* [] 与 ()的区别?

[A-Z]是匹配其中一个字符，()是表示整个表达式，就是要匹配全部字符。

* R1/R2（R1和R2是正则式）表示超前搜索：若要匹配R1，则必须先看紧跟其后的超前搜索部分是否与R2匹配。

* [0-9]{1,2}　至少指定一位但至多两位数字

[1-9][0-9]?　匹配 1~99 的正整数表达式

匹配时，冲突解决规则：

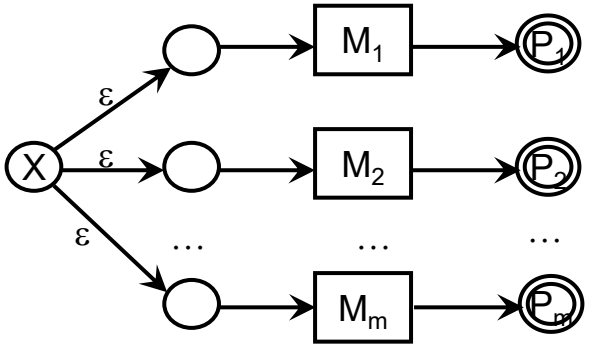
1） 总是选择最长的前缀

2） 如果最长的可能前缀与多个模式匹配，总是选择Lex中先被列出的模式。

例：fo{1,3}d 表示可以匹配？

* LEX的工作过程

* 对每条识别规则P_i构造一个相应的非确定有穷自动机M_i；
* 引进一个新初态X，通过ε弧，将这些自动机连接成一个新的NFA M；
* 把M确定化、最小化，生成该DFA的状态转换表和控制执行程序



由lex创建的词法分析器按如下方式工作：

* 词法分析器自左至右读入字符串，直至发现一个最长的与某个模式P_i匹配的前缀，则执行相关的动作A_i，通常A_i会将控制返回给语法分析器（除空白或注释外）。同时，词法分析器返回一个值，即种别码，但在需要时可以利用整型yylval传递这个词素的附加信息。
* 蓝书P87-88 例3.11的图3-20/ 电子书P89-91例3.11的图3-23

Lex源程序的写法

* 关于lex更进一步的知识及例子：
  https://blog.csdn.net/wp1603710463/article/details/50365495
* 关于正则式的书写：
  https://www.runoob.com/regexp/regexp-syntax.html

3.10 Lex词法分析器自动生成工具

方法1：

- 【编译原理】下载安装flex，实现词法分析器

  https://blog.csdn.net/chaifang0620/article/details/103124090

- 用gcc编译器编译lex编译器生成的c源程序

  gcc [-o outfile] lex.yy.c -lfl

  其中，-lfl是链接flex的库函数的。-o outfile是可选编译选项，默认生成a.exe.

前提：安装好gcc,
https://www.cnblogs.com/raina/p/10656106.html

3.10 Lex词法分析器自动生成工具

方法2：

* 1、首先先下载flex/bison for windows

  flex：http://gnuwin32.sourceforge.net/packages/flex.htm

  bison：http://gnuwin32.sourceforge.net/packages/bison.htm

  选择complete package setup版本。

* 2、安装后为bin目录配置环境变量。

* 3、在命令行中输入flex --version

  可得到版本号：flex version 2.5.4

3.10 Lex词法分析器自动生成工具

* 4、编译源程序：

flex filename.l

5、用gcc编译器编译lex编译器生成的c源程序

gcc [-o outfile] lex.yy.c –lfl

其中，-lfl是链接flex的库函数的。-o outfile是可选编译选项，默认生成a.exe.

前提：安装好gcc, https://www.cnblogs.com/raina/p/10656106.html

6、运行a.exe

./a.exe

* 例1：编写一个lex程序，该程序拷贝一个文件，并将文件中的每个非空的空白符序列替换为单个空格。

mylexer.l

```
%{/*************************************************************************
Exercise 3.5.2
Author: Wei Lan
Date: 2011年4月21日
*************************************************************************
%}

////////////////////////////////////////////////////////////////////////////
// declarations section
%%

////////////////////////////////////////////////////////////////////////////
// rules section
[ \t]+   {printf(" ");}
\n|.     {printf("%s",yytext);}

%%
////////////////////////////////////////////////////////////////////////////
// programs section
#pragma comment(lib,"yl.lib")
int main(void)
{
    yyin=stdin;
    return yylex();
}
```

```
// rules section
[ \t]+   {printf(" ");}
\n|.     {printf("%s",yytext);}
```

