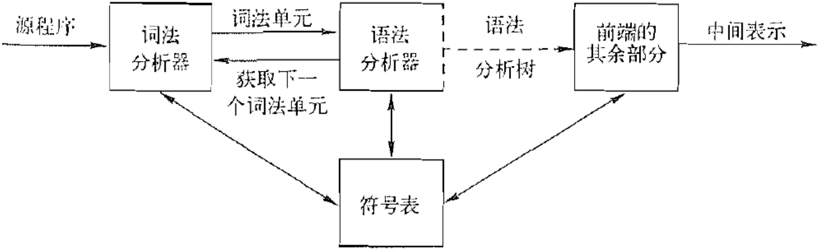
语法分析

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 30 日

输入：词法单元流 & **语言的语法规则**



输出：语法分析树 (Parse Tree)

语法分析**举例**

⟨Stmt⟩	→	⟨Id⟩ `=` ⟨Expr⟩ `;`
⟨Stmt⟩	→	`{` ⟨StmtList⟩ `}`
⟨Stmt⟩	→	`if` `(` ⟨Expr⟩ `)` ⟨Stmt⟩
⟨StmtList⟩	→	⟨Stmt⟩
⟨StmtList⟩	→	⟨StmtList⟩ ⟨Stmt⟩
⟨Expr⟩	→	⟨Id⟩
⟨Expr⟩	→	⟨Num⟩
⟨Expr⟩	→	⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
⟨Id⟩	→	`x`
⟨Id⟩	→	`y`
⟨Num⟩	→	`0`
⟨Num⟩	→	`1`
⟨Num⟩	→	`9`
⟨Optr⟩	→	`>`
⟨Optr⟩	→	`+`

⟨Stmt⟩

if (⟨Expr⟩) ⟨Stmt⟩

if (⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩

if (⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩

if (x ⟨Optr⟩ ⟨Expr⟩) ⟨Stmt⟩

if (x > ⟨Expr⟩) ⟨Stmt⟩

if (x > ⟨Num⟩) ⟨Stmt⟩

if (x > 9) ⟨Stmt⟩

if (x > 9) { ⟨StmtList⟩ }

if (x > 9) { ⟨StmtList⟩ ⟨Stmt⟩ }

if (x > 9) { ⟨Stmt⟩ ⟨Stmt⟩ }

if (x > 9) { ⟨Id⟩ = ⟨Expr⟩ ; ⟨Stmt⟩ }

if (x > 9) { x = ⟨Expr⟩ ; ⟨Stmt⟩ }

if (x > 9) { x = ⟨Num⟩ ; ⟨Stmt⟩ }

if (x > 9) { x = 0 ; ⟨Stmt⟩ }

if (x > 9) { x = 0 ; ⟨Id⟩ = ⟨Expr⟩ ; }

if (x > 9) { x = 0 ; y = ⟨Expr⟩ ; }

if (x > 9) { x = 0 ; y = ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩ ; }

if (x > 9) { x = 0 ; y = ⟨Id⟩ ⟨Optr⟩ ⟨Expr⟩ ; }

if (x > 9) { x = 0 ; y = y ⟨Optr⟩ ⟨Expr⟩ ; }

if (x > 9) { x = 0 ; y = y + ⟨Expr⟩ ; }

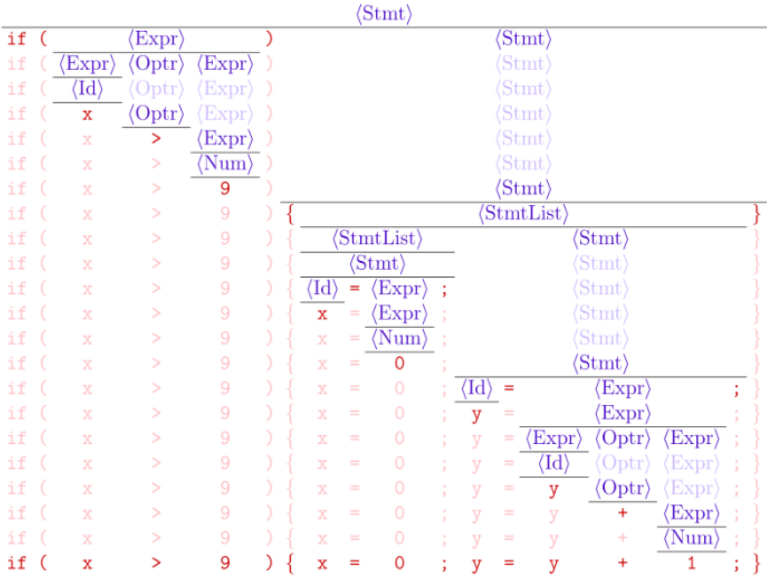
if (x > 9) { x = 0 ; y = y + ⟨Num⟩ ; }

if (x > 9) { x = 0 ; y = y + 1 ; }

语法分析阶段的主题之一: 上下文无关文法

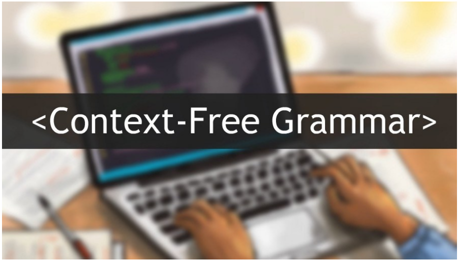
$$
\begin{array}{rcl}
\langle \text{Stmt} \rangle & \rightarrow & \langle \text{Id} \rangle \; \texttt{=} \; \langle \text{Expr} \rangle \; \texttt{;} \\
\langle \text{Stmt} \rangle & \rightarrow & \texttt{\{} \; \langle \text{StmtList} \rangle \; \texttt{\}} \\
\langle \text{Stmt} \rangle & \rightarrow & \texttt{if} \; \texttt{(} \; \langle \text{Expr} \rangle \; \texttt{)} \; \langle \text{Stmt} \rangle \\
\langle \text{StmtList} \rangle & \rightarrow & \langle \text{Stmt} \rangle \\
\langle \text{StmtList} \rangle & \rightarrow & \langle \text{StmtList} \rangle \; \langle \text{Stmt} \rangle \\
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Id} \rangle \\
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Num} \rangle \\
\langle \text{Expr} \rangle & \rightarrow & \langle \text{Expr} \rangle \; \langle \text{Optr} \rangle \; \langle \text{Expr} \rangle \\
\langle \text{Id} \rangle & \rightarrow & \texttt{x} \\
\langle \text{Id} \rangle & \rightarrow & \texttt{y} \\
\langle \text{Num} \rangle & \rightarrow & \texttt{0} \\
\langle \text{Num} \rangle & \rightarrow & \texttt{1} \\
\langle \text{Num} \rangle & \rightarrow & \texttt{9} \\
\langle \text{Optr} \rangle & \rightarrow & \texttt{>} \\
\langle \text{Optr} \rangle & \rightarrow & \texttt{+}
\end{array}
$$

语法分析阶段的主题之二: **构建语法分析树**



语法分析阶段的主题之三：**错误恢复**

报错、**恢复**、继续分析



上下文无关文法

Definition (Context-Free Grammar (CFG); 上下文无关文法)

上下文无关文法 G 是一个四元组 $G = (T, N, P, S)$:

- T 是**终结符号** (Terminal) 集合, 对应于词法分析器产生的词法单元;
- N 是**非终结符号** (Non-terminal) 集合;
- P 是**产生式** (Production) 集合;

$$\boxed{A \in N \longrightarrow \alpha \in (T \cup N)^*}$$

  头部/左部 (Head) A: **单个**非终结符

  体部/右部 (Body) α: 终结符与非终结符构成的串, 也可以是空串 ϵ

- S 为**开始** (Start) 符号。要求 $S \in N$ 且唯一。

$$G = (\{a, b\}, \{S\}, P, S)$$

$$\boxed{\begin{aligned} S &\to aSb \\ S &\to \epsilon \end{aligned}}$$

$$G = (\{(,)\}, \{S\}, P, S)$$

$$\boxed{\begin{array}{c} S \to SS \\ S \to (S) \\ S \to () \\ S \to \epsilon \end{array}}$$

$$
\begin{aligned}
stmt \;\rightarrow\; & \textbf{if } expr \textbf{ then } stmt \\
\mid\; & \textbf{if } expr \textbf{ then } stmt \boxed{\textbf{else } stmt} \\
\mid\; & \boxed{\textbf{other}}
\end{aligned}
$$

条件语句文法

悬空 (Dangling)-else 文法

$S \rightarrow$ if E then S else S
$S \rightarrow$ begin S L
$S \rightarrow$ print E

$L \rightarrow$ end
$L \rightarrow$; S L

$E \rightarrow$ num $=$ num

约定: 如果没有明确指定, 第一个产生式的头部就是开始符号

关于**终结符号**的约定

1）下述符号是终结符号：

① 在字母表里排在前面的小写字母，比如 a、b、c。

② 运算符号，比如 +、* 等。

③ 标点符号，比如括号、逗号等。

④ 数字 0、1、…、9。

⑤ 黑体字符串，比如 **id** 或 **if**。每个这样的字符串表示一个终结符号。

关于**非终结符号**的约定

2）下述符号是非终结符号：

① 在字母表中排在前面的大写字母，比如 A、B、C。

② $\boxed{\text{字母 } S}$。它出现时通常表示开始符号。

③ 小写、斜体的名字，比如 *expr* 或 *stmt*。

语义: 上下文无关文法 G 定义了一个**语言** $L(G)$

语义: 上下文无关文法 G 定义了一个**语言** $L(G)$

语言是**串**的集合

串从何来?

推导 (Derivation)

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

推导即是将某个产生式的左边**替换**成它的右边

每一步推导需要选择**替换哪个非终结符号**, 以及**使用哪个产生式**

推导 (Derivation)

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

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

推导 (Derivation)

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \textbf{id}}$$

$$E \implies -E \implies -(E) \implies -(E+E) \implies -(\textbf{id}+E) \implies -(\textbf{id}+\textbf{id})$$

$E \implies -E$: 经过一步推导得出

$E \overset{+}{\implies} -(\textbf{id} + E)$: 经过一步或多步推导得出

$E \overset{*}{\implies} -(\textbf{id} + E)$: 经过零步或多步推导得出

推导 (Derivation)

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

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

$E \implies -E$: 经过一步推导得出

$E \overset{+}{\Longrightarrow} -(\mathbf{id} + E)$: 经过一步或多步推导得出

$E \overset{*}{\Longrightarrow} -(\mathbf{id} + E)$: 经过零步或多步推导得出

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

Definition (Sentential Form; 句型)

如果 $S \stackrel{*}{\Rightarrow} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**句型**。

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \textbf{id}}$$

$$E \implies -E \implies -(E) \implies -(E{+}E) \implies -(\textbf{id} + E) \implies -(\textbf{id} + \textbf{id})$$

Definition (Sentential Form; 句型)

如果 $S \stackrel{*}{\Rightarrow} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**句型**。

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

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

Definition (Sentence; 句子)

如果 $S \stackrel{*}{\Rightarrow} w$, 且 $w \in T^*$, 则称 w 是文法 G 的一个**句子**。

Definition (文法 G 生成的语言 $L(G)$)

文法 G 的**语言** $L(G)$ 是它能推导出的**所有句子**构成的集合。

$$w \in L(G) \iff S \xRightarrow{*} w$$

关于文法 G 的**两个基本问题:**

- **Membership 问题:** 给定字符串 $x \in T^*$, $x \in L(G)$?
- $L(G)$ 究竟是什么?

