarrow} a \cdots\}$$

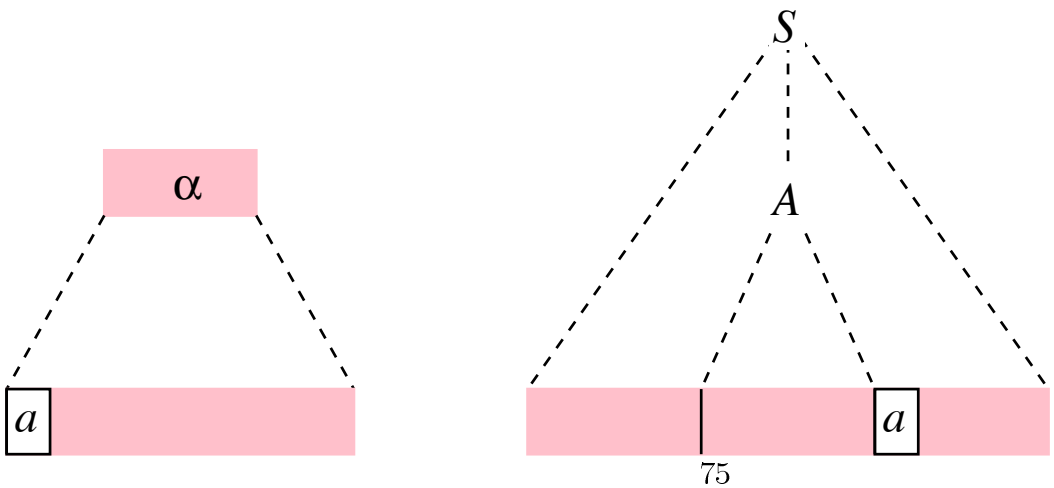
Follow(A)：非終端記号 A の follow 集合

$$\mathrm{Follow}(A) = \{a \mid a \in V_T,\ S\,\$ \overset{*}{\Rightarrow} \cdots Aa \cdots\}$$

$\$ \in \mathrm{Follow}(A)$ なら，

$$S \overset{*}{\Rightarrow} \cdots A$$

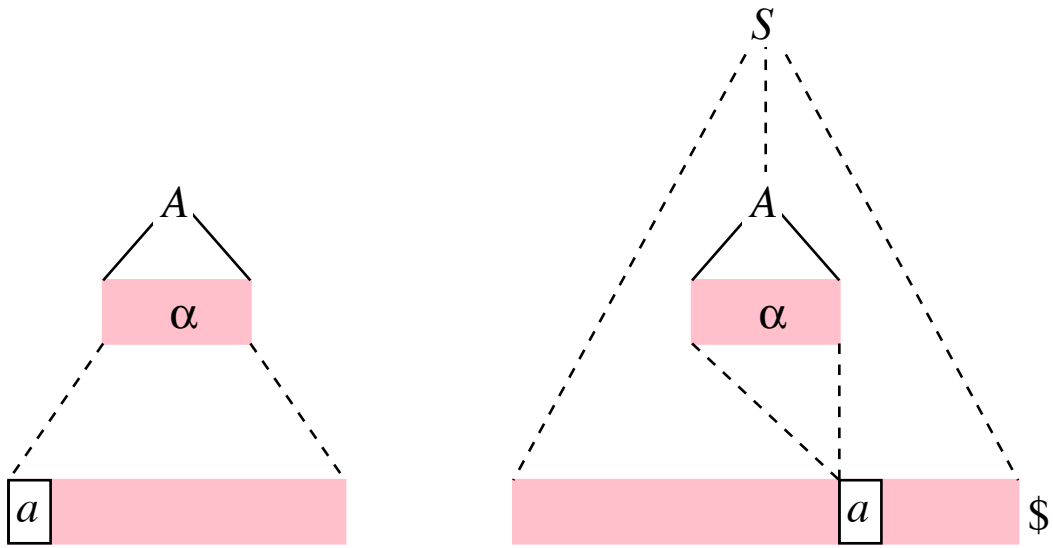
常に，$\$ \in \mathrm{Follow}(S)$



Director(A, α)：生成規則 $A \to \alpha$ の director 集合

$$\mathrm{Director}(A, \alpha) = \begin{cases} \mathrm{First}(\alpha) & (\alpha \overset{*}{\Rightarrow} \varepsilon \text{でない}) \\ \mathrm{First}(\alpha) \cup \mathrm{Follow}(A) & (\alpha \overset{*}{\Rightarrow} \varepsilon \text{のとき}) \end{cases}$$

$A \to \alpha$ で還元するときに，入力トークン列の先頭に現れる可能性のある終端記号の全体



LL(1) 文法：右辺だけが異なる任意の生成規則

$A \to \alpha$ と

$A \to \beta$ に対して

必ず

$\mathrm{Director}(A, \alpha) \cap \mathrm{Director}(A, \beta) = \phi$

が成り立つ文法

LL(1)文法のための計算

記号列αがεを生成する（$\alpha \overset{*}{\Rightarrow} \varepsilon$）かどうかの判定

$\alpha = \varepsilon$なら$\alpha \overset{*}{\Rightarrow} \varepsilon$
$\alpha = a_1 a_2 \cdots a_n \neq \varepsilon$なら$\alpha \overset{*}{\Rightarrow} \varepsilon$は，

　すべてのa_iに対して$a_i \overset{*}{\Rightarrow} \varepsilon$

と同値

$\mathrm{Null}(a)$：記号aに対して$a \overset{*}{\Rightarrow} \varepsilon$かどうか

$$\mathrm{Null}(a) = \begin{cases} \mathrm{true} & (a \overset{*}{\Rightarrow} \varepsilon \text{のとき}) \\ \mathrm{false} & (a \overset{*}{\Rightarrow} \varepsilon \text{でないとき}) \end{cases}$$

1. すべての記号aに対して$\mathrm{Null}(a) \leftarrow \mathrm{false}$
2. ε規則$A \rightarrow \varepsilon$に対して，$\mathrm{Null}(A) \leftarrow \mathrm{true}$
3. ε規則以外の生成規則$A \rightarrow a_1 \cdots a_n$に対して，
   すべてのa_iについて$\mathrm{Null}(a_i) = \mathrm{true}$ならば，
   $\mathrm{Null}(A) \leftarrow \mathrm{true}$
4. 変更がなくなるまで，ステップ3を繰り返す．

例：G1から左再帰を除去した文法G3

$$
\begin{aligned}
&E \to TE' \\
&E' \to \texttt{+}TE' \mid \varepsilon \\
&T \to FT' \\
&T' \to \texttt{*}FT' \mid \varepsilon \\
&F \to \texttt{(}E\texttt{)} \mid i
\end{aligned}
$$

$\mathrm{Null}(E') = \mathrm{Null}(T') = \mathrm{true}$

$\mathrm{Null}(E) = \mathrm{Null}(T) = \mathrm{Null}(F) = \mathrm{false}$

first 集合の計算

$\alpha = \varepsilon$ なら $\mathrm{First}(\alpha) = \phi$
$\alpha = a_1 a_2 \cdots a_n \neq \varepsilon$ に対しては，

$$\mathrm{First}(\alpha) = \begin{cases} \mathrm{First}(a_1) & (a_1 \overset{*}{\Rightarrow} \varepsilon \text{でない}) \\ \mathrm{First}(a_1) \cup \mathrm{First}(a_2 \cdots a_n) & (a_1 \overset{*}{\Rightarrow} \varepsilon \text{のとき}) \end{cases}$$

$\rightarrow$ 記号についての first 集合から求められる．

1. 終端記号 x に対して $\mathrm{First}(x) \leftarrow \{x\}$
   非終端記号 A に対して，$\mathrm{First}(A) \leftarrow \phi$
2. 生成規則 $A \rightarrow \alpha_1 a \alpha_2$ $(a \in (V_N \cup V_T))$ に対して，
   $\alpha_1 \overset{*}{\Rightarrow} \varepsilon$ なら $\mathrm{First}(A) \leftarrow \mathrm{First}(A) \cup \mathrm{First}(a)$
3. 変更がなくなるまで，ステップ 2 を繰り返す．

例：文法G3のfirst集合

$$
\begin{aligned}
&E \to TE' \\
&E' \to \texttt{+}TE' \mid \varepsilon \\
&T \to FT' \\
&T' \to \texttt{*}FT' \mid \varepsilon \\
&F \to \texttt{(}E\texttt{)} \mid i
\end{aligned}
$$