[\t]+ {printf(“ ”);}

表示遇到连续多个空格符或\tab制表符，则替换为一个空格符

\n|. {printf(“%s”,yytext);}

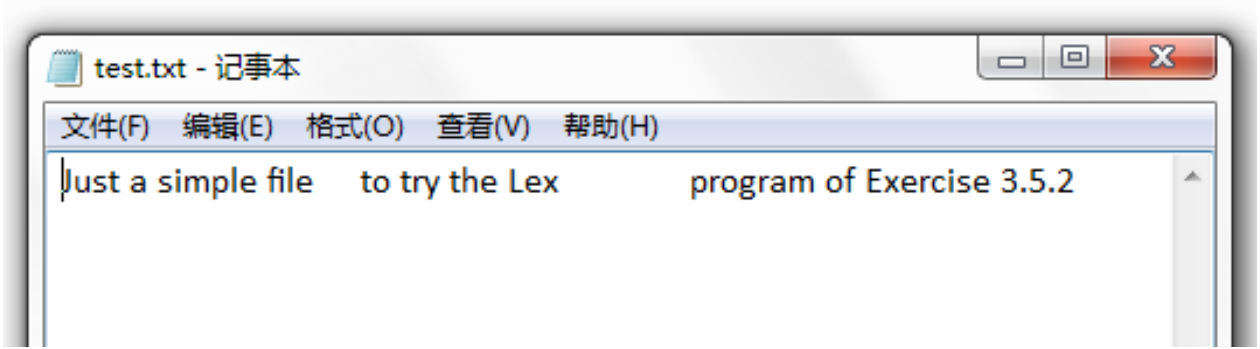
.表示匹配除\n以外的所有字符（通常作为最后一个规则），yytext总是指向当前获得匹配的字符串，这个规则表示其余的语句均照常打印出来。

```
// programs section
#pragma comment(lib,"yl.lib")
int main(void)
{
    yyin=stdin;
    return yylex();
}
```

//program section

- #pragma 包含特定的库文件
- stdin为标准输入，yyin用于读取输入流，yylex()函数表示开始对yyin中的符号进行词法分析。

读入文件：



编译后，运行 ./a.exe < a.txt

替换后结果：

```
Just a simple file to try the Lex program of Exercise 3.5.2
```

The End

第4章 语法分析1
自顶向下分析概述

南信大计软院
凌妙根
2025年秋

NUIST

主要内容

- 自顶向下分析
- 文法转换：消除左递归
- LL(1)文法

1．自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树

1．自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
- 可以看成是从文法开始符号S推导出词串w的过程
  - 例

文法

① $E \rightarrow E+E$

② $E \rightarrow E*E$

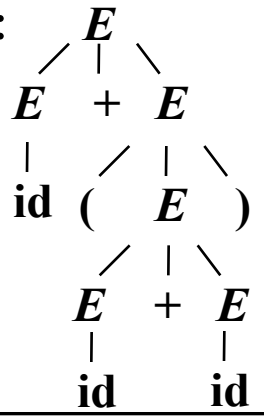
③ $E \rightarrow (E)$

④ $E \rightarrow \mathrm{id}$

输入

id + (id + id)

分析树：



推导过程：$E \Rightarrow E+E$

$\Rightarrow E+(E)$

$\Rightarrow E+(E+E)$

$\Rightarrow E+(\mathrm{id}+E)$

$\Rightarrow \mathrm{id}+(\mathrm{id}+E)$

$\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})$

- 每一步推导中，都需要做两个选择
  - 替换当前句型中的哪个非终结符
  - 用该非终结符的哪个候选式进行替换

最左推导(*Left-most Derivation*)

- 在最左推导中，总是选择每个句型的最左非终结符进行替换
  - 例

文法
① $E \to E + E$
② $E \to E * E$
③ $E \to (E)$
④ $E \to \mathrm{id}$
输入
$\mathrm{id} + (\mathrm{id} + \mathrm{id})$

