语法分析
(2. 语法分析器生成器 ANTLR4)

魏恒峰

hfwei@nju.edu.cn

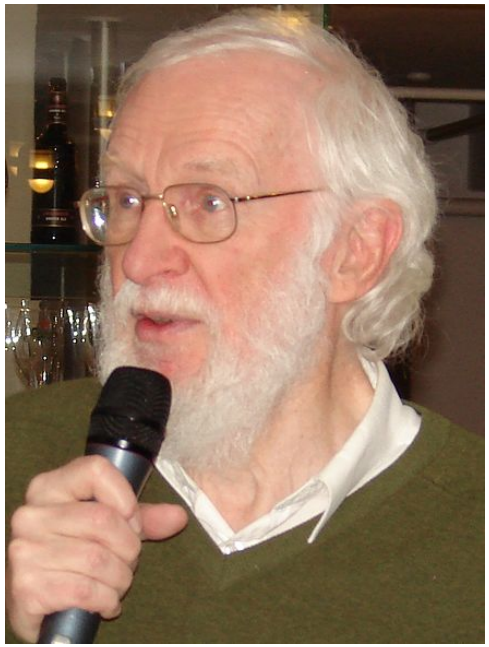
2021 年 11 月 19 日

ANTLR4 语法分析器

CymbolCFG.g4

[Extended] Backus–Naur form ([E]BNF)

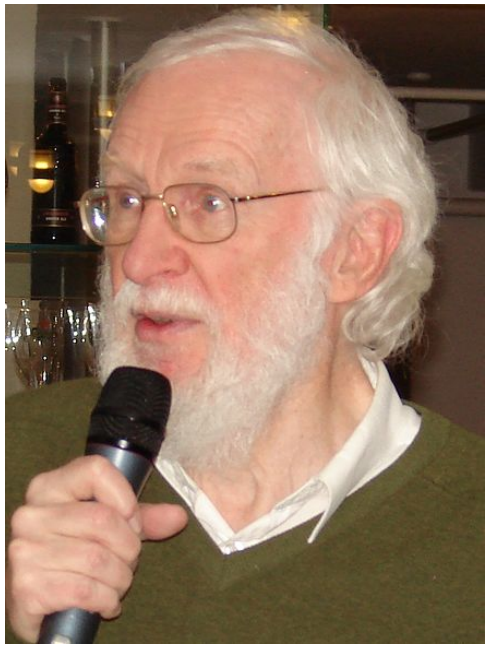
John Backus
(1924 ∼ 2007)



Peter Naur
(1928 ∼ 2016)

[Extended] Backus–Naur form ([E]BNF)

John Backus
(1924 ∼ 2007)



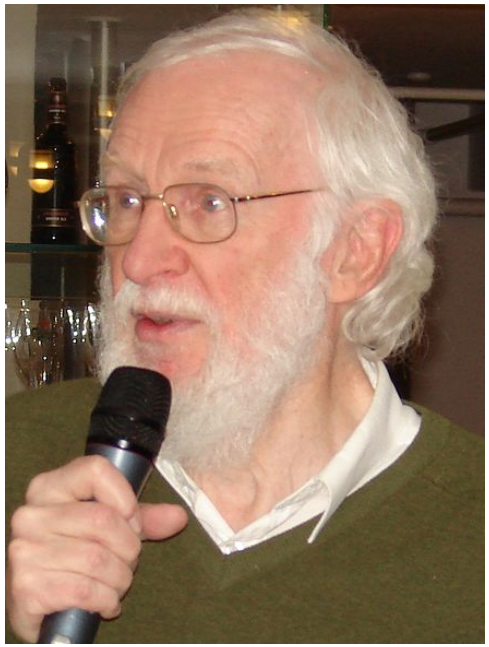
Peter Naur
(1928 ∼ 2016)

Niklaus Wirth (1934 ∼)

[Extended] Backus–Naur form ([E]BNF)

John Backus
(1924 ∼ 2007)

1977 (FORTRAN)