生成規則$F \to \texttt{(}E\texttt{)}$と$F \to i$から

$$\mathrm{First}(F) = \{\texttt{(}, i\}$$

生成規則$T' \to \texttt{*}FT'$から

$$\mathrm{First}(T') = \{\texttt{*}\}$$

$T \to FT'$から（$\mathrm{Null}(F) = \mathrm{false}$なので）

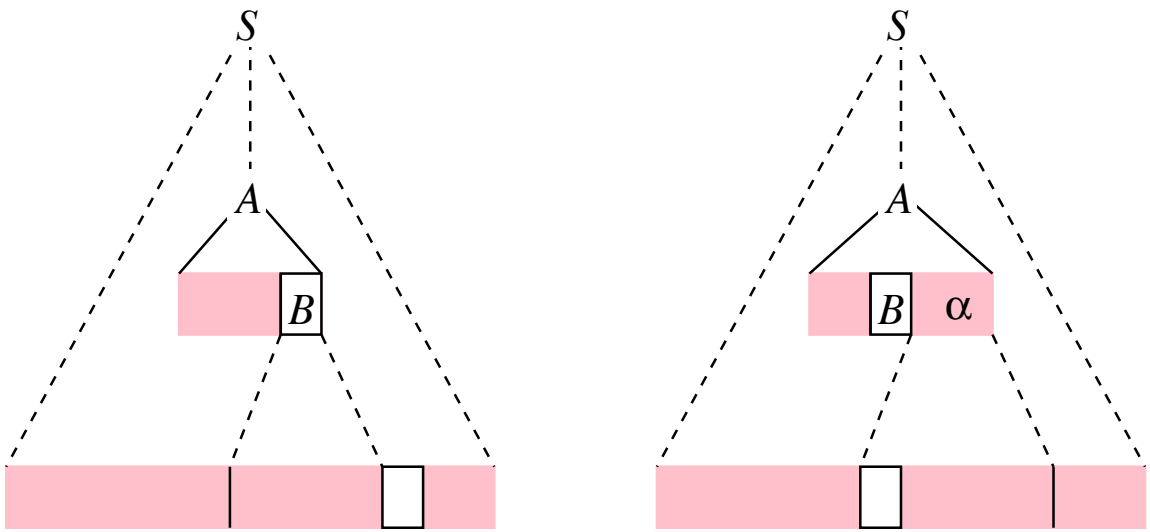
$$\mathrm{First}(T) = \mathrm{First}(F) = \{\texttt{(}, i\}$$

$\mathrm{First}(T')$，$\mathrm{First}(T)$と同様に，

$$
\begin{aligned}
&\mathrm{First}(E') = \{\texttt{+}\} \\
&\mathrm{First}(E) = \mathrm{First}(T) = \{\texttt{(}, i\}
\end{aligned}
$$

follow集合の計算

1. 出発記号Sに対して$\mathrm{Follow}(S) \leftarrow \{\$\}$
   S以外の非終端記号Aに対して$\mathrm{Follow}(A) \leftarrow \phi$
2. 生成規則$A \rightarrow \cdots B$ $(B \in V_N)$に対して，
   $\mathrm{Follow}(B) \leftarrow \mathrm{Follow}(B) \cup \mathrm{Follow}(A)$
3. 生成規則$A \rightarrow \cdots B\alpha$ $(B \in V_N, \alpha \neq \varepsilon)$に対して，
   $\mathrm{Follow}(B) \leftarrow \mathrm{Follow}(B) \cup \mathrm{First}(\alpha)$
   もし$\alpha \overset{*}{\Rightarrow} \varepsilon$なら，さらに
   $\mathrm{Follow}(B) \leftarrow \mathrm{Follow}(B) \cup \mathrm{Follow}(A)$
4. 変更がなくなるまでステップ2と3を繰り返す．



例：文法G3のfollow集合

$$
\begin{aligned}
&E \to TE' \\
&E' \to \texttt{+}TE' \mid \varepsilon \\
&T \to FT' \\
&T' \to \texttt{*}FT' \mid \varepsilon \\
&F \to \texttt{(}E\texttt{)} \mid i
\end{aligned}
$$

出発記号Eが右辺に現れる規則は$F \to \texttt{(}E\texttt{)}$だけ

$$\mathrm{Follow}(E) = \{\,\$, \texttt{)}\}$$

E'が右辺に現れる生成規則は$E \to TE'$と$E' \to \texttt{+}TE'$

$$\mathrm{Follow}(E') = \mathrm{Follow}(E) = \{\,\$, \texttt{)}\}$$

Tが右辺に現れる生成規則も$E \to TE'$と$E' \to \texttt{+}TE'$
Tの直後はE'，$\mathrm{Null}(E') = \mathrm{true}$なので，

$$\mathrm{Follow}(T) = \mathrm{First}(E') \cup \mathrm{Follow}(E') = \{\texttt{+}, \$, \texttt{)}\}$$

$\mathrm{Follow}(E')$，$\mathrm{Follow}(T)$と同様に，

$$
\begin{aligned}
&\mathrm{Follow}(T') = \mathrm{Follow}(T) = \{\texttt{+}, \$, \texttt{)}\} \\
&\mathrm{Follow}(F) = \mathrm{First}(T') \cup \mathrm{Follow}(T') = \{\texttt{*}, \texttt{+}, \$, \texttt{)}\}
\end{aligned}
$$

文法G3におけるdirector集合

$$
\begin{aligned}
E &\to TE' \\
E' &\to \texttt{+}TE' \mid \varepsilon \\
T &\to FT' \\
T' &\to \texttt{*}FT' \mid \varepsilon \\
F &\to \texttt{(}\,E\,\texttt{)} \mid i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Null}(E') = \mathrm{Null}(T') = \mathrm{true} \\
&\mathrm{Null}(E) = \mathrm{Null}(T) = \mathrm{Null}(F) = \mathrm{false} \\
&\mathrm{First}(E) = \mathrm{First}(T) = \mathrm{First}(F) = \{\texttt{(}, i\} \\
&\mathrm{First}(T') = \{\texttt{*}\} \\
&\mathrm{First}(E') = \{\texttt{+}\} \\
&\mathrm{Follow}(E) = \mathrm{Follow}(E') = \{\,\$, \texttt{)}\} \\
&\mathrm{Follow}(T) = \mathrm{Follow}(T') = \{\texttt{+}, \$, \texttt{)}\} \\
&\mathrm{Follow}(F) = \mathrm{First}(T') \cup \mathrm{Follow}(T') = \{\texttt{*}, \texttt{+}, \$, \texttt{)}\} \\
&\mathrm{Director}(E',\ \texttt{+}TE') = \{\texttt{+}\} \\
&\mathrm{Director}(E',\ \varepsilon) = \mathrm{First}(\varepsilon) \cup \mathrm{Follow}(E') = \{\,\$, \texttt{)}\} \\
&\mathrm{Director}(T',\ \texttt{*}FT') = \{\texttt{*}\} \\
&\mathrm{Director}(T',\ \varepsilon) = \mathrm{First}(\varepsilon) \cup \mathrm{Follow}(T') = \{\texttt{+}, \$, \texttt{)}\} \\
&\mathrm{Director}(F,\ \texttt{(}\,E\,\texttt{)}) = \{\texttt{(}\} \\
&\mathrm{Director}(F,\ i) = \{i\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Director}(E',\ \texttt{+}TE') \cap \mathrm{Director}(E',\ \varepsilon) = \phi \\
&\mathrm{Director}(T',\ \texttt{*}FT') \cap \mathrm{Director}(T',\ \varepsilon) = \phi \\
&\mathrm{Director}(F,\ \texttt{(}\,E\,\texttt{)}) \cap \mathrm{Director}(F,\ i) = \phi
\end{aligned}
$$

G3はLL(1)文法である．

LL(1)構文解析法 (Left-to-right scanning, Left-most derivation)

バックトラックのない再帰的下向き構文解析

G3のための解析関数

```
tuple parse_E() {
  if (is_lpar(*p)
       || is_i(*p)) {
    tuple x = parse_T();
    return parse_E1(x);
  } else
    syntax_error();
}
```

```
tuple parse_E1(tuple x) {
  if (is_plus(*p)) {
    tuple y;
    p++;
    y = parse_T();
    return parse_E1(make_tuple("+", x, y));
  } else if (is_eof(*p) || is_rpar(*p))
    return x;
  else
    syntax_error();
}
```

バックトラックがなくなったので

1. 適切な時点で構文エラーを検出できる
2. pの値を保存しておく必要がない.

LR構文解析 (Left-to-right scanning, Right-most derivation in reverse)

文法 $\langle\{E \to E+i \mid i\}, E\rangle$ における $i+i+i$ の最右導出

$$E \underset{\mathrm{rm}}{\Longrightarrow} E+i \underset{\mathrm{rm}}{\Longrightarrow} E+i+i \underset{\mathrm{rm}}{\Longrightarrow} i+i+i$$

注目点挿入

$$\begin{aligned} E\cdot &\underset{\mathrm{rm}}{\Longrightarrow} E+i\cdot \rightharpoonup E+\cdot i \rightharpoonup E\cdot+i \\ &\underset{\mathrm{rm}}{\Longrightarrow} E+i\cdot+i \rightharpoonup E+\cdot i+i \rightharpoonup E\cdot+i+i \\ &\underset{\mathrm{rm}}{\Longrightarrow} i\cdot+i+i \rightharpoonup \cdot i+i+i \end{aligned}$$

$i+i+i\$$ の LR構文解析

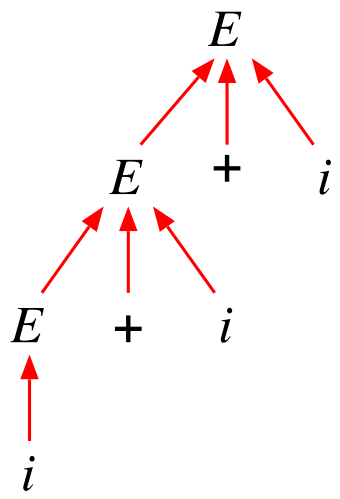
$$\cdot i+i+i\$ \to i\cdot+i+i\$ \to E\cdot+i+i\$ \to E+\cdot i+i\$ \to E+i\cdot+i\$ \to E\cdot+i\$$$
$$\to E+\cdot i\$ \to E+i\cdot\$ \to E\cdot\$$$