推导过程

最左推导 ↓

$$
\begin{aligned}
E &\Rightarrow_{lm} E + E \\
&\Rightarrow_{lm} \mathrm{id} + E \\
&\Rightarrow_{lm} \mathrm{id} + (E) \\
&\Rightarrow_{lm} \mathrm{id} + (E + E) \\
&\Rightarrow_{lm} \mathrm{id} + (\mathrm{id} + E) \\
&\Rightarrow_{lm} \mathrm{id} + (\mathrm{id} + \mathrm{id})
\end{aligned}
$$

↑ 最右归约

- 如果$S \Rightarrow^*_{lm} \alpha$，则称$\alpha$是当前文法的最左句型(*left-sentential form*)

最右推导(*Right-most Derivation*)

- 在最右推导中，总是选择每个句型的最右非终结符进行替换
  - 例

文法

① $E \to E + E$

② $E \to E * E$

③ $E \to (E)$

④ $E \to \mathrm{id}$

输入

$\mathrm{id} + (\mathrm{id} + \mathrm{id})$

推导过程

最右推导

$E \Rightarrow_{rm} E + E$

$\Rightarrow_{rm} E + (E)$

$\Rightarrow_{rm} E + (E + E)$

$\Rightarrow_{rm} E + (E + \mathrm{id})$

$\Rightarrow_{rm} E + (\mathrm{id} + \mathrm{id})$

$\Rightarrow_{rm} \mathrm{id} + (\mathrm{id} + \mathrm{id})$

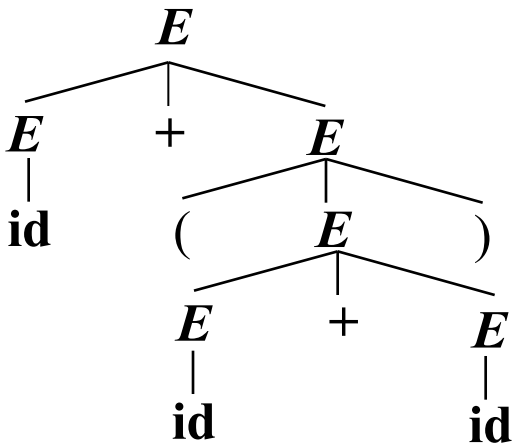
最左归约

- 在自底向上的分析中，总是采用最左归约的方式，因此把最左归约称为规范归约，而最右推导相应地称为规范推导

最左推导和最右推导的唯一性

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+(E) \\
&\Rightarrow E+(E+E) \\
&\Rightarrow E+(\mathrm{id}+E) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+E) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow \mathrm{id}+E \\
&\Rightarrow \mathrm{id}+(E) \\
&\Rightarrow \mathrm{id}+(E+E) \\
&\Rightarrow \mathrm{id}+(E+\mathrm{id}) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$



$$
\begin{aligned}
E &\Rightarrow_{lm} E+E \\
&\Rightarrow_{lm} \mathrm{id}+E \\
&\Rightarrow_{lm} \mathrm{id}+(E) \\
&\Rightarrow_{lm} \mathrm{id}+(E+E) \\
&\Rightarrow_{lm} \mathrm{id}+(\mathrm{id}+E) \\
&\Rightarrow_{lm} \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

$$
\begin{aligned}
E &\Rightarrow_{rm} E+E \\
&\Rightarrow_{rm} E+(E) \\
&\Rightarrow_{rm} E+(E+E) \\
&\Rightarrow_{rm} E+(E+\mathrm{id}) \\
&\Rightarrow_{rm} E+(\mathrm{id}+\mathrm{id}) \\
&\Rightarrow_{rm} \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