给定字符串 $x \in T^*$, $x \in L(G)$?

(即, 检查 x 是否符合文法 G)

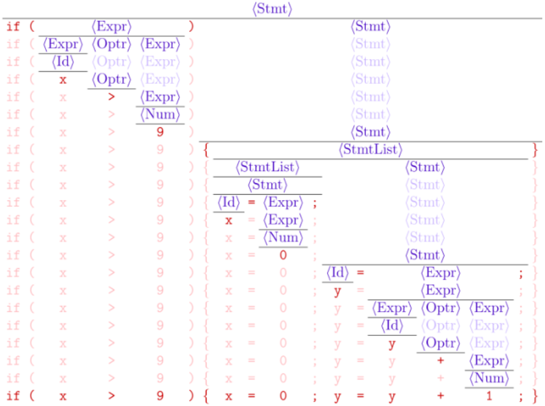
给定字符串 $x \in T^*$, $x \in L(G)$?

(即, 检查 x 是否符合文法 G)

这就是**语法分析器**的任务:

为输入的词法单元流寻找推导、**构建语法分析树**, 或者报错

根节点是文法 G 的起始符号



叶子节点是输入的词法单元流

常用的语法分析器以**自顶向下**或**自底向上**的方式构建中间部分

$L(G)$ 是什么?

这是**程序设计语言设计者**需要考虑的问题

$$\boxed{\begin{aligned} S &\to SS \\ S &\to (S) \\ S &\to () \\ S &\to \epsilon \end{aligned}}$$

$L(G) =$

$$\boxed{\begin{aligned} S &\to SS \\ S &\to (S) \\ S &\to () \\ S &\to \epsilon \end{aligned}}$$

$$L(G) = \{\text{良匹配括号串}\}$$

$$S \to SS$$
$$S \to (S)$$
$$S \to ()$$
$$S \to \epsilon$$

$L(G) = \{$良匹配括号串$\}$

$$S \to aSb$$
$$S \to \epsilon$$

$L(G) =$

$$\boxed{\begin{aligned} S &\to SS \\ S &\to (S) \\ S &\to () \\ S &\to \epsilon \end{aligned}}$$

$$L(G) = \{\text{良匹配括号串}\}$$

$$\boxed{\begin{aligned} S &\to aSb \\ S &\to \epsilon \end{aligned}}$$

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

字母表 $\Sigma = \{a, b\}$ 上的所有**回文串** (Palindrome) 构成的语言

字母表 $\Sigma = \{a, b\}$ 上的所有**回文串** (Palindrome) 构成的语言

$$\boxed{\begin{aligned} S &\to aSa \\ S &\to bSb \\ S &\to a \\ S &\to b \\ S &\to \epsilon \end{aligned}}$$

字母表 $\Sigma = \{a, b\}$ 上的所有**回文串** (Palindrome) 构成的语言

$$\boxed{\begin{aligned} S &\to aSa \\ S &\to bSb \\ S &\to a \\ S &\to b \\ S &\to \epsilon \end{aligned}}$$

$$S \to aSa \mid bSb \mid a \mid b \mid \epsilon$$

$$\{b^n a^m b^{2n} \mid n \geq 0, m \geq 0\}$$

$$\{b^n a^m b^{2n} \mid n \geq 0, m \geq 0\}$$

$$\boxed{\begin{aligned} S &\to bSbb \mid A \\ A &\to aA \mid \epsilon \end{aligned}}$$

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$$

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$$

$$\boxed{V \to aVbV \mid bVaV \mid \epsilon}$$

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$$

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$$

$$\boxed{\begin{aligned} S &\to T \mid U \\ T &\to VaT \mid VaV \\ U &\to VbU \mid VbV \\ V &\to aVbV \mid bVaV \mid \epsilon \end{aligned}}$$

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$$

$$\boxed{\begin{aligned} S &\to T \mid U \\ T &\to VaT \mid VaV \\ U &\to VbU \mid VbV \\ V &\to aVbV \mid bVaV \mid \epsilon \end{aligned}}$$

练习（非作业）：证明之

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$
$$S \rightarrow \text{begin } S\ L$$
$$S \rightarrow \text{print } E$$

$$L \rightarrow \text{end}$$
$$L \rightarrow ;\ S\ L$$

$$E \rightarrow \text{num} = \text{num}$$

$$S \to \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S$$
$$S \to \textbf{begin}\ S\ L$$
$$S \to \textbf{print}\ E$$

$$L \to \textbf{end}$$
$$L \to \textbf{;}\ S\ L$$

$$E \to \textbf{num} = \textbf{num}$$

顺序语句、条件语句、打印语句

L-System

(注: 这不是上下文无关文法, 但精神上高度一致, 并且更有趣)

variables : A B

constants : + −

start : A

rules : (A → B−A−B), (B → A+B+A)

angle : 60°

A, B : 向右移动并画线

$+$: 左转

$-$: 右转

每一步都**并行地**应用**所有**规则

$$A$$

$$B - A - B$$

$$A$$

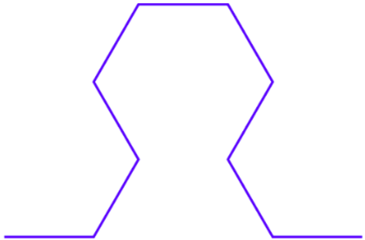
$$B - A - B$$

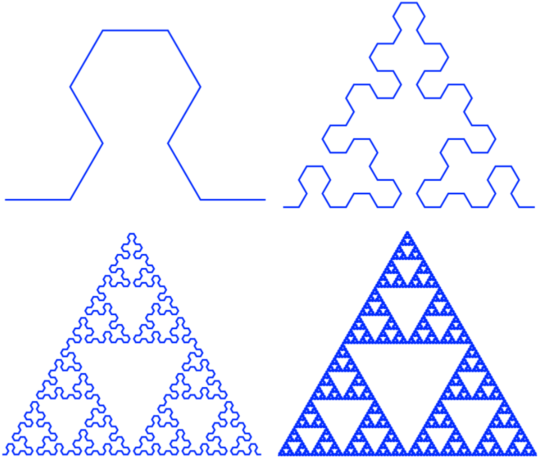
$$A + B + A - B - A - B - A + B + A$$

$$A$$

$$B - A - B$$

$$A + B + A - B - A - B - A + B + A$$





Sierpinski arrowhead curve ($n = 2, 4, 6, 8$)

variables : X Y

constants : F + −

start : FX

rules : (X → X+YF+), (Y → −FX−Y)

angle : 90°

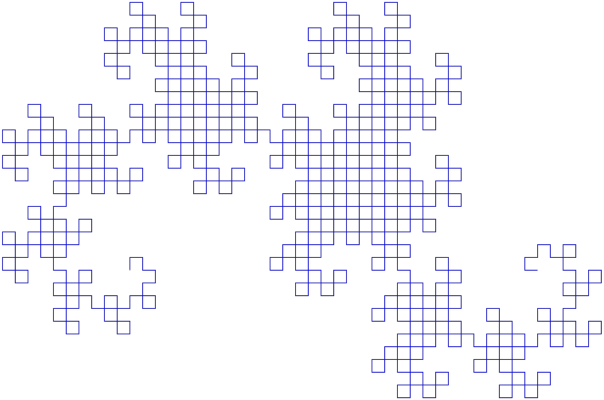
F : 向上移动并画线

$+$: 右转

$-$: 左转

X : 仅用于展开, 在作画时被忽略

每一步都**并行地**应用**所有**规则



Dragon Curve $(n = 10)$

最左（leftmost）推导与最右（rightmost）推导

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

$$E \underset{\text{lm}}{\Longrightarrow} -E \underset{\text{lm}}{\Longrightarrow} -(E) \underset{\text{lm}}{\Longrightarrow} -(E+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+\mathbf{id})$$

最左（leftmost）推导与最右（rightmost）推导

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

$$E \underset{\text{lm}}{\Longrightarrow} -E \underset{\text{lm}}{\Longrightarrow} -(E) \underset{\text{lm}}{\Longrightarrow} -(E+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+\mathbf{id})$$

$E \underset{\text{lm}}{\Longrightarrow} -E$: 经过一步最左推导得出

$E \overset{+}{\underset{\text{lm}}{\Longrightarrow}} -(\mathbf{id}+E)$: 经过一步或多步最左推导得出

$E \overset{*}{\underset{\text{lm}}{\Longrightarrow}} -(\mathbf{id}+E)$: 经过零步或多步最左推导得出

最左 (leftmost) 推导与最右 (rightmost) 推导

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

$$E \underset{\text{lm}}{\Longrightarrow} -E \underset{\text{lm}}{\Longrightarrow} -(E) \underset{\text{lm}}{\Longrightarrow} -(E+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+E) \underset{\text{lm}}{\Longrightarrow} -(\mathbf{id}+\mathbf{id})$$

$E \underset{\text{lm}}{\Longrightarrow} -E$: 经过一步最左推导得出

$E \overset{+}{\underset{\text{lm}}{\Longrightarrow}} -(\mathbf{id}+E)$: 经过一步或多步最左推导得出

$E \overset{*}{\underset{\text{lm}}{\Longrightarrow}} -(\mathbf{id}+E)$: 经过零步或多步最左推导得出

$$E \underset{\text{rm}}{\Longrightarrow} -E \underset{\text{rm}}{\Longrightarrow} -(E) \underset{\text{rm}}{\Longrightarrow} -(E+E) \underset{\text{rm}}{\Longrightarrow} -(E+\mathbf{id}) \underset{\text{rm}}{\Longrightarrow} -(\mathbf{id}+\mathbf{id})$$

Definition (Left-sentential Form; 最左句型)