各ステップでは,

還元（reduce）：注目点直前の記号列を還元

シフト（shift）：注目点を一つ右に移動

$\cdot i + i + i\,\$ \to i \cdot + i + i\,\$ \to E \cdot + i + i\,\$ \to E + \cdot i + i\,\$ \to E + i \cdot + i\,\$ \to E \cdot + i\,\$$
$\to E + \cdot i\,\$ \to E + i \cdot \$ \to E \cdot \$$



$A \to \alpha$で還元するときに
　Aの節を用意し，αの各記号の節へ枝を張る．

上向き構文解析法(bottom-up parsing)

初期状態：$\cdot x\,\$$
　x：入力トークン列

解析中の状態：$u \cdot v\,\$$
　$u \in (V_T \cup V_N)^*$：構築した部分解析木の記号列
　$v \in V_T^*$：未処理の入力トークン列

最終状態：$S \cdot \$$
　S：出発記号

最終状態に変換できれなければ，構文エラー．

LR(0)項

生成規則の右辺にドットをつけたもの

例：$E \to E+i$ のLR(0)項：

$E \to \cdot E+i$　（導入項）
　$E+i$ を読み込めば E に還元できる．

$E \to E\cdot+i$
　E を読み込んだ．$+i$ を読み込めば E に還元できる．

$E \to E+\cdot i$
　$E+$ を読み込んだ．i を読み込めば E に還元できる．

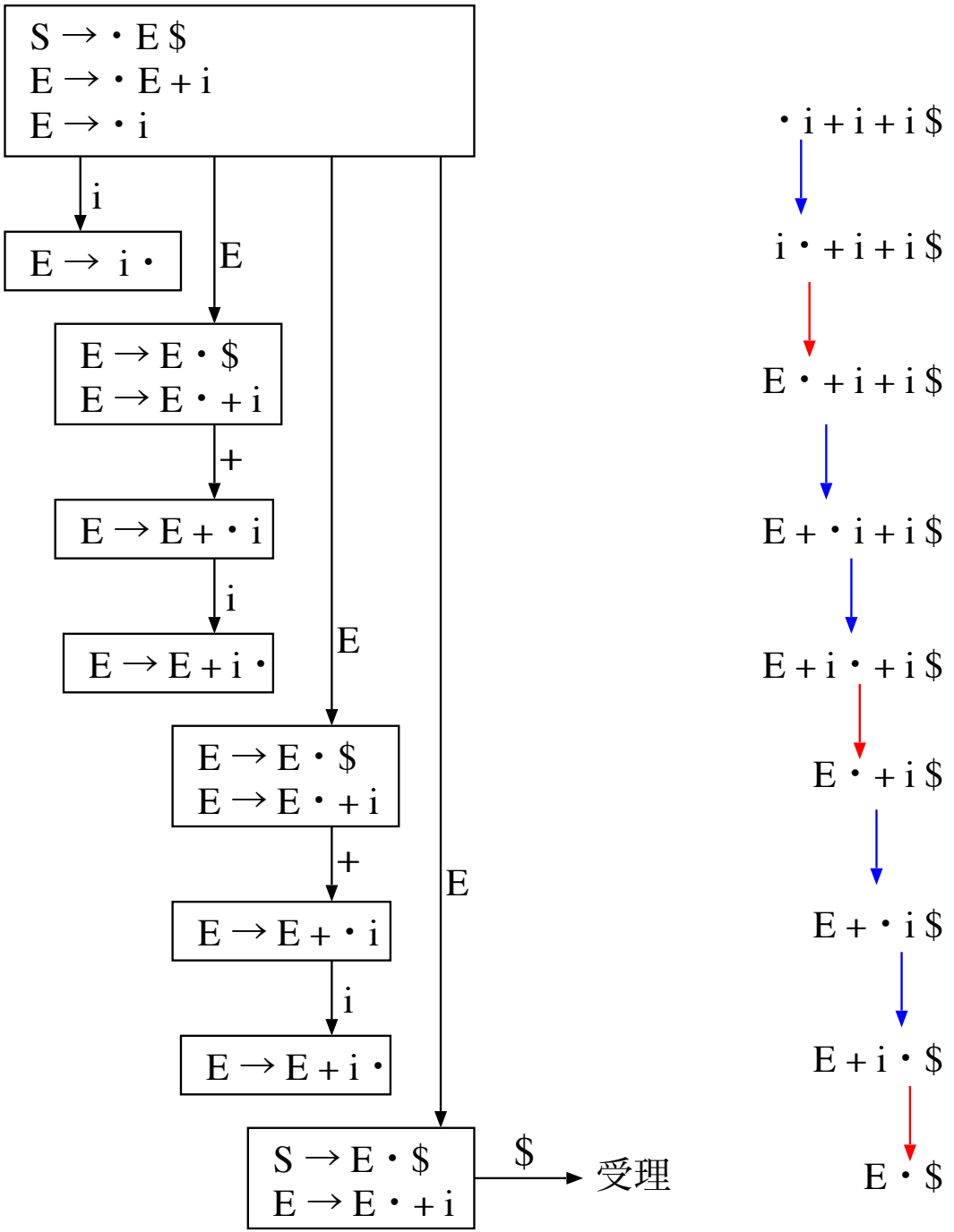
$E \to E+i\cdot$　（完全項）
　$E+i$ を読み込んだ．E に還元できる．

E: 文法上の出発記号
S: 新しい出発記号

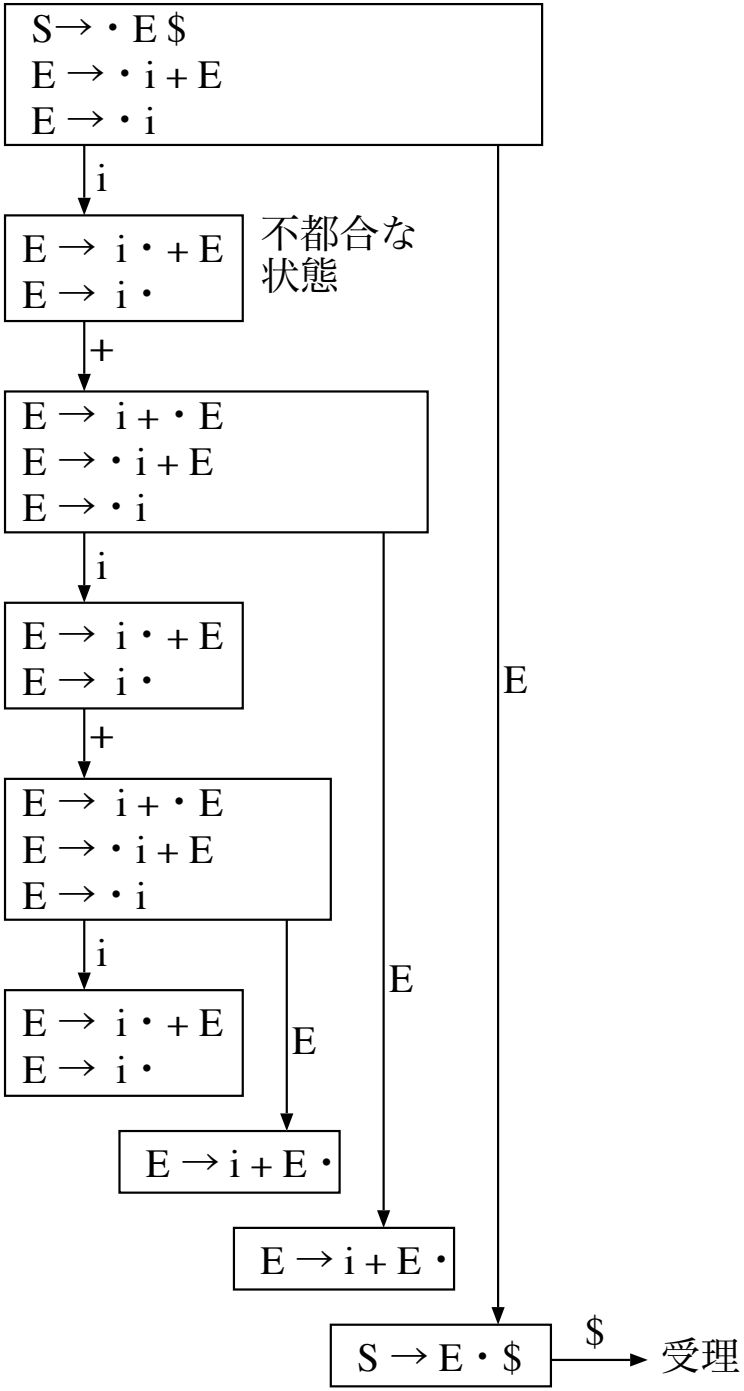
$S \to \cdot E\$$　（出発項）
　E を読み込んで $\$$ に達すれば受理できる．

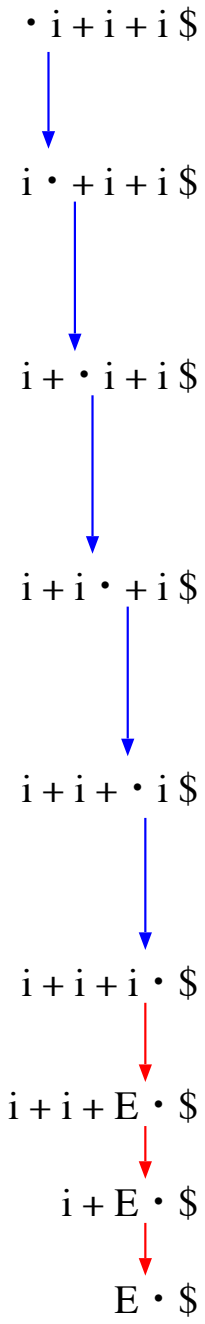
$S \to E\cdot\$$　（受理項）
　E を読み込んだ．$\$$ に達していれば受理できる．