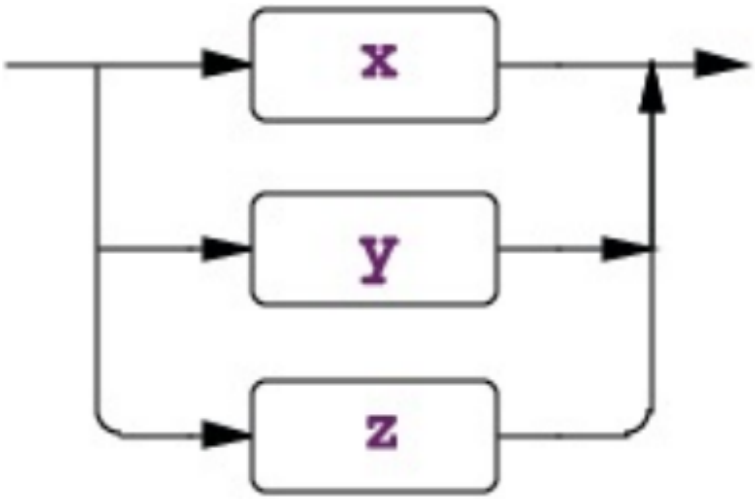
Peter Naur
(1928 ∼ 2016)

2005 (ALGOL60)

Niklaus Wirth (1934 ∼)

1984 (PLs; PASCAL)

Extended Backus–Naur form (EBNF)



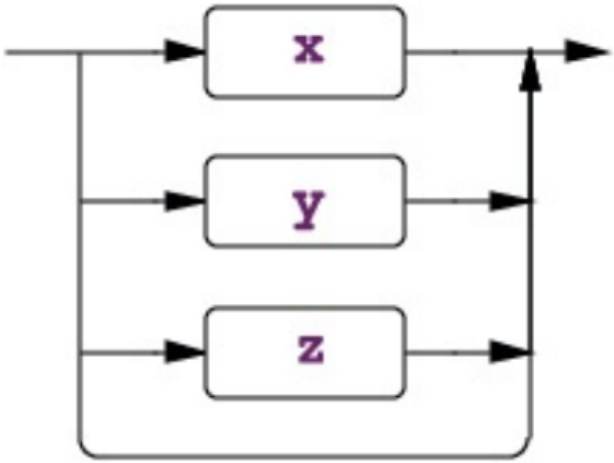
(x|y|z)

Match any alternative within the subrule exactly once. Here's an example:

```
returnType : (type | 'void') ;
```

Choice

Extended Backus–Naur form (EBNF)



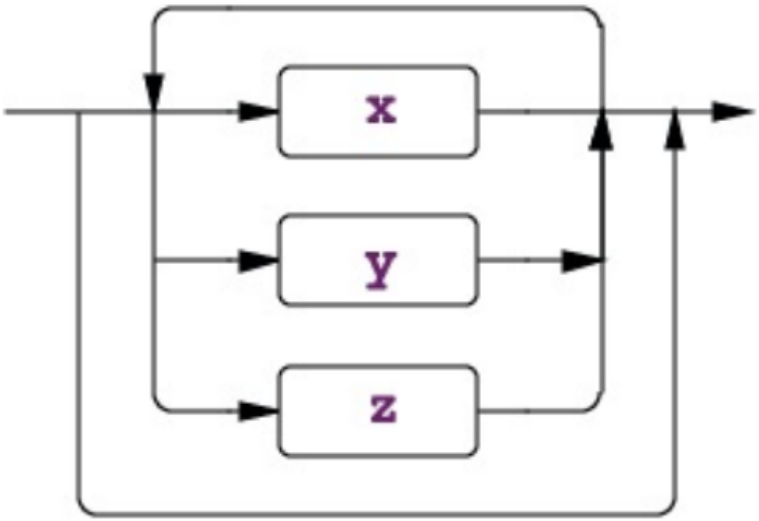
(x|y|z)?