如果 $S \xRightarrow[\text{lm}]{*} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**最左句型**。

$$E \xRightarrow[\text{lm}]{} -E \xRightarrow[\text{lm}]{} -(E) \xRightarrow[\text{lm}]{} -(E+E) \xRightarrow[\text{lm}]{} -(\mathbf{id}+E) \xRightarrow[\text{lm}]{} -(\mathbf{id}+\mathbf{id})$$

Definition (Left-sentential Form; 最左句型)

如果 $S \xRightarrow[\text{lm}]{*} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**最左句型**。

$$E \xRightarrow[\text{lm}]{} -E \xRightarrow[\text{lm}]{} -(E) \xRightarrow[\text{lm}]{} -(E+E) \xRightarrow[\text{lm}]{} -(\textbf{id}+E) \xRightarrow[\text{lm}]{} -(\textbf{id}+\textbf{id})$$

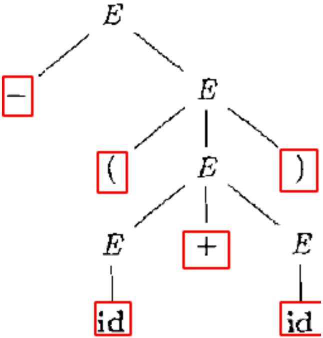
Definition (Right-sentential Form; 最右句型)

如果 $S \xRightarrow[\text{rm}]{*} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**最右句型**。

$$E \xRightarrow[\text{rm}]{} -E \xRightarrow[\text{rm}]{} -(E) \xRightarrow[\text{rm}]{} -(E+E) \xRightarrow[\text{rm}]{} -(E+\textbf{id}) \xRightarrow[\text{rm}]{} -(\textbf{id}+\textbf{id})$$

语法分析树

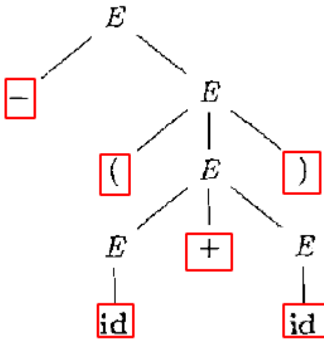
语法分析树是静态的, 它不关心动态的推导顺序



一棵语法分析树对应多个推导

语法分析树

语法分析树是静态的，它不关心动态的推导顺序



一棵语法分析树对应多个推导

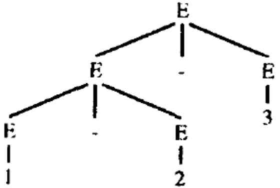
但是，一棵语法分析树与**最左（最右）推导**一一对应

$$\boxed{E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}}$$

$1 - 2 - 3$ 的语法树?

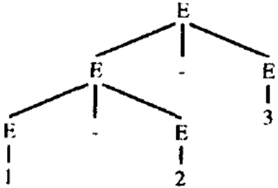
$$\boxed{E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}}$$

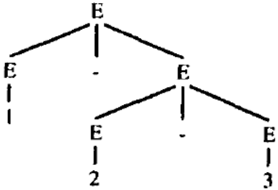
$1 - 2 - 3$ 的语法树?



$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

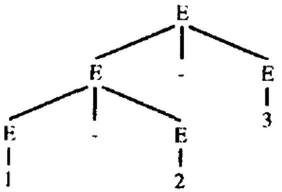
$1 - 2 - 3$ 的语法树?

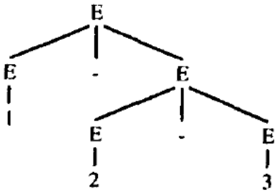




$$E \to E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \mathbf{id} \mid \mathbf{num}$$

$1 - 2 - 3$ 的语法树?





Definition (二义性(Ambiguous) 文法)

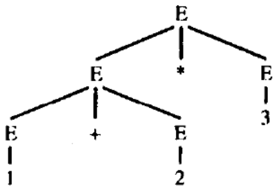
如果 $L(G)$ 中的**某个**句子有**一个以上**语法树/最左推导/最右推导,
则文法 G 是二义性的。

$$\boxed{E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}}$$

$1 + 2 * 3$ 的语法树?

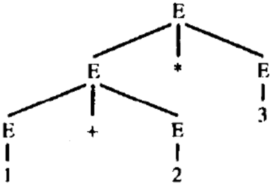
$$E \to E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \mathbf{id} \mid \mathbf{num}$$

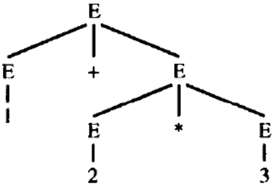
$1 + 2 * 3$ 的语法树?



$$E \to E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

$1 + 2 * 3$ 的语法树?





$$
\begin{aligned}
stmt \;\rightarrow\; & \textbf{if } expr \textbf{ then } stmt \\
\mid\; & \textbf{if } expr \textbf{ then } stmt\ \boxed{\textbf{else } stmt} \\
\mid\; & \boxed{\textbf{other}}
\end{aligned}
$$

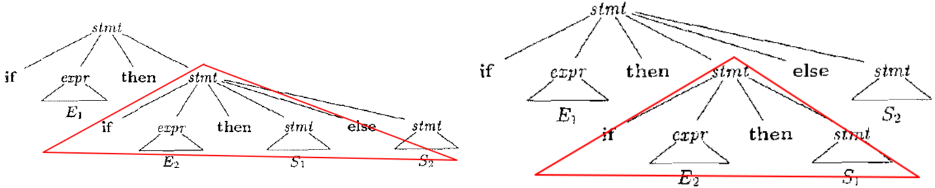
“悬空-else” 文法

$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$

$$
\begin{aligned}
\mathit{stmt} \;\rightarrow\;& \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt} \\
\mid\;& \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \boxed{\textbf{else}\ \mathit{stmt}} \\
\mid\;& \boxed{\textbf{other}}
\end{aligned}
$$

“悬空-else” 文法

if E_1 **then if** E_2 **then** S_1 **else** S_2

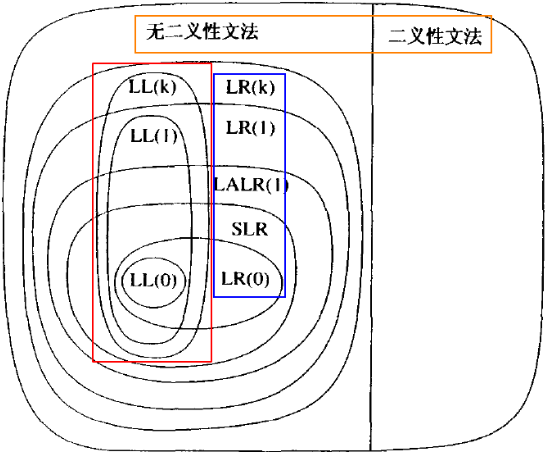


if E_1 **then** (**if** E_2 **then** S_1 **else** S_2)

if E_1 **then** (**if** E_2 **then** S_1) **else** S_2

二义性文法

不同的语法分析树产生不同的语义



所有语法分析器都要求文法是**无二义性**的

二义性文法

Q : 如何**识别**二义性文法?

Q : 如何**消除**文法的二义性?

二义性文法

Q : 如何**识别**二义性文法?

这是**不可判定**的问题

Q : 如何**消除**文法的二义性?

二义性文法

Q：如何**识别**二义性文法?

这是**不可判定**的问题

Q：如何**消除**文法的二义性?

LEARN BY EXAMPLES

$$E \to E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

四则运算均是**左结合**的

优先级: 括号最先, 先乘除后加减

二义性表达式文法以**相同的方式**处理所有的算术运算符

要消除二义性, 需要**区别对待**不同的运算符

$$\boxed{E \to E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \textbf{id} \mid \textbf{num}}$$

四则运算均是**左结合**的

优先级: 括号最先, 先乘除后加减

二义性表达式文法以**相同的方式**处理所有的算术运算符

要消除二义性, 需要**区别对待**不同的运算符

将运算的“先后”顺序信息编码到语法树的“层次”结构中

$$\boxed{E \to E + E \mid \textbf{id}}$$

$$\boxed{E \to E + E \mid \textbf{id}}$$

$$\boxed{\begin{array}{l} E \to E + T \\ T \to \textbf{id} \end{array}}$$

左结合文法

$$\boxed{E \to E + E \mid \textbf{id}}$$

$$E \to E + T$$

$$T \to \textbf{id}$$

左结合文法

$$E \to T + E$$

$$T \to \textbf{id}$$

右结合文法

$$\boxed{E \to E + E \mid \mathbf{id}}$$

$$E \to E + T$$

$$T \to \mathbf{id}$$

左结合文法

$$E \to T + E$$

$$T \to \mathbf{id}$$

右结合文法

使用**左（右）递归**实现**左（右）结合**

$$\boxed{E \to E + E \mid E * E \mid (E) \mid \mathbf{id}}$$

$$E \to E + E \mid E * E \mid (E) \mid \mathbf{id}$$

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

$$T \to T * F \mid F$$

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

括号最先，先乘后加文法

$$\boxed{E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \mathbf{id} \mid \mathbf{num}}$$

$$
\begin{aligned}
E &\to E + T \mid E - T \mid T \\
T &\to T * F \mid T/F \mid F \\
F &\to (E) \mid \mathbf{id} \mid \mathbf{num}
\end{aligned}
$$

无二义性的表达式文法

E : 表达式($expression$); T : 项($term$) F : 因子($factor$)

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid \textbf{id} \mid \textbf{num}$$

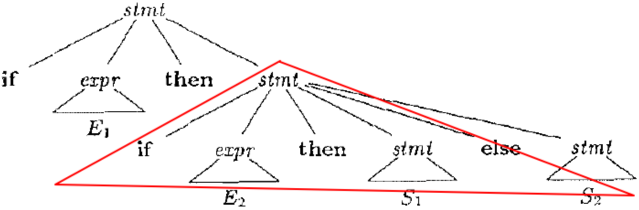
无二义性的表达式文法

E : 表达式($expression$); T : 项($term$) F : 因子($factor$)

将运算的“先后”顺序信息编码到语法树的“层次”结构中

$$
\begin{aligned}
stmt \;\rightarrow\;& \textbf{if } expr \textbf{ then } stmt \\
\mid\;& \textbf{if } expr \textbf{ then } stmt \; \boxed{\textbf{else } stmt} \\
\mid\;& \boxed{\textbf{other}}
\end{aligned}
$$

$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$

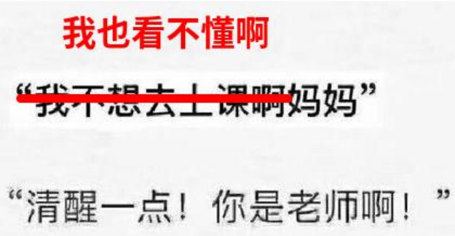


“每个else与**最近的尚未匹配的**then匹配”