文法 $\langle\{E \to E+i \mid i\}, E\rangle$ におけるLR構文解析



文法 $\langle\{E \to i + E \mid i\}, E\rangle$ におけるLR構文解析





I: LR(0)項集合
Closure(I)：Iの閉包(closure)

- Closure(I) $\supseteq I$
- $A \to \alpha_1 \cdot B\,\alpha_2$がClosure($I$)の要素なら，
  任意の導入項$B \to \cdot\,\beta$も，Closure(I)の要素

例：文法 $\langle \{E \to i + E \mid i\}, E \rangle$

$$\begin{aligned}&\mathrm{Closure}(S \to \cdot\,E\,\$)\\&= \{S \to \cdot\,E\,\$,\ \ E \to \cdot\,i + E,\ \ E \to \cdot\,i\}\end{aligned}$$

$$\begin{aligned}&\mathrm{Goto}(I, a)\\&= \mathrm{Closure}(\{A \to \alpha_1\,a \cdot \alpha_2 \mid (A \to \alpha_1 \cdot a\,\alpha_2) \in I\})\end{aligned}$$

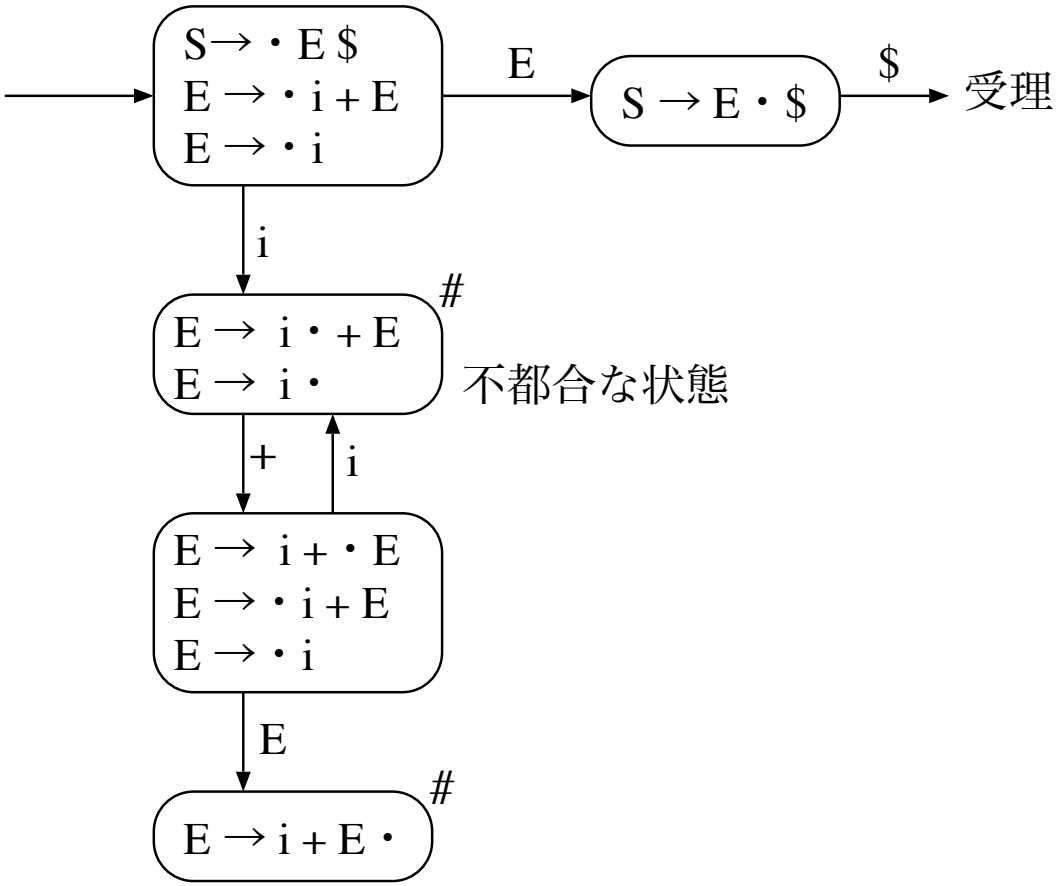
例：文法 $\langle \{E \to i + E \mid i\}, E \rangle$

$$\begin{aligned}&\mathrm{Goto}(\{E \to i \cdot + E,\ \ E \to i \cdot\},\ +)\\&= \mathrm{Closure}(\{E \to i + \cdot\,E\})\\&= \{E \to i + \cdot\,E,\ \ E \to \cdot\,i + E,\ \ E \to \cdot\,i\}\end{aligned}$$

正準オートマトン (canonical automaton)

初期状態 Closure($\{S \to \cdot E\,\$\}$) から始め，可能な遷移をすべて求める

例：文法 $\langle\{E \to i + E \mid i\}, E\rangle$ の正準オートマトン



#: 還元状態（完全項を含む状態）

正準集合：正準オートマトンの状態集合

配置(configuration)

LR構文解析のある時点

$$a_1 a_2 \cdots a_n \bullet x_k x_{k+1} \cdots x_m \$$$

における配置は,

$$(q_0\, a_1\, q_1\, a_2\, q_2 \cdots a_n\, q_n,\quad x_k x_{k+1} \cdots x_m \$)$$

q_0：初期状態，$q_i = \mathrm{Goto}(q_{i-1}, a_i) = \mathrm{Goto}(q_0, a_1 a_2 \cdots a_n)$

LR構文解析開始時点の配置：

$$(q_0,\quad x_1 x_2 \cdots x_m \$)$$

$A \to a_i \cdots a_n$による還元後：

$$(q_0\, a_1\, q_1 \cdots a_{i-1}\, q_{i-1}\, A\, q,\quad x_k x_{k+1} \cdots x_m \$)$$

$q = \mathrm{Goto}(q_{i-1}, A)$

x_kを読み込んでシフト後：

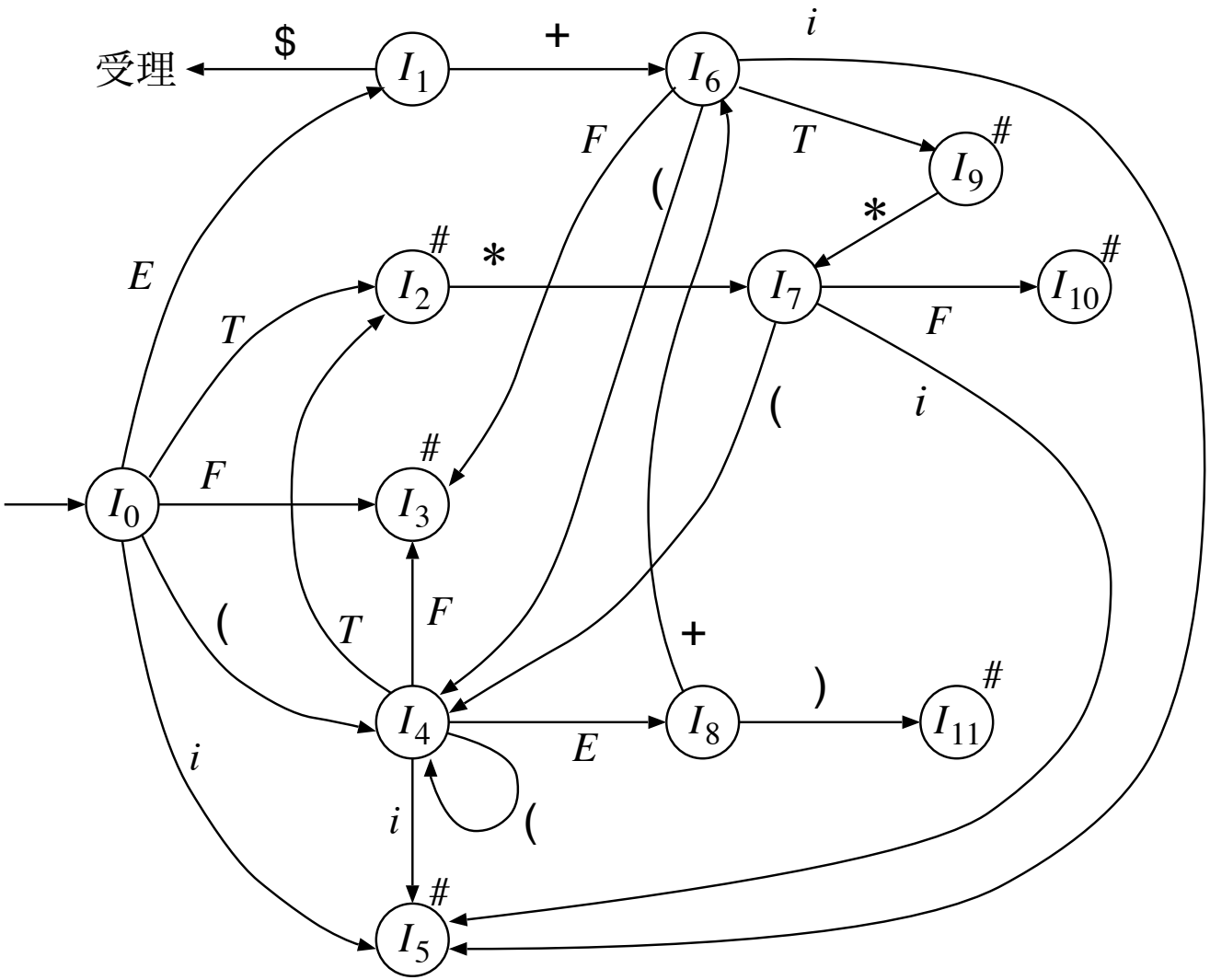
$$(q_0\, a_1\, q_1 \cdots a_n\, q_n\, x_k\, q,\quad x_{k+1} \cdots x_m \$)$$