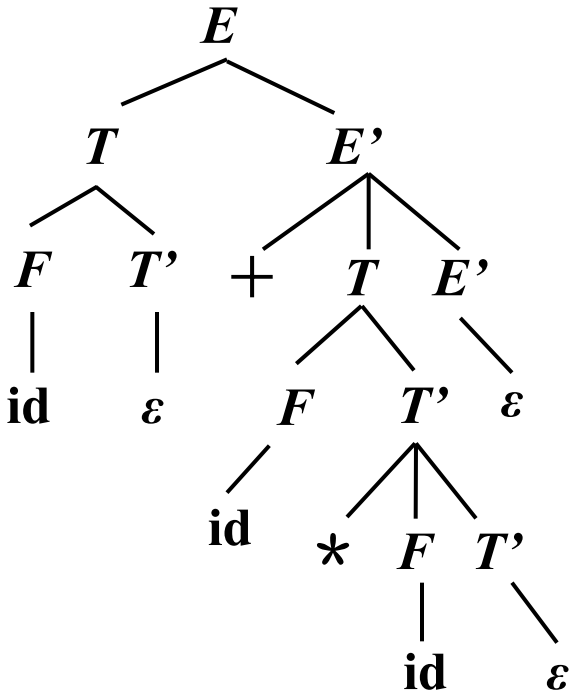
自顶向下的语法分析采用最左推导方式

- 总是选择每个句型的最左非终结符进行替换
- 根据输入流中的下一个终结符，选择最左非终结符的一个候选式

例

- 文法
  - ① $E \rightarrow T\ E'$
  - ② $E' \rightarrow +\ T\ E' \mid \varepsilon$
  - ③ $T \rightarrow F\ T'$
  - ④ $T' \rightarrow *\ F\ T' \mid \varepsilon$
  - ⑤ $F \rightarrow (\ E\ ) \mid \mathbf{id}$
- 输入

  id + id * id

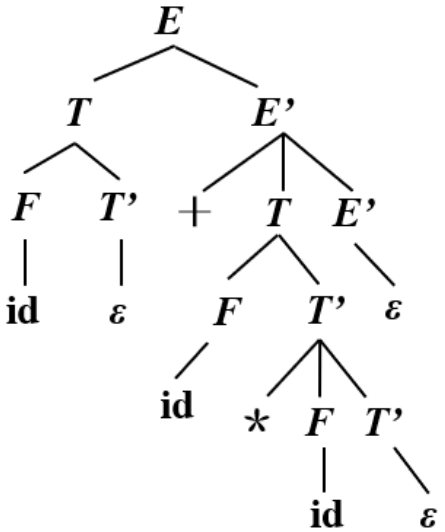


自顶向下语法分析的通用形式

- 递归下降分析 (*Recursive-Descent Parsing*)
  - 由一组过程组成，每个过程对应一个非终结符
  - 从文法开始符号*S*对应的过程开始，递归调用文法中其它非终结符对应的过程。如果*S*对应的过程体恰好扫描了整个输入串，则成功完成语法分析

可能需要回溯(*backtracking*)，导致效率较低

```
void A( ) {
1)    选择一个A产生式，A →X1 X2 … Xk ;
2)    for ( i = 1 to k ) {
3)        if ( Xi是一个非终结符号)
4)            调用过程 Xi( ) ;
5)        else if ( Xi 等于当前的输入符号a)
6)            读入下一个输入符号;
7)        else /* 发生了一个错误 */ ;
      }
}
```



预测分析法/LL(1)分析法 (*Predictive Parsing*)

- 预测分析是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来选择正确的*A*-产生式。
  - 可以对某些文法构造出向前看*k*个输入符号的预测分析器，该类文法有时也称为***LL(k)* 文法类**
- 预测分析不需要回溯，是一种确定的自顶向下分析方法

2. 预测分析法首先要文法转换

同一非终结符的多个候选式存在共同前缀，可能产生回溯现象

- 例1
  - 文法G

$$S \rightarrow aAd \mid aBe$$

$$A \rightarrow c$$

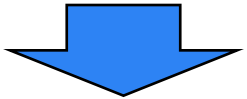
$$B \rightarrow b$$

  - 输入

$$a\ b\ c$$

解决方法：提取左公因子(*Left Factoring*)

- 例
  - 文法G
    - $S \rightarrow aAd \mid aBe$
    - $A \rightarrow c$
    - $B \rightarrow b$



通过改写产生式来推迟决定，等读入了足够多的输入，获得足够信息后再做出正确的选择

  - 文法G'
    - $S \rightarrow aS'$
    - $S' \rightarrow Ad \mid Be$
    - $A \rightarrow c$
    - $B \rightarrow b$

提取左公因子算法

- 输入：文法G
- 输出：等价的提取了左公因子的文法
- 方法：

对于每个非终结符A，找出它的两个或多个选项之间的最长公共前缀α。如果$\alpha \neq \varepsilon$，即存在一个非平凡的(*nontrivial*)公共前缀，那么将所有A-产生式

$$A \to \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n | \gamma_1 | \gamma_2 | \ldots | \gamma_m$$