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \boxed{\textbf{else } stmt} \\
 & | & \boxed{\textbf{other}}
\end{array}
$$

$$
\begin{array}{rcl}
stmt & \rightarrow & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

基本思想: **then** 与 **else** 之间的语句必须是**“已匹配的”**


我也看不懂啊
“~~我不想去上课啊~~妈妈”
“清醒一点！你是老师啊！”

我们要证明**两件**事情

我们要证明**两件**事情

$$L(G) = L(G')$$

我们要证明**两件**事情

$$L(G) = L(G')$$

G' 是无二义性的

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
& | & \textbf{if } expr \textbf{ then } stmt \; \boxed{\textbf{else } stmt} \\
& | & \boxed{\textbf{other}}
\end{array}
$$

$$
\begin{array}{rcl}
stmt & \rightarrow & matched\_stmt \\
& | & open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
& | & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
& | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

$$
\begin{aligned}
stmt \;&\rightarrow\; \textbf{if } expr \textbf{ then } stmt \\
&\mid\; \textbf{if } expr \textbf{ then } stmt \boxed{\textbf{else } stmt} \\
&\mid\; \boxed{\textbf{other}}
\end{aligned}
$$

$$
\begin{aligned}
stmt \;&\rightarrow\; matched\_stmt \\
&\mid\; open\_stmt \\
matched\_stmt \;&\rightarrow\; \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
&\mid\; \textbf{other} \\
open\_stmt \;&\rightarrow\; \textbf{if } expr \textbf{ then } stmt \\
&\mid\; \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{aligned}
$$

$$L(G) \subseteq L(G')$$

$$L(G') \subseteq L(G)$$

$$
\begin{aligned}
stmt &\rightarrow \textbf{if } expr \textbf{ then } stmt \\
&\mid \textbf{if } expr \textbf{ then } stmt \ \boxed{\textbf{else } stmt} \\
&\mid \boxed{\textbf{other}}
\end{aligned}
$$

$$
\begin{aligned}
stmt &\rightarrow matched\_stmt \\
&\mid open\_stmt \\
matched\_stmt &\rightarrow \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
&\mid \textbf{other} \\
open\_stmt &\rightarrow \textbf{if } expr \textbf{ then } stmt \\
&\mid \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{aligned}
$$

$L(G) \subseteq L(G')$

$L(G') \subseteq L(G)$

对推导步数作数学归纳

G' 是无二义性的

$$
\begin{array}{rcl}
stmt & \to & matched\_stmt \\
& | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
& | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
& | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

G' 是无二义性的

$$
\begin{array}{rcl}
stmt & \to & matched\_stmt \\
& | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
& | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
& | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

每个句子对应的**语法分析树**是唯一的

G' 是**无二义性**的

$$
\begin{array}{lcl}
stmt & \to & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

每个句子对应的**语法分析树**是唯一的

只需证明：每个非终结符的**"展开" 方式**是唯一的

G' 是无二义性的

$$
\begin{array}{lcl}
stmt & \to & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

每个句子对应的**语法分析树**是唯一的

只需证明: 每个非终结符的**"展开" 方式**是唯一的

$$L(matched\_stmt) \cap L(open\_stmt) = \emptyset$$

G' 是无二义性的

$$
\begin{array}{lcl}
stmt & \to & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

每个句子对应的**语法分析树**是唯一的

只需证明: 每个非终结符的**“展开” 方式**是唯一的

$$L(matched\_stmt) \cap L(open\_stmt) = \emptyset$$

$$L(matched\_stmt_1) \cap L(matched\_stmt_2) = \emptyset$$

G' 是无二义性的

$$
\begin{array}{lcl}
stmt & \to & matched\_stmt \\
& | & open\_stmt \\
matched\_stmt & \to & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } matched\_stmt \\
& | & \textbf{other} \\
open\_stmt & \to & \textbf{if } expr \textbf{ then } stmt \\
& | & \textbf{if } expr \textbf{ then } \boxed{matched\_stmt} \textbf{ else } open\_stmt
\end{array}
$$

每个句子对应的**语法分析树**是唯一的

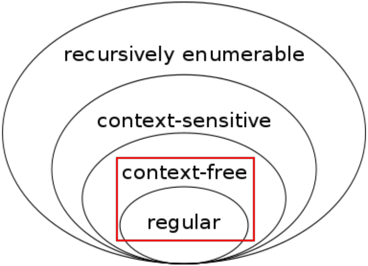
只需证明：每个非终结符的**"展开" 方式**是唯一的

$$L(matched\_stmt) \cap L(open\_stmt) = \emptyset$$

$$L(matched\_stmt_1) \cap L(matched\_stmt_2) = \emptyset$$

$$L(open\_stmt_1) \cap L(open\_stmt_2) = \emptyset$$

为什么不使用优雅、强大的**正则表达式**描述程序设计语言的语法?



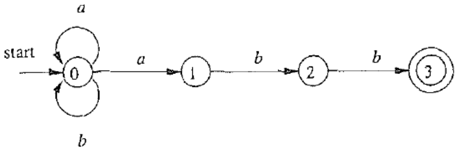
正则表达式的表达能力**严格弱于**上下文无关文法

每个正则表达式 r 对应的语言 $L(r)$ 都可以使用上下文无关文法来描述

$$r = (a|b)^{*}abb$$

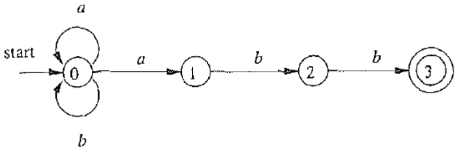
每个正则表达式 r 对应的语言 $L(r)$ 都可以使用上下文无关文法来描述

$$r = (a|b)^* abb$$



每个正则表达式 r 对应的语言 $L(r)$ 都可以使用上下文无关文法来描述

$$r = (a|b)^* abb$$



$$A_0 \to aA_0 \mid bA_0 \mid aA_1$$
$$A_1 \to bA_2$$
$$A_2 \to bA_3$$
$$A_3 \to \epsilon$$

此外，若 $\delta(A_i, \epsilon) = A_j$，则添加 $A_i \to A_j$

$$\boxed{\begin{aligned} S &\to aSb \\ S &\to \epsilon \end{aligned}}$$

$$L = \{a^n b^n \mid n \geq 0\}$$

该语言**无法**使用正则表达式来描述

Theorem

$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

Theorem

$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

Theorem

$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

Theorem

$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

Theorem

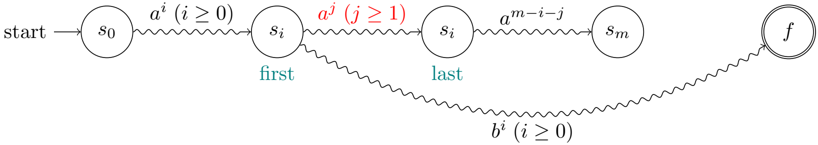
$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

$\boxed{\text{考虑输入 } a^m (m > k)}$



Theorem

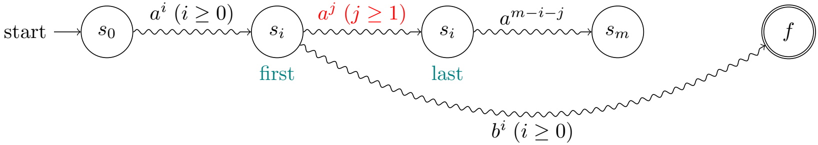
$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

$\boxed{考虑输入\ a^m (m > k)}$



$D(r)$ 也能接受 $a^{i+j}b^i$; **矛盾!**

$$L = \{a^n b^n \mid n \geq 0\}$$

Pumping Lemma for Regular Languages

$$L = \{a^n b^n \mid n \geq 0\}$$

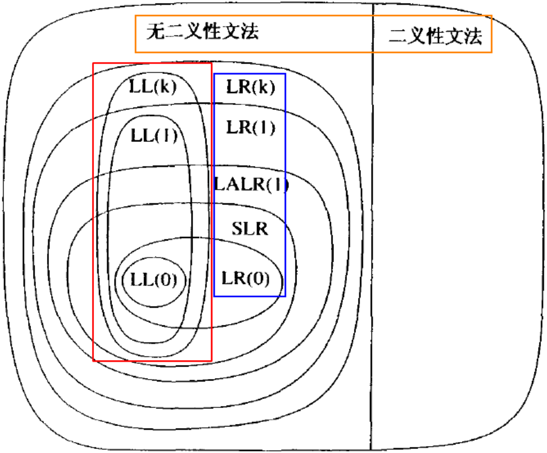
Pumping Lemma for Regular Languages

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

Pumping Lemma for Context-free Languages

只考虑**无二义性**的文法

这意味着, 每个句子对应唯一的一棵语法分析树



今日份主题: **$LL(1)$ 语法分析器**

自顶向下的、

递归下降的、

预测分析的、

适用于$LL(1)$ **文法**的、

$LL(1)$ 语法分析器

自顶向下构建语法分析树

根节点是文法的起始符号 S

叶节点是词法单元流 $w\$$

仅包含终结符号与特殊的**文件结束符** $\$$

自顶向下构建语法分析树

根节点是文法的起始符号 S

每个**中间节点**表示**对某个非终结符应用某个产生式进行推导**

(Q：选择哪个非终结符, 以及选择哪个产生式)

叶节点是词法单元流 $w\$$

仅包含终结符号与特殊的**文件结束符** $\$$

递归下降的实现框架

```
void A() {先不考虑这里是如何选择产生式的
1)    选择一个 A 产生式, A → X1X2···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 */;
      }
}
```