$q = \mathrm{Goto}(q_n, x_k)$

最終的な配置：

$$(q_0\, S\, q,\quad \$)$$

例：文法 $\mathrm{G1} = \langle \mathrm{P1}, E \rangle$ の正準オートマトン



$$\mathrm{P1} = \{\ E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (\ E\ ) \mid i\ \}$$

初期状態 I_0
$= \mathrm{Closure}(\{S \to \cdot E\,\$\})$

#: 還元状態

不都合な状態（とりあえずシフトを優先）：

$I_2 = \{\ E \to T\cdot,\ \ T \to T\cdot * F\ \}$

$I_9 = \{\ E \to E + T\cdot,\ \ T \to T\cdot * F\ \}$

G1の正準集合

$$
\begin{aligned}
I_0 = \{\ & S \to \cdot E\,\$, \\
& E \to \cdot E + T, \\
& E \to \cdot T, \\
& T \to \cdot T * F, \\
& T \to \cdot F, \\
& F \to \cdot (\ E\ ), \\
& F \to \cdot i\ \}
\end{aligned}
$$

$$
\begin{aligned}
I_1 = \{\ & S \to E \cdot \$, \\
& E \to E \cdot + T\ \}
\end{aligned}
$$

$$
\begin{aligned}
I_2 = \{\ & E \to T \cdot, \\
& T \to T \cdot * F\ \}
\end{aligned}
$$

$$
I_3 = \{\ T \to F \cdot\ \}
$$

$$
\begin{aligned}
I_4 = \{\ & F \to (\ \cdot E\ ), \\
& E \to \cdot E + T, \\
& E \to \cdot T \\
& T \to \cdot T * F, \\
& T \to \cdot F, \\
& F \to \cdot (\ E\ ), \\
& F \to \cdot i\ \}
\end{aligned}
$$

$$
I_5 = \{\ F \to i \cdot\ \}
$$

$$
\begin{aligned}
I_6 = \{\ & E \to E + \cdot T, \\
& T \to \cdot T * F, \\
& T \to \cdot F, \\
& F \to \cdot (\ E\ ), \\
& F \to \cdot i\ \}
\end{aligned}
$$

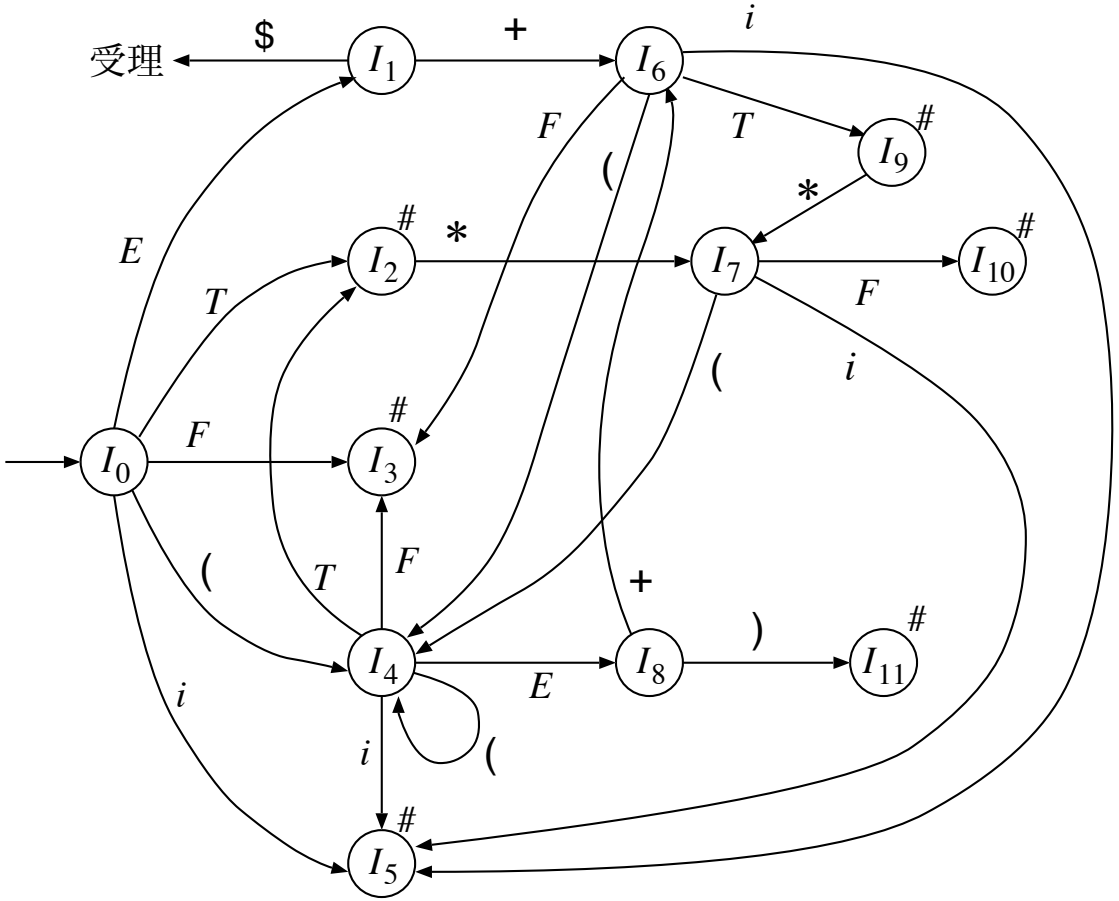
$$
\begin{aligned}
I_7 = \{\ & T \to T * \cdot F, \\
& F \to \cdot (\ E\ ), \\
& F \to \cdot i\ \}
\end{aligned}
$$

$$
\begin{aligned}
I_8 = \{\ & F \to (\ E \cdot\ ), \\
& E \to E \cdot + T\ \}
\end{aligned}
$$

$$
\begin{aligned}
I_9 = \{\ & E \to E + T \cdot, \\
& T \to T \cdot * F\ \}
\end{aligned}
$$

$$
I_{10} = \{\ T \to T * F \cdot\ \}
$$

$$
I_{11} = \{\ F \to (\ E\ ) \cdot\ \}
$$



$(I_0,$	$i+i*i\$)$	iの読込み	$(I_0\,E\,I_1+I_6\,F\,I_3,$	$*i\$)$	$T \to F$で還元
$(I_0\,i\,I_5,$	$+i*i\$)$	$F \to i$で還元	$(I_0\,E\,I_1+I_6\,T\,I_9,$	$*i\$)$	‘$*$’の読込み
$(I_0\,F\,I_3,$	$+i*i\$)$	$T \to F$で還元	$(I_0\,E\,I_1+I_6\,T\,I_9*I_7,$	$i\$)$	iの読込み
$(I_0\,T\,I_2,$	$+i*i\$)$	$E \to T$で還元	$(I_0\,E\,I_1+I_6\,T\,I_9*I_7\,i\,I_5,$	$\$)$	$F \to i$で還元
$(I_0\,E\,I_1,$	$+i*i\$)$	‘+’の読込み	$(I_0\,E\,I_1+I_6\,T\,I_9*I_7\,F\,I_{10},$	$\$)$	$T \to T*F$で還元
$(I_0\,E\,I_1+I_6,$	$i*i\$)$	iの読込み	$(I_0\,E\,I_1+I_6\,T\,I_9,$	$\$)$	$E \to E+T$で還元
$(I_0\,E\,I_1+I_6\,i\,I_5,$	$*i\$)$	$F \to i$で還元	$(I_0\,E\,I_1,$	$\$)$	受理

SLR(1)構文解析

Simple LR,
1文字先読み(lookahead)し，
follow集合で不都合な状態を解消

SLR(1)文法：SLR(1)構文解析が可能な文法

例：G1の不都合な状態I_2

$$I_2 = \{\ E \to T\cdot,\ \ T \to T\cdot * F\ \}$$

次の入力トークンが

1. Follow$(E) = \{+,), \$\}$の要素なら，$E \to T$で還元
2. ‘$*$’なら，‘$*$’を読み込んでシフト
3. どちらでもなければ，構文エラー

$I_9 = \{\ E \to E + T\cdot,\ \ T \to T\cdot * F\ \}$
Follow$(E) = \{+,), \$\} \not\ni *$

$\to$ G1はSLR(1)文法

	i	+	*	(	)	$
I_0	s5			s4		
I_1		s6				受理
I_2		r2	s7		r2	r2
I_3		r4	r4		r4	r4
I_4	s5			s4		
I_5		r6	r6		r6	r6
I_6	s5			s4		
I_7	s5			s4		
I_8		s6			s11	
I_9		r1	s7		r1	r1
I_{10}		r3	r3		r3	r3
I_{11}		r5	r5		r5	r5