替换为

$$A \to \alpha A' | \gamma_1 | \gamma_2 | \ldots | \gamma_m$$

$$A' \to \beta_1 | \beta_2 | \ldots | \beta_n$$

其中，γ_i 表示所有不以α开头的产生式体；A'是一个新的非终结符。不断应用这个转换，直到每个非终结符的任意两个产生式体都没有公共前缀为止

左递归文法会使递归下降分析器陷入无限循环

- 例2
  - 文法G

$$E \to E + T \mid E - T \mid T$$
$$T \to T * F \mid T / F \mid F$$
$$F \to (E) \mid \text{id}$$

$$E \Rightarrow E + T$$
$$\Rightarrow E + T + T$$
$$\Rightarrow E + T + T + T$$
$$\Rightarrow \cdots$$

  - 输入

id + id * id

含有$A \to A\alpha$形式产生式的文法称为是直接左递归的(*immediate left recursive*)

如果一个文法中有一个非终结符A使得对某个串α存在一个推导$A \Rightarrow^{+} A\alpha$，那么这个文法就是左递归的

经过两步或两步以上推导产生的左递归称为是间接左递归的

消除直接左递归

$A \to A\alpha \mid \beta (\alpha \neq \varepsilon, \beta$不以$A$开头$)$ $r=\beta\alpha^*$

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

事实上，这种消除过程就是把左递归转换成了右递归

$A \Rightarrow A\alpha$
$\Rightarrow A\alpha\alpha$
$\Rightarrow A\alpha\alpha\alpha$
…
$\Rightarrow A\alpha \ldots\alpha$
$\Rightarrow \beta\,\alpha \ldots\alpha$

➢ 例

$E \to E + T \mid T$ （α = $+T$，β = T）

$T \to T * F \mid F$ （α = $*F$，β = F）

$F \to (E) \mid \mathrm{id}$

$E \to T\,E'$

$E' \to +\,T\,E' \mid \varepsilon$

$T \to F\,T'$

$T' \to *\,F\,T' \mid \varepsilon$

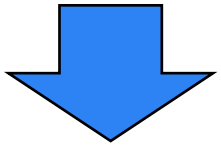
$F \to (E) \mid \mathrm{id}$

$A' \Rightarrow \alpha A'$
$\Rightarrow \alpha\alpha A'$
$\Rightarrow \alpha\alpha\alpha A'$
…
$\Rightarrow \alpha \ldots\alpha A'$
$\Rightarrow \alpha \ldots\alpha$

消除直接左递归的一般形式

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

$$(\alpha_i \neq \varepsilon, \beta_j \text{不以} A \text{开头})$$



$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$

$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$$

消除左递归是要付出代价的——引进了一些非终结符和ε_产生式

消除间接左递归

➢ 例

$$S \rightarrow A\,a \mid b$$

$$S \Rightarrow Aa$$
$$\Rightarrow Sda$$

$$A \rightarrow A\,c \mid S\,d \mid \varepsilon$$

➢ 将S的定义代入A-产生式，得：

$$A \rightarrow A\,c \mid A\,a\,d \mid b\,d \mid \varepsilon$$

➢ 消除A-产生式的直接左递归？

消除间接左递归

$$A \rightarrow A\,c \mid A\,a\,d \mid b\,d \mid \varepsilon$$

➢ 消除直接左递归：

$$A \rightarrow A\,(c \mid a\,d) \mid b\,d \mid \varepsilon$$

$$A \rightarrow A\,(c \mid a\,d) \mid (b\,d \mid \varepsilon)$$

应用公式得到：

$$A \rightarrow (b\,d \mid \varepsilon)\,A'$$

$$A' \rightarrow (c \mid a\,d)\,A' \mid \varepsilon$$

展开：

$$A \rightarrow b\,d\,A' \mid A'$$

$$A' \rightarrow c\,A' \mid a\,d\,A' \mid \varepsilon$$

消除间接左递归

➢ 例

$$S \to A\,a \mid b$$

$$A \to A\,c \mid S\,d \mid \varepsilon$$

$S \Rightarrow Aa$
$\Rightarrow Sda$

➢ 消除A-产生式的直接左递归：

$$S \to A\,a \mid b$$

$$A \to b\,d\,A' \mid A'$$

$$A' \to c\,A' \mid a\,d\,A' \mid \varepsilon$$

消除左递归算法 蓝书P124/电子书P135 图4-11

- 输入：不含循环推导（即形如$A \Rightarrow^{+} A$的推导）和ε-产生式的文法G
- 输出：等价的无左递归文法
- 方法：

1) 按照某个顺序将非终结符号排序为A_1，A_2，…，A_n .
2) for (从1到n的每个i) {
3) for (从1到i -1的每个j) {
4) 将每个形如$A_i \rightarrow A_j \gamma$产生式的$A_j$进行替换，利用$A_j \rightarrow \delta_1|\delta_2|\ldots|\delta_k$替换为产生式组 $A_i \rightarrow \delta_1 \gamma|\delta_2 \gamma|\ldots|\delta_k \gamma$
5) }
6) 消除A_i产生式之间的立即左递归
7) }