不需要任何参数

递归下降调用其它非终结符对应的递归函数

匹配当前词法单元

出现了不期望出现的词法单元

为每个**非终结符**写一个**递归函数**

内部按需调用其它非终结符对应的递归函数

$$\boxed{\begin{aligned} S &\to F \\ S &\to (S+F) \\ F &\to a \end{aligned}}$$

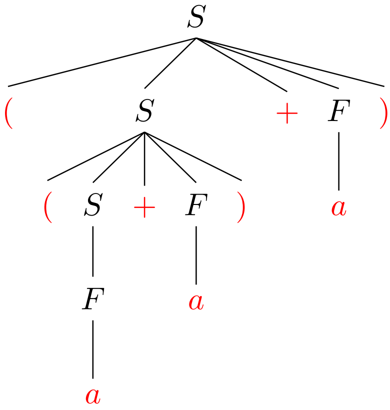
$$w = ((a+a)+a)$$

演示**递归下降**过程

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$



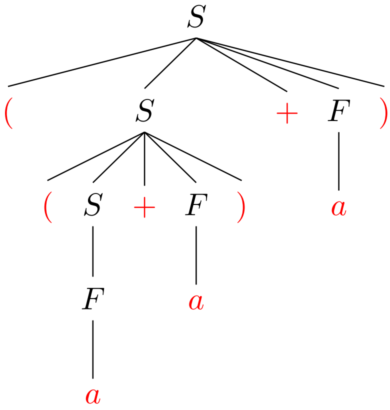
$$w = ((a + a) + a)$$

演示**递归下降**过程

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$



$$w = ((a + a) + a)$$

每次都选择语法分析树**最左边**的非终结符进行展开

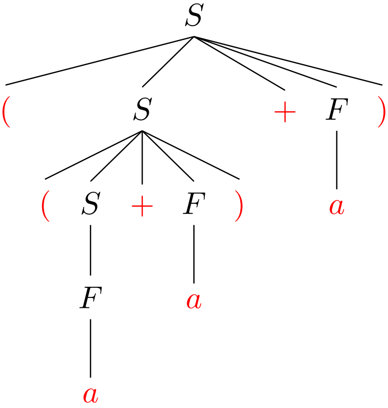
同样是展开非终结符 S,

为什么前两次选择了 $S \to (S+F)$, 而第三次选择了 $S \to F$?

$$S \to F$$

$$S \to (S+F)$$

$$F \to a$$

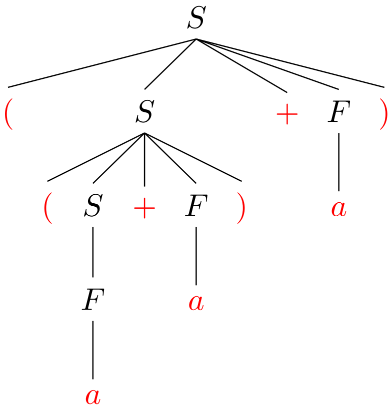


$$w = ((a+a)+a)$$

同样是展开非终结符 S,

为什么前两次选择了 $S \to (S + F)$, 而第三次选择了 $S \to F$?

$$\boxed{\begin{aligned} S &\to F \\ S &\to (S + F) \\ F &\to a \end{aligned}}$$



$w = ((a + a) + a)$

因为它们面对的**当前词法单元**不同

使用**预测分析表**确定产生式

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	+	\$
S	2		1		
F			3		

指明了每个**非终结符**在面对不同的**词法单元或文件结束符**时，

该选择哪个**产生式**（按编号进行索引）或者**报错**

Definition ($LL(1)$ 文法)

如果文法 G 的**预测分析表**是**无冲突**的，则 G 是 $LL(1)$ 文法。

无冲突: 每个单元格里只有一个生成式 (编号)

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	+	\$
S	2		1		
F			3		

对于当前选择的非终结符，

仅根据输入中当前的词法单元即可确定需要使用哪条产生式

递归下降的、预测分析实现方法

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	+	\$
S	2		1		
F			3		

```
procedure MATCH(t)
    if token = t then
        token ← NEXT-TOKEN()
    else
        ERROR(token, t)
```

```
procedure S()
    if token = '(' then
        MATCH('(')
        S()
        MATCH('+')
        F()
        MATCH(')')
    else if token = 'a' then
        F()
    else
        ERROR(token, {'(', 'a'})
```

递归下降的、预测分析实现方法

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	$+$	$
S	2		1		
F			3		

```
procedure MATCH(t)
    if token = t then
        token ← NEXT-TOKEN()
    else
        ERROR(token, t)
```

```
procedure F()
    if token = 'a' then
        MATCH('a')
    else
        ERROR(token, {'a'})
```

如何计算给定文法 G 的预测分析表?

$\textsc{First}(\alpha)$ 是可从 α 推导得到的句型的**首终结符号**的集合

Definition ($\textsc{First}(\alpha)$ 集合)

对于任意的 (产生式的右部) $\alpha \in (N \cup T)^*$:

$$\textsc{First}(\alpha) = \Big\{ t \in T \cup \{\epsilon\} \mid \alpha \xRightarrow{*} t\beta \lor \alpha \xRightarrow{*} \epsilon \Big\}.$$

如何计算给定文法 G 的预测分析表?

$\textsc{First}(\alpha)$ 是可从 α 推导得到的句型的**首终结符号**的集合

Definition ($\textsc{First}(\alpha)$ 集合)

对于任意的 (产生式的右部) $\alpha \in (N \cup T)^*$:

$$\textsc{First}(\alpha) = \Big\{ t \in T \cup \{\epsilon\} \mid \alpha \overset{*}{\Rightarrow} t\beta \vee \alpha \overset{*}{\Rightarrow} \epsilon \Big\}.$$

考虑非终结符 A 的所有产生式 $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_m$,

如果它们对应的 $\textsc{First}(\alpha_i)$ 集合互不相交,

则只需查看当前输入词法单元, 即可确定选择哪个产生式 (或报错)

如何计算给定文法 G 的预测分析表?

$\textsc{Follow}(A)$ 是可能在某些句型中**紧跟在 A 右边的终结符**的集合

Definition ($\textsc{Follow}(A)$ 集合)

对于任意的 (产生式的左部) 非终结符 $A \in N$:

$$\textsc{Follow}(A) = \Big\{ t \in T \cup \{\$\} \mid \exists s.\ S \xRightarrow{*} s \triangleq \beta A t \gamma \Big\}.$$

如何计算给定文法 G 的预测分析表?

$\textsc{Follow}(A)$ 是可能在某些句型中**紧跟在 A 右边的终结符**的集合

Definition ($\textsc{Follow}(A)$ 集合)

对于任意的 (产生式的左部) 非终结符 $A \in N$:

$$\textsc{Follow}(A) = \Big\{ t \in T \cup \{\$\} \mid \exists s.\ S \xRightarrow{*} s \triangleq \beta A t \gamma \Big\}.$$

考虑产生式 $A \to \alpha$,

如果从 α 可能推导出空串 ($\alpha \xRightarrow{*} \epsilon$),

则只有当当前词法单元 $t \in \textsc{Follow}(A)$, 才可以选择该产生式

先计算每个符号 X 的 $\textsc{First}(X)$ 集合

procedure $\textsc{first}(X)$

  if $X \in T$ **then** ▷ 规则 1: X 是终结符

    $\textsc{First}(X) = X$

  for $X \to Y_1 Y_2 \ldots Y_k$ **do** ▷ 规则 2: X 是非终结符

    $\textsc{First}(X) \leftarrow \textsc{First}(X) \cup \{\textsc{First}(Y_1) \setminus \{\epsilon\}\}$

    for $i \leftarrow 2$ **to** k **do**

      if $\epsilon \in L(Y_1 \ldots Y_{i-1})$ **then**

        $\textsc{First}(X) \leftarrow \textsc{First}(X) \cup \{\textsc{First}(Y_i) \setminus \{\epsilon\}\}$

    if $\epsilon \in L(Y_1 \ldots Y_k)$ **then** ▷ 规则 3: X 可推导出空串

      $\textsc{First}(X) \leftarrow \textsc{First}(X) \cup \{\epsilon\}$

不断应用上面的规则, 直到每个 $\textsc{First}(X)$ 都不再变化 (**闭包!!!**)

再计算每个符号串 α 的 $\textsc{First}(\alpha)$ 集合

$$\alpha = X\beta$$

$$\textsc{First}(\alpha) = \begin{cases} \textsc{First}(X) & \epsilon \notin L(X) \\ (\textsc{First}(X) \setminus \{\epsilon\}) \cup \textsc{First}(\beta) & \epsilon \in L(X) \end{cases}$$

最后, 如果 $\epsilon \in L(\alpha)$, 则将 ϵ 加入 $\textsc{First}(\alpha)$。

$$
\begin{array}{|c|}
\hline
(1)\ X \to Y \\
(2)\ X \to a \\
(3)\ Y \to \epsilon \\
(4)\ Y \to c \\
(5)\ Z \to d \\
(6)\ Z \to XYZ \\
\hline
\end{array}
$$

$$
\boxed{
\begin{array}{l}
(1)\ X \to Y \\
(2)\ X \to a \\
(3)\ Y \to \epsilon \\
(4)\ Y \to c \\
(5)\ Z \to d \\
(6)\ Z \to XYZ
\end{array}
}
$$

$$
\begin{aligned}
\text{FIRST}(X) &= \{a, c, \epsilon\} \\
\text{FIRST}(Y) &= \{c, \epsilon\} \\
\text{FIRST}(Z) &= \{a, c, d\}
\end{aligned}
$$

$$\text{FIRST}(XYZ) = \{a, c, d\}$$

为每个非终结符 X 计算 $\text{FOLLOW}(X)$ 集合

procedure $\text{FOLLOW}(X)$

  for X 是开始符号 **do** ▷ 规则 1: X 是开始符号

    $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \{\$\}$

  for $A \to \alpha X \beta$ **do** ▷ 规则 2: X 是某产生式右部中间的一个符号

    $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup (\text{FIRST}(\beta) \setminus \{\epsilon\})$

    if $\epsilon \in \text{FIRST}(\beta)$ **then**

      $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \text{FOLLOW}(A)$

  for $A \to \alpha X$ **do** ▷ 规则 3: X 是某产生式右部的最后一个符号

    $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \text{FOLLOW}(A)$

不断应用上面的规则, 直到每个 $\text{FOLLOW}(X)$ 都不再变化 (**闭包!!!**)

$$\boxed{\begin{array}{l} (1)\ X \to Y \\ (2)\ X \to a \\ (3)\ Y \to \epsilon \\ (4)\ Y \to c \\ (5)\ Z \to d \\ (6)\ Z \to XYZ \end{array}}$$

$$
\boxed{
\begin{array}{l}
(1)\ X \to Y \\[1em]
(2)\ X \to a \\[1em]
(3)\ Y \to \epsilon \\[1em]
(4)\ Y \to c \\[1em]
(5)\ Z \to d \\[1em]
(6)\ Z \to XYZ
\end{array}
}
$$

$$
\begin{aligned}
\textsc{Follow}(X) &= \{a, c, d, \$\} \\
\textsc{Follow}(Y) &= \{a, c, d, \$\} \\
\textsc{Follow}(Z) &= \emptyset
\end{aligned}
$$

如何根据FIRST 与 FOLLOW 集合计算给定文法 G 的预测分析表?

按照以下规则, 在表格 $[A, t]$ 中填入生成式 $A \to \alpha$ (编号):

$$t \in \textsc{First}(\alpha) \tag{1}$$

$$\alpha \xRightarrow{*} \epsilon \land t \in \textsc{Follow}(A) \tag{2}$$

如何根据First 与 Follow 集合计算给定文法 G 的预测分析表?