動作表

	E	T	F
I_0	I_1	I_2	I_3
I_4	I_8	I_2	I_3
I_6		I_9	I_3
I_7			I_{10}

行先表

1: $E \to E + T$
2: $E \to T$
3: $T \to T * F$
4: $T \to F$
5: $F \to (\ E\ )$
6: $F \to i$

動作表(action table)
- sj：状態 I_j にシフト
- rj：j 番目の生成規則で還元
- 受理：構文解析終了
- 空欄：動作未定義 → 構文エラー

行先表(goto table)
- 還元後の状態（Goto(I, a)）

SLR(1)構文解析プログラム（文法に依存せず）

```
tuple parse() {
  int state = 0;
  for (;;) {
    action act = lookup_action(state, *p);
    if (act.todo == SHIFT) {
      push_state(state);
      push_value(*p++);
      state = act.num;
    } else if (act.todo == REDUCE) {
      int nt = reduce(act.num);
      state = lookup_goto(top_state(), nt);
    } else if (act.todo == ACCEPT)
      return pop_value();
    else  /* if (act.todo == UNDEFINED) */
      syntax_error();
  }}
```

配置$(q_0\, a_1\, q_1 \cdots q_{n-1}\, a_n\, q_n,\ \ x_k x_{k+1} \cdots x_m\$)$ の表現：

変数state：q_n

stateスタック：$q_0, q_1, \cdots, q_{n-1}$　valueスタック：$a_1, \cdots, a_n$

還元用関数（文法に依存）

```
int reduce(int production) {
  tuple x, y;
  switch (production) {
  case 1: /* E -> E + T */
    x = pop_value(); pop_value();
    y = pop_value();
    pop_state(); pop_state();
    push_value(make_tuple("+", y, x));
  case 2: /* E -> T */
    return E;
  case 3: /* T -> T * F */ ...
  case 4: /* T -> F */     ...
  case 5: /* F -> ( E ) */ ...
  case 6: /* F -> i */     ...
  }}
```

$E \to E + T$ による還元：

$$(q_0\, a_1\, q_1 \cdots a_{n-3}\, q_{n-3}\, E_1\, q_{n-2} + q_{n-1}\, T_1\, q_n,\ \cdots) \to (q_0\, a_1\, q_1 \cdots a_{n-3}\, q_{n-3}\, E_2\, q\,,\ \cdots)$$

E_1 の構文木は T_1 と同じ

LR(1) や LALR(1) でも，プログラムは同じ．動作表と行先表が異なる．

LR(1)構文解析

follow集合より正確な先読み記号を使用

$A \to \alpha$による還元

$$S \cdot \$ \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} uA \cdot v\,\$ \underset{\mathrm{rm}}{\Longrightarrow} u\alpha \cdot v\,\$$$

次の入力xは，$x \in \mathrm{First}(v\,\$)$のはず.

LR(1)項：次の入力がxなら$A \to \alpha$で還元可能

$$[A \to \alpha \cdot,\ \ x]$$

LR(1)項の一般形：

$$[A \to \alpha_1 \cdot \alpha_2,\ \ x]$$

完全項を核(core)とするLR(1)項

$$[A \to \alpha_1\alpha_2 \cdot,\ \ x]$$

になって次の入力がxなら，$A \to \alpha_1\alpha_2$で還元可能

LR(1)構文解析の初期状態：

$$I_0 = \mathrm{Closure}(\{\ [S' \to \cdot S,\ \ \$]\ \})$$

$$\mathrm{Goto}(I, a) = \mathrm{Closure}(\{[A \to \alpha_1 a \cdot \alpha_2,\ c] \mid [A \to \alpha_1 \cdot a\,\alpha_2,\ c] \in I\})$$

LR(1)項集合Iの閉包$I' = \mathrm{Closure}(I)$

1. $I' \leftarrow I$
2. I'の要素$[A \to \alpha_1 \cdot B\,\alpha_2,\ x]$,
   生成規則$B \to \beta$,
   $x' \in \mathrm{First}(\alpha_2\, x)$であるすべての$x'$について,
   $[B \to \cdot\,\beta,\ x']$を$I'$に追加.
3. I'に追加がなくなるまで，ステップ2を繰り返す.

例：G1をLR(1)構文解析するときの初期状態

$$
\begin{aligned}
I_0 &= \mathrm{Closure}(\{\ [S \to \cdot\,E,\ \ \$]\ \}) \\
&= \{\ [S \to \cdot\,E,\ \ \$], \\
&\qquad [E \to \cdot\,E\mathtt{+}T,\ \ \$],\ \ [E \to \cdot\,T,\ \ \$],\ \ [E \to \cdot\,E\mathtt{+}T,\ \ \mathtt{+}],\ \ [E \to \cdot\,T,\ \ \mathtt{+}], \\
&\qquad [T \to \cdot\,T\mathtt{*}F,\ \ \$],\ \ [T \to \cdot\,F,\ \ \$],\ \ [T \to \cdot\,T\mathtt{*}F,\ \ \mathtt{+}],\ \ [T \to \cdot\,F,\ \ \mathtt{+}], \\
&\qquad [T \to \cdot\,T\mathtt{*}F,\ \ \mathtt{*}],\ \ [T \to \cdot\,F,\ \ \mathtt{*}],\ \ [F \to \cdot\,\mathtt{(}\,E\,\mathtt{)},\ \ \$],\ \ [F \to \cdot\,i,\ \ \$], \\
&\qquad [F \to \cdot\,\mathtt{(}\,E\,\mathtt{)},\ \ \mathtt{+}],\ \ [F \to \cdot\,i,\ \ \mathtt{+}],\ \ [F \to \cdot\,\mathtt{(}\,E\,\mathtt{)},\ \ \mathtt{*}],\ \ [F \to \cdot\,i,\ \ \mathtt{*}]\ \} \\
&= \{\ [S \to \cdot\,E,\ \ \$], \\
&\qquad [E \to \cdot\,E\mathtt{+}T,\ \ \$/\mathtt{+}],\ \ [E \to \cdot\,T,\ \ \$/\mathtt{+}], \\
&\qquad [T \to \cdot\,T\mathtt{*}F,\ \ \$/\mathtt{+}/\mathtt{*}],\ \ [T \to \cdot\,F,\ \ \$/\mathtt{+}/\mathtt{*}], \\
&\qquad [F \to \cdot\,\mathtt{(}\,E\,\mathtt{)},\ \ \$/\mathtt{+}/\mathtt{*}],\ \ [F \to \cdot\,i,\ \ \$/\mathtt{+}/\mathtt{*}]\ \}
\end{aligned}
$$

LR(1)構文解析のための正準オートマトンの作り方：

SLR(1)の場合とまったく同じ．

例：G1をLR(1)構文解析するためのオートマトン

$$
\begin{aligned}
I_2 &= \mathrm{Goto}(I_0,\ T) = \mathrm{Closure}(\{\ [E \to T\cdot,\ \ \$/\mathtt{+}],\ \ [T \to T\cdot\mathtt{*}F,\ \ \$/\mathtt{+}/\mathtt{*}]\ \}) \\
&= \{\ [E \to T\cdot,\ \ \$/\mathtt{+}],\ [T \to T\cdot\mathtt{*}F,\ \ \$/\mathtt{+}/\mathtt{*}]\ \} \\
I_4 &= \mathrm{Goto}(I_0,\ \mathtt{(}) = \mathrm{Closure}(\{\ [F \to \mathtt{(}\cdot E\mathtt{)},\ \ \$/\mathtt{+}/\mathtt{*}]\ \}) \\
&= \{\ [F \to \mathtt{(}\cdot E\mathtt{)},\ \ \$/\mathtt{+}/\mathtt{*}], \\
&\qquad [E \to \cdot E\mathtt{+}T,\ \ \mathtt{)}/\mathtt{+}],\ \ [E \to \cdot T,\ \ \mathtt{)}/\mathtt{+}], \\
&\qquad [T \to \cdot T\mathtt{*}F,\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}],\ \ [T \to \cdot F,\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}], \\
&\qquad [F \to \cdot\mathtt{(}E\mathtt{)},\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}],\ \ [F \to \cdot i,\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}]\ \} \\
I_2' &= \mathrm{Goto}(I_4,\ T) = \mathrm{Closure}(\{\ [E \to T\cdot,\ \ \mathtt{)}/\mathtt{+}],\ \ [T \to T\cdot\mathtt{*}F,\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}]\ \}) \\
&= \{\ [E \to T\cdot,\ \ \mathtt{)}/\mathtt{+}],\ \ [T \to T\cdot\mathtt{*}F,\ \ \mathtt{)}/\mathtt{+}/\mathtt{*}]\ \} \\
&\neq I_2
\end{aligned}
$$

SLR(1)のときは$\mathrm{Goto}(I_4, T) = I_2$だった．
$\to$ LR(1)は，より詳細な構文解析が可能．

SLR(1)文法はすべてLR(1)文法．
LR(1)文法で，SLR(1)文法でないものが存在する．

LALR(1)構文解析

LR(1)構文解析法は，状態数が多過ぎて実用的でない．

核が同じであるLR(1)項を同一視して状態数を減らす．

例：

$$I_2 = \{\ [E \to T\bullet,\ \ \$/+],\ \ [T \to T\bullet * F,\ \$/+/*]\ \}$$
$$I_2' = \{\ [E \to T\bullet,\ \ )/+],\ \ [T \to T\bullet * F,\ )/+/*]\ \}$$