1)按照某个顺序将非终结符号排序为A_1，A_2，…，A_n .

2)for (从1到n的每个i) {

3)　　for (从1到i -1的每个j) {

4)　　将每个形如$A_i \rightarrow A_j \gamma$产生式的$A_j$进行替换，利用$A_j \rightarrow \delta_1|\delta_2|\ldots|\delta_k$替换为产生式组 $A_i \rightarrow \delta_1 \gamma|\delta_2 \gamma|\ldots|\delta_k \gamma$

　　}

5)　　消除A_i产生式之间的立即左递归

6) }

- i=1,j=Ø　　$A_1 \rightarrow A_1 \gamma$　　$A_1 \rightarrow A_2 \gamma$

  $A_1 \rightarrow A_k \gamma'$(k>1)　　$A_2 \rightarrow A_1 \gamma'$

- i=2,j=1　　$A_2 \rightarrow A_1 \gamma$

  $A_2 \rightarrow A_k \gamma'$(k ≥ 2)

  $A_2 \rightarrow A_k \gamma'$(k >2)

- i=3,j=1,2

$$A_3 \rightarrow A_1\,\gamma$$

$$A_3 \rightarrow A_2\,\gamma$$

$$A_3 \rightarrow A_k\,\gamma\,'(k \geq 3)$$

$$A_3 \rightarrow A_k\,\gamma\,'(k > 3)$$

...

- i=n,j=1: i-1

$$A_n \rightarrow A_k\,\gamma(k \geq n)$$

$$A_n \rightarrow A_k\,\gamma\,'(k > n)$$

3. LL(1)文法

S_文法

假如允许S_文法包含ε产生式，将会产生什么问题？

- 预测分析法的工作过程
  - 从文法开始符号出发，在每一步推导过程中根据当前句型的最左非终结符A和当前输入符号a，选择正确的A-产生式。为保证分析的确定性，选出的候选式必须是唯一的。
- *S*_文法（简单的确定性文法，*Korenjak & Hopcroft*，1966）

每个产生式的右部都以终结符开始

同一非终结符的各个候选式的首终结符都不同

*S*_文法不含ε产生式

例

➢ 文法	➢ 输入 $a\ d\ a$	➢ 输入 $a\ d\ e$
① $S \to aBC$ ② $B \to bC$ ③ $B \to dB$ ④ $B \to \varepsilon$ ⑤ $C \to c$ ⑥ $C \to a$ ⑦ $D \to e$	➢ 推导 S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$ $\Rightarrow ada$	➢ 推导 S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$

可以紧跟B后面出现的终结符：c、a

➢ 什么时候使用ε产生式是有效的呢?

➢ 如果当前某非终结符A与当前输入符a不匹配时，若存在$A \to \varepsilon$，可以通过检查a是否可以出现在A的后面，来决定是否使用产生式$A \to \varepsilon$（若文法中无$A \to \varepsilon$，则应报错）

➢非终结符A的后继符号集

➢ 可能在某个句型中紧跟在A后边的终结符a的集合，记为$FOLLOW(A)$

$FOLLOW(A)=\{a \mid S \Rightarrow^{*} \alpha A a \beta, a \in V_T, \alpha,\beta \in (V_T \cup V_N)^{*}\}$

例

(1) $S \rightarrow aBC$

(2) $B \rightarrow bC$ ← b 输入

(3) $B \rightarrow dB$ ← d

(4) $B \rightarrow \varepsilon$ ← $\{a, c\}$

(5) $C \rightarrow c$

(6) $C \rightarrow a$

$FOLLOW(B) = \{a,c\}$

如果A是某个句型的最右符号，
则将结束符“$”添加到$FOLLOW(A)$中

产生式的可选集

- 产生式$A\rightarrow\beta$的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为$SELECT(A\rightarrow\beta)$
  - $SELECT(A\rightarrow a\beta)=\{a\}$
  - $SELECT(A\rightarrow\varepsilon)=FOLLOW(A)$