按照以下规则, 在表格 $[A, t]$ 中填入生成式 $A \to \alpha$ (编号):

$$t \in \textsc{First}(\alpha) \tag{1}$$

$$\alpha \stackrel{*}{\Longrightarrow} \epsilon \wedge t \in \textsc{Follow}(A) \tag{2}$$

Definition ($LL(1)$ 文法)

如果文法 G 的**预测分析表**是**无冲突**的, 则 G 是 $LL(1)$ 文法。

$$(1)\ X \to Y$$

$$(2)\ X \to a$$

$$(3)\ Y \to \epsilon$$

$$(4)\ Y \to c$$

$$(5)\ Z \to d$$

$$(6)\ Z \to XYZ$$

$$\textsc{First}(X) = \{a, c, \epsilon\}$$
$$\textsc{First}(Y) = \{c, \epsilon\}$$
$$\textsc{First}(Z) = \{a, c, d\}$$

$$\textsc{First}(XYZ) = \{a, c, d\}$$

$$\textsc{Follow}(X) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Y) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Z) = \emptyset$$

	a	c	d	$\$$
X	1, 2	1	1	1
Y	3	3, 4	3	3
Z	6	6	5, 6	

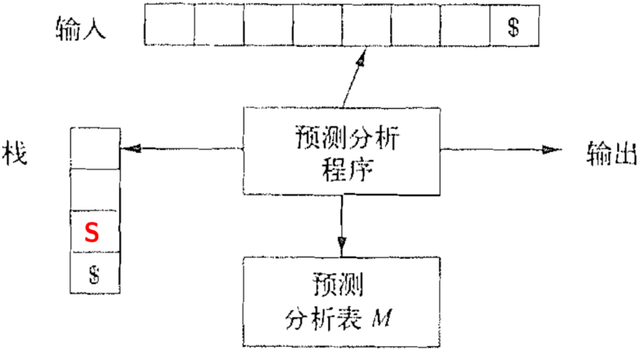
$LL(1)$ 语法分析器

L : 从左向右 (left-to-right) 扫描输入

L : 构建最左 (leftmost) 推导

1 : 只需向前看一个输入符号便可确定使用哪条产生式

非递归的预测分析算法



非递归的预测分析算法

```
设置 ip 使它指向 w 的第一个符号，其中 ip 是输入指针;
令 [X]= 栈顶符号;
while ( X ≠ $ ) { /* 栈非空 */
    if ( X 等于 ip 所指向的符号 a ) 执行栈的弹出操作，将 ip 向前移动一个位置;
    else if ( X 是一个终结符号 ) error();
    else if ( M[X, a] 是一个报错条目 ) error();
    else if ( M[X, a] = X → Y1Y2 ··· Yk ) {
        输出产生式 X → Y1Y2 ··· Yk;
        [弹出栈顶符号];
        将 Yk, Yk-1, ..., Y1 压入栈中，其中 Y1 位于栈顶。
    }
    令 [X] = 栈顶符号;
}
```

不是 $LL(1)$ 文法怎么办?

改造它

消除左递归

提取左公因子

$$\boxed{\begin{aligned} &E \to E + T \mid E - T \mid T \\ &T \to T * F \mid T/F \mid F \\ &F \to (E) \mid \textbf{id} \mid \textbf{num} \end{aligned}}$$

E 在**不消耗任何词法单元**的情况下, 直接递归调用 E, 造成**死循环**

$$
\boxed{
\begin{aligned}
E &\to E + T \mid E - T \mid T \\
T &\to T * F \mid T/F \mid F \\
F &\to (E) \mid \mathbf{id} \mid \mathbf{num}
\end{aligned}
}
$$

E 在**不消耗任何词法单元**的情况下, 直接递归调用 E, 造成**死循环**

$$\boxed{\begin{aligned} E &\to E + T \mid E - T \mid T \\ T &\to T * F \mid T/F \mid F \\ F &\to (E) \mid \textbf{id} \mid \textbf{num} \end{aligned}}$$

$$\text{First}(E + T) \cap \text{First}(T) \neq \emptyset$$

不是 $LL(1)$ 文法

消除左递归

$$\boxed{E \to E + T \mid T}$$

消除左递归

$$\boxed{E \to E + T \mid T}$$

$$\boxed{\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \epsilon \end{aligned}}$$

将左递归转为**右递归**

消除左递归

$$\boxed{E \to E + T \mid T}$$

$$\boxed{\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \epsilon \end{aligned}}$$

将左递归转为**右递归**

(注: 右递归对应右结合; 需要在后续阶段进行额外处理)

$$\boxed{A \to A\alpha_1 \mid A\alpha_2 \mid \ldots A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \beta_n}$$

其中, β_i 都不以 A 开头

$$\boxed{\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned}}$$

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

$$T \to T * F \mid F$$

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

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

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

非直接左递归

$$\boxed{\begin{aligned} S &\to Aa \mid b \\ A &\to Ac \mid Sb \mid \epsilon \end{aligned}}$$

$$S \implies Aa \implies Sda$$

非直接左递归

$$S \to Aa \mid b$$

$$A \to Ac \mid Sb \mid \epsilon$$

$$S \implies Aa \implies Sda$$

1) 按照某个顺序将非终结符号排序为 $A_1, A_2, \ldots, A_n$.
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 $A_i \to A_j\gamma$ 的产生式替换为产生式组 $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$，其中 $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ 是所有的 A_j 产生式
5) }
6) 消除 A_i 产生式之间的立即左递归
7) }

图 4-11 消除文法中的左递归的算法

$$\boxed{A_k \to A_l\alpha \implies l > k}$$

$$\boxed{\begin{aligned} S &\to Aa \mid b \\ A &\to Ac \mid Sb \mid \epsilon \end{aligned}}$$

$$A \to Ac \mid Aad \mid bd \mid \epsilon$$

$$\boxed{\begin{aligned} S &\to Aa \mid b \\ A &\to bdA' \mid A' \\ A' &\to cA' \mid adA' \mid \epsilon \end{aligned}}$$

$$\boxed{A_k \to A_l\alpha \implies l > k}$$

$$\boxed{\begin{aligned} &E \to TE' \\ &E' \to +\,TE' \mid \epsilon \\ &T \to FT' \\ &T' \to *\,FT' \mid \epsilon \\ &F \to (E) \mid \textbf{id} \mid \textbf{num} \end{aligned}}$$

$$\begin{aligned} \text{First}(F) &= \{(, \textbf{id}\} \\ \text{First}(T) &= \{(, \textbf{id}\} \\ \text{First}(E) &= \{(, \textbf{id}\} \\ \text{First}(E') &= \{+, \epsilon\} \\ \text{First}(T') &= \{*, \epsilon\} \end{aligned}$$

$$\begin{aligned} \text{Follow}(E) &= \text{Follow}(E') = \{), \$\} \\ \text{Follow}(T) &= \text{Follow}(T') = \{+,), \$\} \\ \text{Follow}(F) &= \{+, *,), \$\} \end{aligned}$$

$$E \to TE'$$

$$E' \to +TE' \mid \epsilon$$

$$T \to FT'$$

$$T' \to *FT' \mid \epsilon$$

$$F \to (E) \mid \textbf{id} \mid \textbf{num}$$

非终结符号	输入符号					
	id	+	*	(	)	$
E	$\boxed{E \to TE'}$			$\boxed{E \to TE'}$		
E'		$E' \to +TE'$			$\boxed{E' \to \epsilon}$	$\boxed{E' \to \epsilon}$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \text{id}$			$F \to (E)$		

$$\textsc{First}(F) = \{(, \textbf{id}\}$$

$$\textsc{First}(T) = \{(, \textbf{id}\}$$

$$\textsc{First}(E) = \{(, \textbf{id}\}$$

$$\textsc{First}(E') = \{+, \epsilon\}$$

$$\textsc{First}(T') = \{*, \epsilon\}$$

$$\textsc{Follow}(E) = \textsc{Follow}(E') = \{), \$\}$$

$$\textsc{Follow}(T) = \textsc{Follow}(T') = \{+,), \$\}$$

$$\textsc{Follow}(F) = \{+, *,), \$\}$$

句型

已匹配	栈	输入	动作
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	输出 $E \to TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	输出 $T \to FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	输出 $F \to \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	匹配 $\mathbf{id}$
$\mathbf{id}$	$E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	输出 $T' \to \epsilon$
$\mathbf{id}$	$+ TE'\$$	$+ \mathbf{id} * \mathbf{id}\$$	输出 $E' \to + TE'$
$\mathbf{id} +$	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	匹配 $+$
$\mathbf{id} +$	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	输出 $T \to FT'$
$\mathbf{id} +$	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	输出 $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$T'E'\$$	$* \mathbf{id}\$$	匹配 $\mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$* FT'E'\$$	$* \mathbf{id}\$$	输出 $T' \to * FT'$
$\mathbf{id} + \mathbf{id} *$	$FT'E'\$$	$\mathbf{id}\$$	匹配 $*$
$\mathbf{id} + \mathbf{id} *$	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	输出 $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$T'E'\$$	$\$$	匹配 $\mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$E'\$$	$\$$	输出 $T' \to \epsilon$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$\$$	$\$$	输出 $E' \to \epsilon$

图 4-21 对输入 $\mathbf{id} + \mathbf{id} * \mathbf{id}$ 进行预测分析时执行的步骤

$$S \rightarrow i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a$$
$$E \rightarrow b$$

提取左公因子

$$S \rightarrow i\ E\ t\ S\ S' \mid a$$
$$S' \rightarrow e\ S \mid \epsilon$$
$$E \rightarrow b$$

$$S \rightarrow i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a$$

$$E \rightarrow b$$

非终结符号	输入符号					
	a	b	e	i	t	$\$$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$\boxed{S' \rightarrow \epsilon \quad S' \rightarrow eS}$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

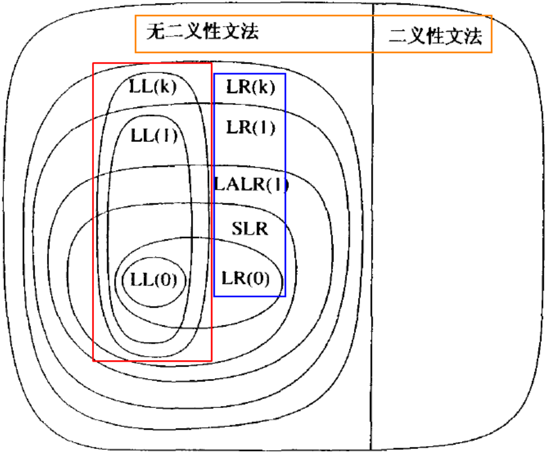
解决二义性: 选择 $S' \rightarrow eS$, 将 **else** 与前面最近的 **then** 关联起来