を融合し，

$$I_2 = \{\ [E \to T\bullet,\ \ \$/+/)\ ],\ \ [T \to T\bullet * F,\ \$/+/*/)\ ]\ \}$$

さらに

$$\mathrm{Goto}(I_4,\ T) = I_2$$

LALR(1)は，正確さはLR(1)より劣るが，
　SLR(1)よりは正確．
状態数はSLR(1)と同じ．

LALR(1)文法で，SLR(1)文法でない例：

$$\mathrm{P4} = \{\ S \to E + E \mid i$$
$$E \to i\ \}$$

$I_0 = \{\ [S' \to \cdot S,\ \$],\ [S \to \cdot E + E,\ \$],\ [S \to \cdot i,\ \$],\ [E \to \cdot i,\ +\ ]\ \}$

$I_1 = \{\ [S' \to S \cdot,\ \$]\ \}$

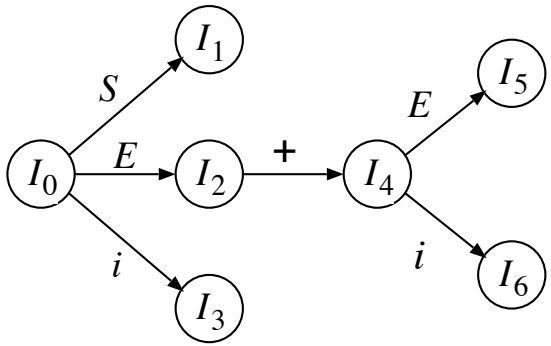
$I_2 = \{\ [S \to E \cdot + E,\ \$]\ \}$

$I_3 = \{\ [S \to i \cdot,\ \$],\ [E \to i \cdot,\ +\ ]\ \}$

$I_4 = \{\ [S \to E \cdot E,\ \$],\ [E \to \cdot i,\ \$]\ \}$

$I_5 = \{\ [S \to E + E \cdot,\ \$]\ \}$

$I_6 = \{\ [E \to i \cdot,\ \$]\ \}$



この文法はLALR(1)文法（LR(1)文法でもある）

SLR(1)用の $I_3 = \{S \to i \cdot,\ \ E \to i \cdot\}$ について

$\mathrm{Follow}(S) = \{\$\}$，$\mathrm{Follow}(E) = \{\$, +\}$

なので，SLR(1)文法でない．

ポイント：

I_3は，I_0からiを読み込んだ直後の状態．

$E \to i$で還元すれば，$S \to E + E$の最初のE．

だから，次の入力は‘+’

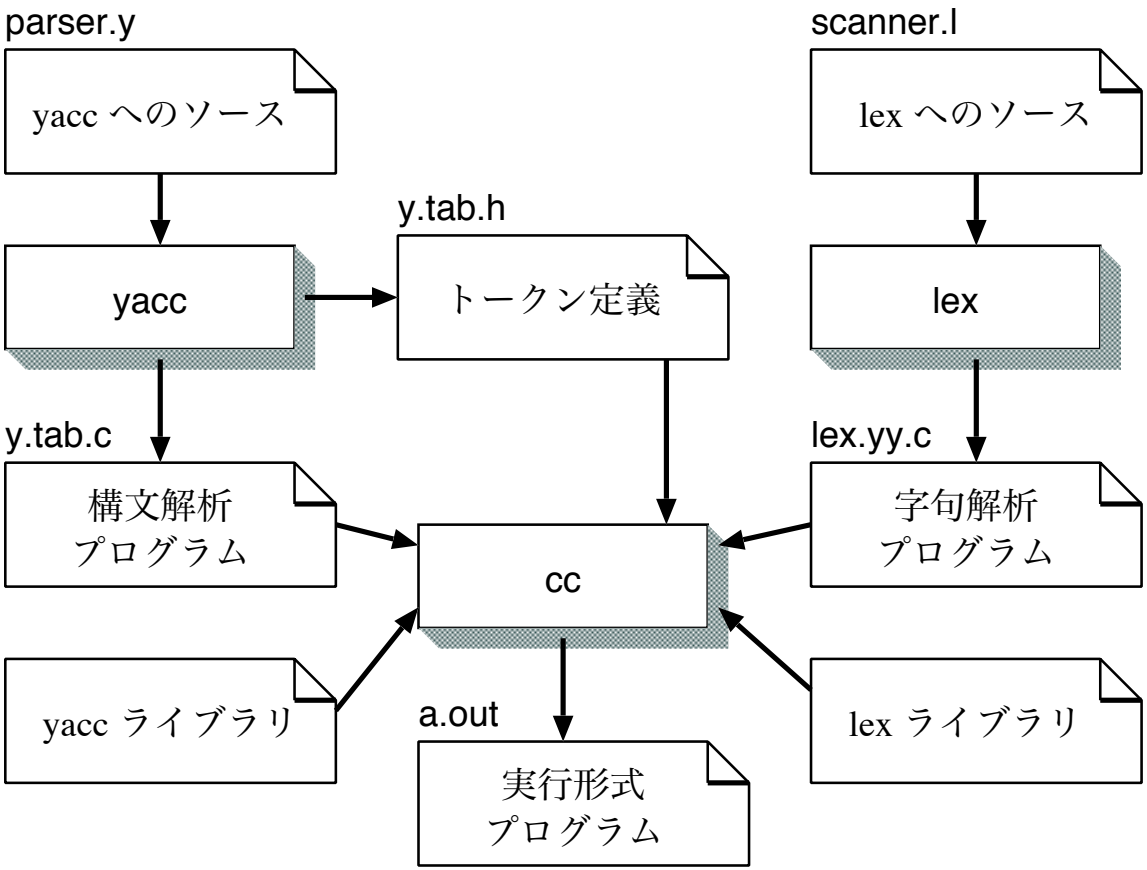
二つ目のEのために，$\mathrm{Follow}(E)$が$\$$を含む．

構文解析の自動化(yacc – Yet Another Compiler Compiler)

```
% cat parser.y
%token PLUS TIMES LPAR RPAR IDENTIFIER
%%
E:  E PLUS T {$$=make_tuple("+",$1,$3);}
  | T        {$$=$1;}
T:  T TIMES F {$$=make_tuple("*",$1,$3);}
  | F         {$$=$1;}
F:  LPAR E RPAR {$$=$2;}
  | IDENTIFIER  {$$=$1;}
%%
  ··· ここにmake_tuple等の定義 ···
% yacc -d parser.y
% ls -l
...    565 Aug  2 23:36 parser.y
...  11838 Aug  2 23:36 y.tab.c
...     92 Aug  2 23:36 y.tab.h
% cat y.tab.h
#define PLUS 257
#define TIMES 258
...
%
```

yacc用にlexを使う例：

```
% cat scanner.l
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[\t ]+ {}
"+"    {return PLUS;}
"*"    {return TIMES;}
"("    {return LPAR;}
")"    {return RPAR;}
[a-z][a-z0-9]* {yylval = copy_lexeme(); return IDENTIFIER;}
\n     {return 0;}
.      {printf("lexical error: '%c'\n", yytext[0]); return 0;}
%%
char StringTable[1000]; char *STp = StringTable;
int copy_lexeme() {
  int p = (int) STp; strcpy(STp, yytext);
  STp += strlen(yytext)+1; return p;
}
%
```



make_tupleに構文木を表示させる：

```
int prev_tuple = 0;
int make_tuple(char *op, int arg1, int arg2) {
  printf("T%d: (%s, ", ++prev_tuple, op);
  if (arg1 > 0) printf("%s, ", arg1); else printf("T%d, ", -arg1);
  if (arg2 > 0) printf("%s", arg2); else printf("T%d", -arg2);
  printf(")\n"); return -prev_tuple;
}
```

実行例：

```
% lex scanner.l
% cc y.tab.c lex.yy.c -ly -ll
% a.out
x*y+z*w
T1: (*, x, y)
T2: (*, z, w)
T3: (+, T1, T2)
% a.out
x+y+
T1: (+, x, y)
syntax error
%
```

yaccを使って，動作表と行先表の情報を調べる：

```
% yacc -v parser.y
% cat y.output
state 0
    $accept : . E $end  (0)
    IDENTIFIER  shift 5
    LPAR  shift 4
    .  error
    E  goto 1
    T  goto 2
    F  goto 3
state 1
  ... 中略 ...
state 11
    F : LPAR E RPAR .  (5)
    .  reduce 5
7 terminals, 4 nonterminals
7 grammar rules, 12 states
%
```

あいまいな文法への対処

文法 $\mathrm{Gif} = \langle \mathrm{Pif}, S \rangle$ を使用する.

$$\begin{aligned}\mathrm{Pif} = \{\ & S \to \texttt{if}\ (\ E\ )\ S \mid \texttt{if}\ (\ E\ )\ S\ \texttt{else}\ S \mid i\ \texttt{=}\ E\ \texttt{;} \\ & E \to i\ \}\end{aligned}$$