例

➢ 文法	➢ 输入 a d a	➢ 输入 a d e
① $S \to aBC$ ② $B \to bC$ ③ $B \to dB$ ④ $B \to \varepsilon$ ⑤ $C \to c$ ⑥ $C \to a$ ⑦ $D \to e$	➢ 推导 S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$ $\Rightarrow ada$	➢ 推导 S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$

可以紧跟B后面出现的终结符：c、a

➢ 什么时候使用ε产生式?

➢ 如果当前某非终结符A与当前输入符a不匹配时，若存在$A \to \varepsilon$，可以通过检查a是否可以出现在A的后面，来决定是否使用产生式$A \to \varepsilon$（若文法中无$A \to \varepsilon$，则应报错）

产生式的可选集

- 产生式$A\rightarrow\beta$的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为$SELECT(A\rightarrow\beta)$
  - $SELECT(A\rightarrow a\beta)=\{a\}$
  - $SELECT(A\rightarrow\varepsilon)=FOLLOW(A)$
- q_文法
  - 每个产生式的右部或为ε，或以终结符开始
  - 具有相同左部的产生式有不相交的可选集

q_文法不含右部以非终结符打头的产生式

串首终结符集

- 串首终结符
  - 串首第一个符号，并且是终结符。简称首终结符
- 给定一个文法符号串α，α的串首终结符集*FIRST*(α)被定义为可以从α推导出的所有串首终结符构成的集合。如果$\alpha \Rightarrow^* \varepsilon$，那么ε也在*FIRST*(α)中
  - 对于$\forall\alpha\in(V_T\cup V_N)^+$, $FIRST(\alpha)=\{a \mid \alpha \Rightarrow^* a\beta,\ a\in V_T,\ \beta\in(V_T\cup V_N)^*\}$;
  - 如果$\alpha \Rightarrow^* \varepsilon$，那么$\varepsilon\in FIRST(\alpha)$
- 产生式$A\rightarrow\alpha$的可选集*SELECT*
  - 如果$\varepsilon\notin FIRST(\alpha)$，那么$SELECT(A\rightarrow\alpha)= FIRST(\alpha)$
  - 如果$\varepsilon\in FIRST(\alpha)$，那么$SELECT(A\rightarrow\alpha)=(FIRST(\alpha)-\{\varepsilon\})\cup FOLLOW(A)$

$LL(1)$文法

- 文法G是$LL(1)$的，当且仅当G的任意两个具有相同左部的产生式$A \to \alpha \mid \beta$满足下面的条件：
  - 不存在终结符a使得α和β都能够推导出以a开头的串
  - α和β至多有一个能推导出ε
  - 如果$\beta \Rightarrow^* \varepsilon$，则$FIRST(\alpha) \cap FOLLOW(A) = \Phi$；

    如果$\alpha \Rightarrow^* \varepsilon$，则$FIRST(\beta) \cap FOLLOW(A) = \Phi$；

同一非终结符的各个产生式的可选集互不相交

可以为$LL(1)$文法构造预测分析器

$LL(1)$文法

- 文法G是$LL(1)$的，当且仅当G的任意两个具有相同左部的产生式$A \to \alpha \mid \beta$ 满足下面的条件：
  - 不存在终结符a使得α 和β都能够推导出以a开头的串
  - α 和β至多有一个能推导出ε
  - 如果$\beta \Rightarrow^* \varepsilon$，则$FIRST(\alpha) \cap FOLLOW(A) = \Phi$；

    如果$\alpha \Rightarrow^* \varepsilon$，则$FIRST(\beta) \cap FOLLOW(A) = \Phi$；

- 第一个“L”表示从左向右扫描输入
- 第二个“L”表示产生最左推导
- “1”表示在每一步中只需要向前看一个输入符号来决定语法分析动作

总结

- 1.计算机实现自顶向下分析？
- 递归下降分析法
- 2. 文法转换：
- 提取左公因子和 消除左递归
- 3. LL(1)文法的First集和Select集

内容回顾

- 识别单词的DFA
- 词法分析器的实现
- 词法分析器自动生成工具：lex

The End