语法分析阶段的主题之三：**错误恢复**

报错、**恢复**、继续分析

只考虑**无二义性**的文法

这意味着，每个句子对应唯一的一棵语法分析树



今日份主题: $LR(1)$ ($LR(0)$) **语法分析器**

自顶向下的、

不断规约的、

基于句柄查找自动机的、

适用于 $LR(*)$ **文法**的、

$LR(*)$ 语法分析器

自底向上构建语法分析树

根节点是文法的起始符号 S

叶节点是词法单元流 $w\$$

仅包含终结符号与特殊的**文件结束符** $\$$

自底向上构建语法分析树

根节点是文法的起始符号 S

每个**中间非终结符节点**表示**使用它的某条产生式进行归约**

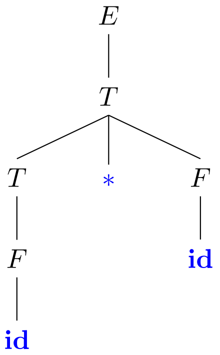
叶节点是词法单元流 $w\$$

仅包含终结符号与特殊的**文件结束符** $\$$

自顶向下的“推导”与自底向上的“归约”

$$E \underset{rm}{\Longrightarrow} T \underset{rm}{\Longrightarrow} T * F \underset{rm}{\Longrightarrow} T * \mathbf{id} \underset{rm}{\Longrightarrow} F * \mathbf{id} \underset{rm}{\Longrightarrow} \mathbf{id} * \mathbf{id}$$

$$\boxed{\begin{array}{l} (1)\ E \to E + T \\ (2)\ E \to T \\ (3)\ T \to T * F \\ (4)\ T \to F \\ (5)\ F \to (E) \\ (6)\ F \to \mathbf{id} \end{array}}$$



$w = \mathbf{id} * \mathbf{id}$

$$E \Longleftarrow T \Longleftarrow T * F \Longleftarrow T * \mathbf{id} \Longleftarrow F * \mathbf{id} \Longleftarrow \mathbf{id} * \mathbf{id}$$

“推导” $(A \to \alpha)$ 与 “归约” $(A \leftarrow \alpha)$

$$S \triangleq \gamma_0 \implies \ldots \gamma_{i-1} \implies \gamma_i \implies \gamma_{r+1} \implies \cdots \implies r_n = w$$

$$S \triangleq \gamma_0 \impliedby \ldots \gamma_{i-1} \impliedby \gamma_i \impliedby \gamma_{r+1} \impliedby \cdots \impliedby r_n = w$$

自底向上语法分析器为输入构造**反向推导**

LR 语法分析器

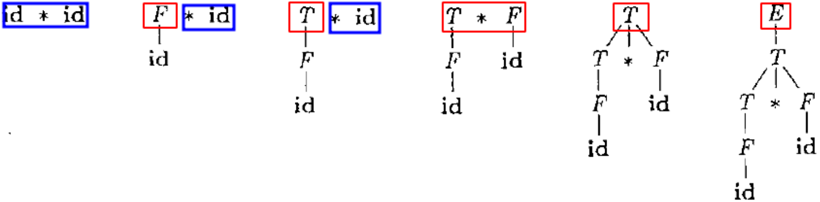
L : **从左向右** (Left-to-right) 扫描输入

R : 构建**反向** (Reverse) **最右推导**

“反向最右推导” 与 “从左到右扫描” 相一致

LR 语法分析器的状态

在任意时刻, 语法分析树的**上边缘**与**剩余的输入**构成当前句型



$$E \Longleftarrow T \Longleftarrow T * F \Longleftarrow T * \mathbf{id} \Longleftarrow F * \mathbf{id} \Longleftarrow \mathbf{id} * \mathbf{id}$$

LR 语法分析器使用**栈**存储语法分析树的**上边缘**

它包含了语法分析器目前所知的所有信息

板书演示 “**栈**” 上操作

$(1)\ E \rightarrow E + T$

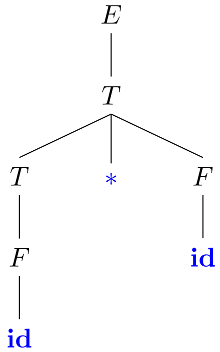
$(2)\ E \rightarrow T$

$(3)\ T \rightarrow T * F$

$(4)\ T \rightarrow F$

$(5)\ F \rightarrow (E)$

$(6)\ F \rightarrow \mathbf{id}$



$w = \mathbf{id} * \mathbf{id}$

两大操作: **移入输入符号** 与 **按产生式归约**

直到栈中仅剩开始符号 S, 且输入已结束, 则成功停止

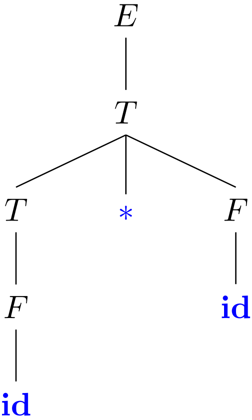
基于栈的 LR 语法分析器

Q_1 : 何时归约? (何时移入?)

Q_2 : 按哪条产生式进行规约?

基于栈的 LR 语法分析器

$$\boxed{\begin{array}{l}(1)\ E \to E + T \\ (2)\ E \to T \\ (3)\ T \to T * F \\ (4)\ T \to F \\ (5)\ F \to (E) \\ (6)\ F \to \mathbf{id}\end{array}}$$



为什么第二个 F 以 $T * F$ 整体被归约为 T?

这与**栈**的当前状态 “$T * F$” 相关

$LR(0)$ **分析表**指导 $LR(0)$ 语法分析器

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

在**当前状态（编号）**下，面对**当前文法符号**时，该采取什么**动作**

ACTION 表指明动作，GOTO 表仅用于归约时的状态转换

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

sn	移入输入符号, 并进入**状态** n
rk	使用k **号产生式**进行归约
gn	转换到**状态** n
acc	成功接受, 结束
空白	错误

Definition ($LR(0)$ 文法)

如果文法 G 的 $LR(0)$ **分析表**是**无冲突**的, 则 G 是 $LR(0)$ 文法。

无冲突: ACTION 表中每个单元格最多只有一种动作

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

两类可能的冲突: "移入/归约" 冲突、"归约/归约" 冲突

再次板书演示 “栈” 上操作: 移入与归约

$(1)\ E \to E + T$

$(2)\ E \to T$

$(3)\ T \to T * F$

$(4)\ T \to F$

$(5)\ F \to (E)$

$(6)\ F \to \mathbf{id}$

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

$$w = \mathbf{id} * \mathbf{id}\$$$

栈中存储语法分析器的**状态 (编号)**, “编码” 了语法分析树的上边缘

```
procedure LR()
    Push(S, s_0)                                          ▷ 或 Push(S, $_{s_0})
    token ← NEXT-TOKEN()

    while (1) do
        s ← Top(S)

        if ACTION[s, token] = s_i then                    ▷ 移入
            Push(S, i)                                    ▷ 或 Push(S, token_{s_i})
            token ← NEXT-TOKEN()
        else if ACTION[s, token] = r_j then               ▷ 规约; j : A → α
            |α| 次 Pop(S)
            s ← Top(S)
            Push(S, GOTO[s, A])   ▷ 转换状态; 或 Push(S, A_{GOTO[s,A]})
        else if ACTION[s, token] = acc then               ▷ 接受
            break
        else
            ERROR(...)
```

行号	栈	符号	输入	动作
(1)	0	\$	id * id \$	移入到 5
(2)	0 5	\$ id	* id \$	按照 $F \to \mathbf{id}$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \to F$ 归约
(4)	0 2	\$ T	* id \$	移入到 7
(5)	0 2 7	\$ T *	id \$	移入到 5
(6)	0 2 7 5	\$ T * id	\$	按照 $F \to \mathbf{id}$ 归约
(7)	0 2 7 10	\$ $T * F$	\$	按照 $T \to T * F$ 归约
(8)	0 2	\$ T	\$	按照 $E \to T$ 归约
(9)	0 1	\$ E	\$	接受

句型

$w = \mathbf{id} * \mathbf{id}\$$ 的分析过程

如何构造 $LR(0)$ 分析表?

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

在**当前状态 (编号)**下, 面对**当前文法符号**时, 该采取什么**动作**

状态是什么？如何跟踪状态？

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

状态是语法分析树的上边缘，存储在栈中

可以用**自动机**跟踪状态变化（自动机中的路径 ⇔ 栈中符号/状态编号）

何时归约？使用哪条产生式进行归约？

状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

必要条件： 当前状态中，已观察到某个产生式的完整右部

对于 $LR(0)$ 文法，这是当前**唯一**的选择

何时归约？使用哪条产生式进行归约？

Definition (句柄 (Handle))

在输入串的 (唯一) 反向最右推导中，**如果**下一步是逆用产生式 $A \to \alpha$ 将 α 规约为 A，则称 α 是当前句型的**句柄**。

最右句型	句柄	归约用的产生式
$\boxed{\mathbf{id}_1} * \mathbf{id}_2$	$\mathbf{id}_1$	$F \to \mathbf{id}$
$\boxed{F} * \mathbf{id}_2$	F	$T \to F$
$T * \boxed{\mathbf{id}_2}$	$\mathbf{id}_2$	$F \to \mathbf{id}$
$\boxed{T * F}$	$T * F$	$T \to T * F$
$\boxed{T}$	T	$E \to T$

LR 语法分析器的关键就是高效**寻找每个归约步骤所使用的句柄**。

句柄可能在哪里?

Theorem

存在一种 LR 语法分析方法, 保证**句柄总是出现在栈顶**。

句柄可能在哪里?