LL(1)構文解析の場合

くくり出しの結果Gif' $= \langle$Pif', $S\rangle$

$$
\begin{aligned}
\text{Pif'} = \{\ & S \to \texttt{if}\ (\ E\ )\ S\,A \mid i\ \texttt{=}\ E\ \texttt{;} \\
& A \to \texttt{else}\ S \mid \varepsilon \\
& E \to i\ \}
\end{aligned}
$$

$$
\begin{aligned}
&\text{First}(E) = \{i\} \\
&\text{First}(A) = \{\texttt{else}\} \cup \text{Follow}(A) \\
&\quad = \{\texttt{else}\} \cup \text{Follow}(S) = \{\texttt{else}, \$\} \\
&\text{First}(S) = \{\texttt{if}, i\} \\
&\text{Follow}(S) = \{\$\} \cup \text{First}(A) = \{\$,\ \texttt{else}\} \\
&\text{Follow}(A) = \text{Follow}(S) = \{\$,\ \texttt{else}\} \\
&\text{Follow}(E) = \{\ \texttt{)},\ \texttt{;}\}
\end{aligned}
$$

$$
\begin{aligned}
&\text{Director}(S,\ \texttt{if}\ (\ E\ )\ S\,A) = \{\texttt{if}\} \\
&\text{Director}(S,\ i\ \texttt{=}\ E\ \texttt{;}) = \{i\} \\
&\text{Director}(A,\ \texttt{else}\ S) = \{\texttt{else}\} \\
&\text{Director}(A,\ \varepsilon) = \text{Follow}(A) = \{\$,\ \texttt{else}\} \\
&\text{Director}(E,\ i) = \{i\}
\end{aligned}
$$

$\text{Director}(A,\ \texttt{else}\ S) \cap \text{Director}(A,\ \varepsilon) = \{\texttt{else}\} \neq \phi$

なので，Gif'は，LL(1)文法でない．

次が else なら，$A \to$ else S を採用することにする．

```
tuple parse_A() {
  if (is_else(*p)) {
    p++;
    return parse_S();
  } else if (is_eof(*p))
    return EMPTY;
  else
    syntax_error();
}
```

```
tuple parse_S() {
  tuple x, y, z;
  if (is_if(*p)) {
    p++;
    if (!is_lpar(*p++)) syntax_error();
    x = parse_E();
    if (!is_rpar(*p++)) syntax_error();
    y = parse_S(); z = parse_A();
    return make_tuple("if", x, y, z);
  } else if (is_i(*p)) {
    x = *p++;
    if (!is_eq(*p++)) syntax_error();
    y = parse_E();
    if (!is_semicolon(*p++)) syntax_error();
    return make_tuple("=", x, y);
  } else
    syntax_error();
}
```

LR構文解析の場合

Gifの正準集合（抜粋）：

$I_0 = \mathrm{Closure}(\{S' \to \cdot S\,\$\})$
$\quad = \{S' \to \cdot S\,\$,\ \ S \to \cdot\,\texttt{if}\ (\ E\ )\ S,\ \ S \to \cdot\,\texttt{if}\ (\ E\ )\ S\ \texttt{else}\ S,\ \ S \to \cdot\, i\ \texttt{=}\ E\ \texttt{;}\}$
$I_1 = \mathrm{Goto}(I_0,\ \texttt{if}\,) = \{S \to \texttt{if}\cdot(\ E\ )\ S,\ \ S \to \texttt{if}\cdot(\ E\ )\ S\ \texttt{else}\ S\}$
$I_2 = \mathrm{Goto}(I_1,\ (\,) = \mathrm{Closure}(\{S \to \texttt{if}\ (\cdot E\ )\ S,\ \ S \to \texttt{if}\ (\cdot E\ )\ S\ \texttt{else}\ S\})$
$\quad = \{S \to \texttt{if}\ (\cdot E\ )\ S,\ \ S \to \texttt{if}\ (\cdot E\ )\ S\ \texttt{else}\ S,\ \ E \to \cdot i\}$
$I_3 = \mathrm{Goto}(I_2,\ E) = \{S \to \texttt{if}\ (\ E\cdot\ )\ S,\ \ S \to \texttt{if}\ (\ E\cdot\ )\ S\ \texttt{else}\ S\}$
$I_4 = \mathrm{Goto}(I_3,\ )\,) = \mathrm{Closure}(\{S \to \texttt{if}\ (\ E\ )\cdot S,\ \ S \to \texttt{if}\ (\ E\ )\cdot S\ \texttt{else}\ S\})$
$\quad = \{S \to \texttt{if}\ (\ E\ )\cdot S,\ \ S \to \texttt{if}\ (\ E\ )\cdot S\ \texttt{else}\ S,$
$\qquad S \to \cdot\,\texttt{if}\ (\ E\ )\ S,\ \ S \to \cdot\,\texttt{if}\ (\ E\ )\ S\ \texttt{else}\ S,\ \ S \to \cdot\, i\ \texttt{=}\ E\ \texttt{;}\}$
$I_5 = \mathrm{Goto}(I_4,\ S) = \{S \to \texttt{if}\ (\ E\ )\ S\cdot,\ \ S \to \texttt{if}\ (\ E\ )\ S\cdot\texttt{else}\ S\}$

不都合な状態I_5について

$$\mathrm{Follow}(S) = \{\$,\ \texttt{else}\}$$

なので，GifはSLR(1)文法ではない．

シフトすると決めてしまえばSLR(1)構文解析は可能．
$\mathrm{Action}(I_5, \texttt{else}) = \mathrm{s6}$として動作表を作る．

GifはLR(1)文法でもLALR(1)文法でもない．
同様に対処すれば，LR(1)・LALR(1)構文解析も可能．

yaccの場合

yaccのルール

1. 還元よりもシフトを優先する.
2. 先に与えた生成規則による還元を優先する.

```
% cat Gif.y
  ... 中略 ...
S:  IF LPAR E RPAR S {$$=make_tuple("if",$3,$5,EMPTY);}
  | IF LPAR E RPAR S ELSE S {$$=make_tuple("if",$3,$5,$7);}
  | IDENTIFIER EQ E SEMICOLON {$$=make_tuple("=",$1,$3);}
E:  IDENTIFIER {$$=$1;}
  ... 中略 ...
% yacc -d Gif.y
conflicts: 1 shift/reduce
% cc y.tab.c lex.yy.c -ly -ll
% a.out
if (x) if (y) a=b; else b=a;
T1: (=, a, b)
T2: (=, b, a)
T3: (if, y, T1, T2)
T4: (if, x, T3, -)
%
```

エラーリカバリ

1回のコンパイルで，できるだけ多くのエラーを検出

エラー検出で構文解析をただちに終了しないで，
なんらかの処置を行って実行を継続.

LL(1)構文解析のエラーリカバリ

`parse_`Aがエラーを検出するのは：

1. 次の入力が，どの$\mathrm{Director}(A, \alpha)$にも属さない.
2. $A \to \alpha$の解析中に，次の入力が，期待した終端記号と異なる.

例：Gif'の`parse_S`

Aから生成されたであろうトークン列を予測し，それらを読み飛ばす.

1の場合：

$A \overset{*}{\Rightarrow} \cdots a$である終端記号$a$まで
または，$\mathrm{Follow}(A)$の要素の直前まで読み飛ばす.

2の場合：

期待する終端記号の自動挿入（読み飛ばしなし）
または，期待する終端記号まで読み飛ばす.

LR構文解析のエラーリカバリ

動作表の値が「未定義」のときにエラー検出

非終端記号ごとのエラーリカバリは困難
→大きな単位（文，宣言，関数定義など）について

非終端記号 A についてのエラーリカバリ：

$$(q_0\, a_1\, q_1 \cdots a_n\, q_n,\ \ x_k x_{k+1} \cdots x_m \$)$$

の配置で，$\mathrm{Action}(q_n, x_k)$ が未定義

エラーがなく A への還元が成功していれば，

$$(q_0\, a_1\, q_1 \cdots q_i\, A\, q,\ \ x_j x_{j+1} \cdots x_m \$) \qquad 0 \le i < n,\ k \le j \le m,\ q = \mathrm{Goto}(q_i, A)$$

スタックの巻き戻し： $\mathrm{Goto}(q_i, A)$ が定義されている q_i が見つかるまで

入力トークンの読み飛ばし： $\mathrm{Follow}(A)$ の要素 x_j が見つかるまで

Gif の S についてのエラーリカバリ：

```
while ((state = lookup_goto(top_state(), S)) < 0) {
  pop_state(); pop_value();
}
pop_value(); push_value(ERROR);
while (!is_eof(*p) && !is_else(*p)) p++;
```

yaccのエラーリカバリ: エラー規則を使用

```
S:  IF LPAR E RPAR S { ... }
  | IF LPAR E RPAR S ELSE S { ... }
  | IDENTIFIER EQ E SEMICOLON { ... }
  | error EQ E SEMICOLON {
      printf("Error: bad lvar.\n"); $$=$3;}
  | IDENTIFIER error E SEMICOLON {
      printf("Error: '=' expected.\n");
      $$=make_tuple("=",$1,$3);}
```

「$S \rightarrow i$ = E ;」に対応するエラー規則

$S \rightarrow$ error = E ;

$S \rightarrow i$ error E ;

```
% a.out
if (x) =b; else b=a;
Error: bad lvar.
T1: (=, b, a)
T2: (if, x, b, T1)
% a.out
if (x) a b; else b=a;
Error: '=' expected.
T1: (=, a, b)
T2: (=, b, a)
T3: (if, x, T1, T2)
%
```

「$A \to \alpha_1$ `error` α_2」の`error`の位置で構文エラー：
「$A \to \alpha_1$・`error` α_2」を含む状態で検出

1. エラートークンを生成して，

   $A \to \alpha_1$ `error`・α_2

   にシフト．
2. α_2に還元できるトークン列を探し，Aへ還元．
   （トークンを読み飛ばしながら，通常の解析）
3. 通常の構文解析を再開．