Theorem

存在一种 LR 语法分析方法，保证**句柄总是出现在栈顶**。

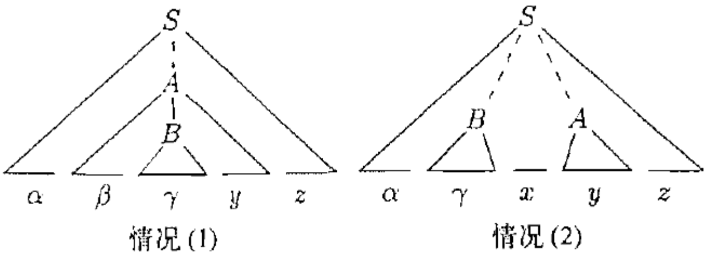


图 4-29 一个最右推导中两个连续步骤的两种情况

$$S \underset{rm}{\overset{*}{\Longrightarrow}} \alpha A z \underset{rm}{\overset{*}{\Longrightarrow}} \alpha\beta B y z \underset{rm}{\overset{*}{\Longrightarrow}} \alpha\beta\gamma y z \quad S \underset{rm}{\overset{*}{\Longrightarrow}} \alpha B x A z \underset{rm}{\overset{*}{\Longrightarrow}} \alpha B x y z \underset{rm}{\overset{*}{\Longrightarrow}} \alpha\gamma x y$$

可以用**自动机**跟踪状态变化

(自动机中的路径 $\Leftrightarrow$ 栈中符号/状态编号)

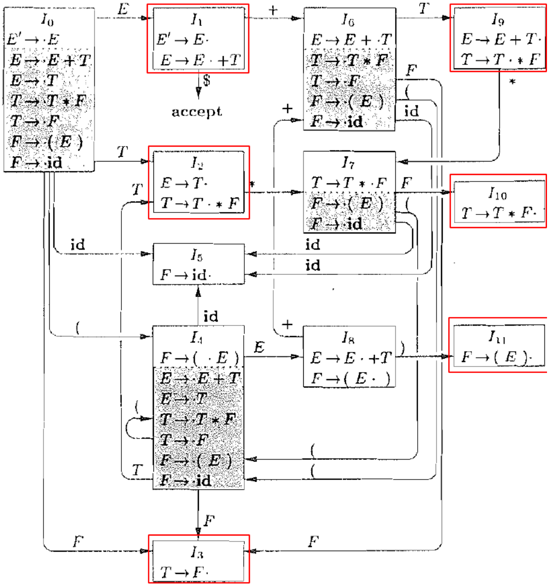
Theorem

存在一种 LR 语法分析方法，保证**句柄总是出现在栈顶**。

在自动机的当前状态识别可能的句柄 (观察到的完整右部)

(自动机的当前状态 $\Leftrightarrow$ 栈顶)

$LR(0)$ 句柄识别有穷状态自动机 (Handle-Finding Automaton)

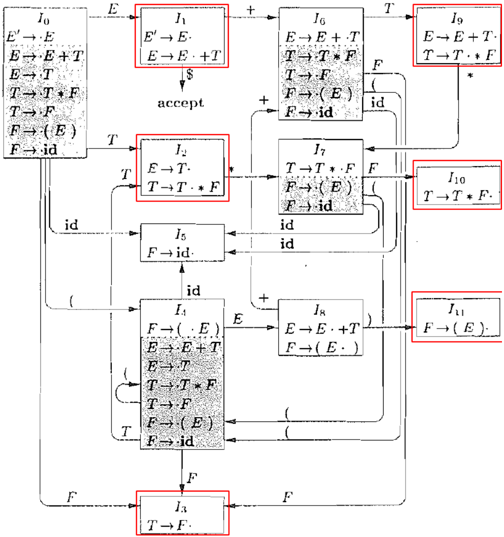


$LR(0)$ **句柄识别自动机**

为给定的**文法** G构造相应的句柄识别自动机

该自动机用于识别该**文法** G **所允许的所有可能的句柄**

$LR(0)$ 句柄识别自动机



状态是什么？状态之间如何转移？

状态刻画了"当前观察到的**针对所有产生式的右部的前缀**"

Definition ($LR(0)$ 项 (Item))

文法 G 的一个 $LR(0)$ **项**是 G 的某个产生式加上一个位于体部的**点**。

$$A \to XYZ$$

$$A \to \cdot XYZ$$

$$A \to X \cdot YZ$$

$$A \to XY \cdot Z$$

$$A \to XYZ \cdot$$

(产生式 $A\epsilon$ 只有一个项 $A \to \cdot$)

项指明了语法分析器已经观察到了某个产生式的哪些部分

点指示了**栈顶**, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition (项集)

项集就是若干**项**构成的集合。

因此，句柄识别自动机的一个**状态**可以表示为一个**项集**

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition (项集)

项集就是若干**项**构成的集合。

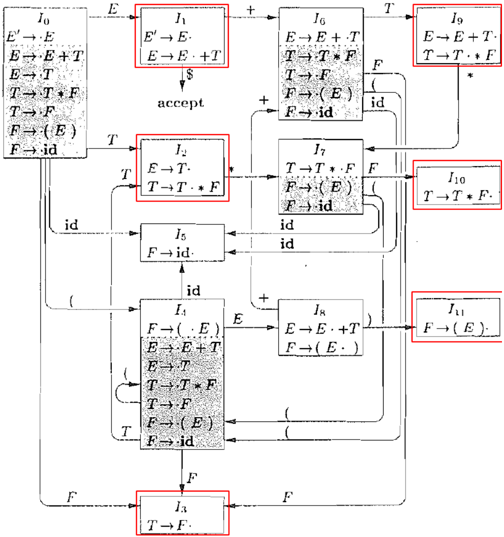
因此，句柄识别自动机的一个**状态**可以表示为一个**项集**

Definition (项集族)

项集族就是若干**项集**构成的集合。

因此，句柄识别自动机的**状态集**可以表示为一个**项集族**

$LR(0)$ 句柄识别自动机



项、项集、项集族

Definition (增广文法 (Augmented Grammar))

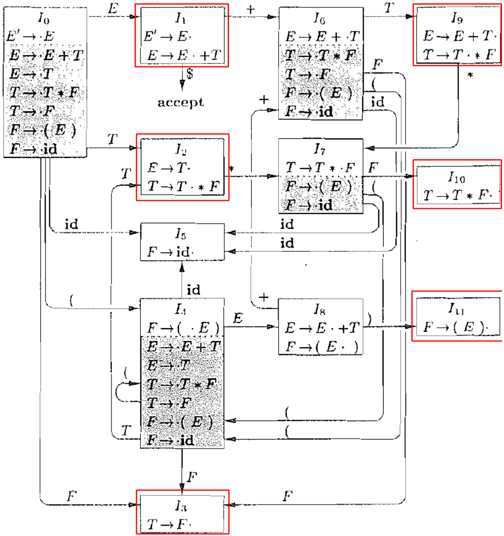
文法 G 的**增广文法**是在 G 中加入产生式 $S' \to S$ 得到的文法。

目的: 告诉语法分析器何时停止分析并接受输入符号串

当语法分析器**面对 \$**且**要使用 $S' \to S$ 进行归约**时, 输入符号串被接受

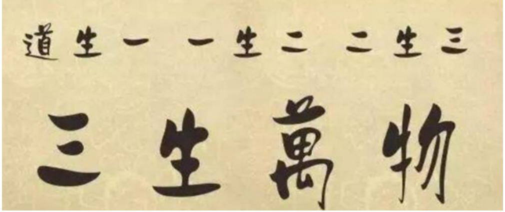
注: 此 "接受" (输入串) 非彼 "接受" (句柄识别自动机)

$LR(0)$ 句柄识别自动机



注：此“接受”（输入串）非彼“接受”（句柄识别自动机）

$LR(0)$ 句柄识别自动机



初始状态是什么?

状态之间如何转移?

点指示了**栈顶**, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号

$$(0)\ E' \to E$$

$$(1)\ E \to E + T$$

$$(2)\ E \to T$$

$$(3)\ T \to T * F$$

$$(4)\ T \to F$$

$$(5)\ F \to (E)$$

$$(6)\ F \to \mathbf{id}$$

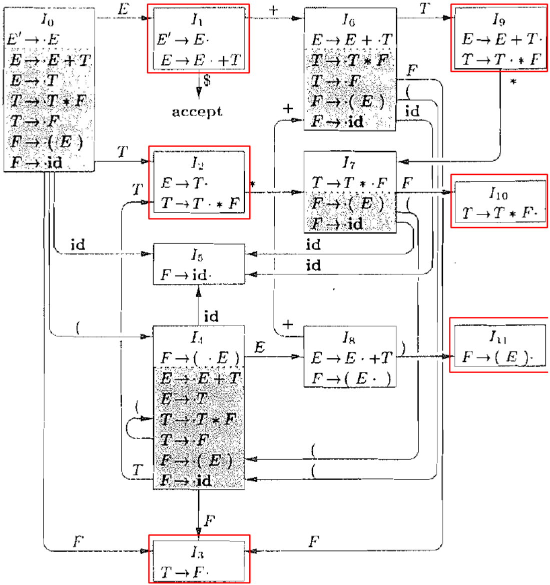
I_0

$$\boxed{E' \to \cdot E}$$

$$\boxed{\begin{array}{l} E \to \cdot E + T \\ E \to \cdot T \\ T \to \cdot T * F \\ T \to \cdot F \\ F \to \cdot (E) \\ F \to \cdot \mathbf{id} \end{array}}$$

$$\boxed{\text{CLOSURE}(\{[E' \to \cdot E]\})}$$

板书演示 $LR(0)$ 句柄识别自动机的构造过程



```
SetOfItems CLOSURE(I) {
    J = I;
    repeat
        for ( J中的每个项 A → α·Bβ)
            for (G 的每个产生式 B → γ )
                if (项 B → ·γ 不在 J中 )
                    将 B → ·γ 加入 J中;
    until 在某一轮中没有新的项被加入到J 中;
    return J;
}
```

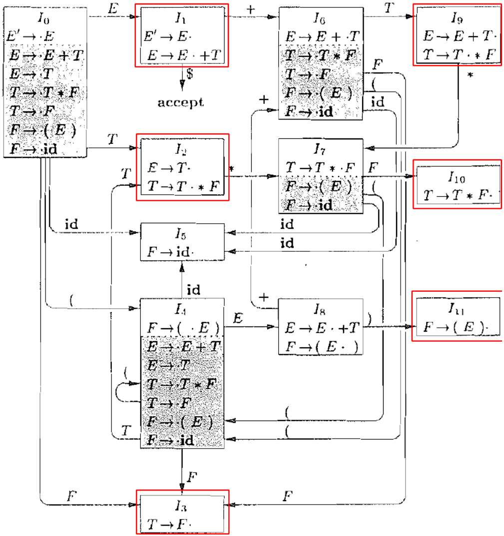
$$\text{GOTO}(I, X) = \text{CLOSURE}\left(\left\{[A \to \alpha X \cdot \beta] \,\middle|\, [A \to \alpha \cdot X\beta] \in I\right\}\right)$$

$$(X \in N \cup T \cup \{\$\})$$

```
void items(G') {
    C = {CLOSURE({[S' → ·S]})};          初始状态
    repeat
        for ( C 中的每个项集 I )
            for ( 每个文法符号 X )
                if ( GOTO(I, X) 非空且不在 C 中 )
  下一个状态        将 GOTO(I, X) 加入 C 中;
    until 在某一轮中没有新的项集被加入到 C 中;
}
```

图 4-33　规范 LR(0) 项集族的计算

$LR(0)$ 句柄识别自动机



Q : 为什么是个有穷状态自动机?


Thank
You!

Your opinon Matters

Office 926

hfwei@nju.edu.cn