Match nothing or any alternative within the subrule. Here's an example:

```
classDeclaration
    :   'class' ID (typeParameters)?
  ('extends' type)?
        ('implements' typeList)?
        classBody
    ;
```

Optional

Extended Backus–Naur form (EBNF)



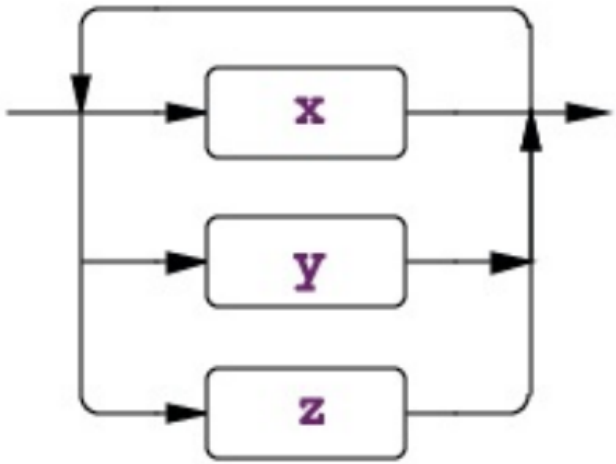
(x|y|z)*

Match an alternative within the subrule zero or more times. Here's an example:

```
annotationName : ID ('.' ID)* ;
```

Zero or More

Extended Backus–Naur form (EBNF)



(x|y|z)+

Match an alternative within the subrule one or more times. Here's an example:

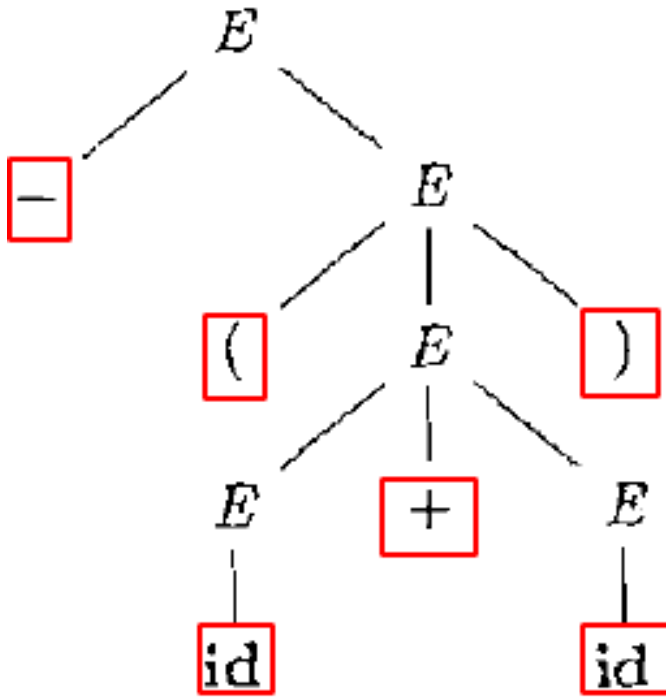
```
annotations : (annotation)+ ;
```

One or More

Cymbol.g4

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



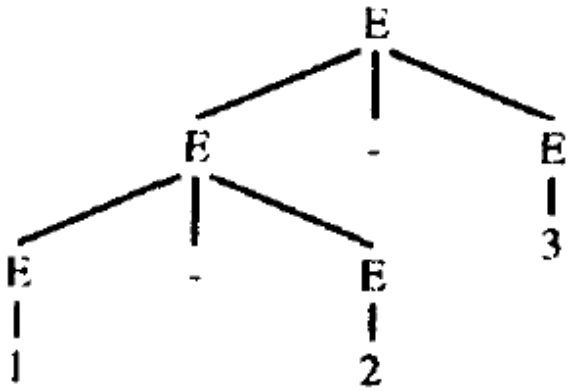
一棵语法分析树对应多个推导

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

$1 - 2 - 3$ 的语法树?

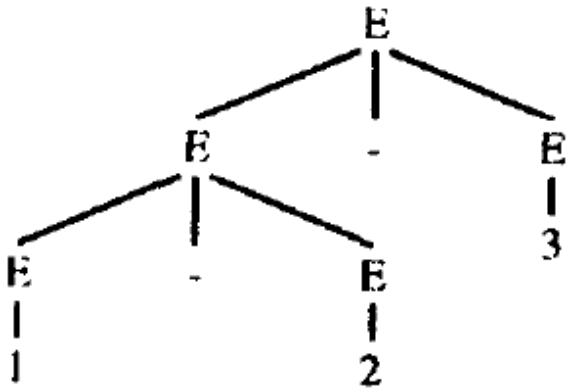
$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

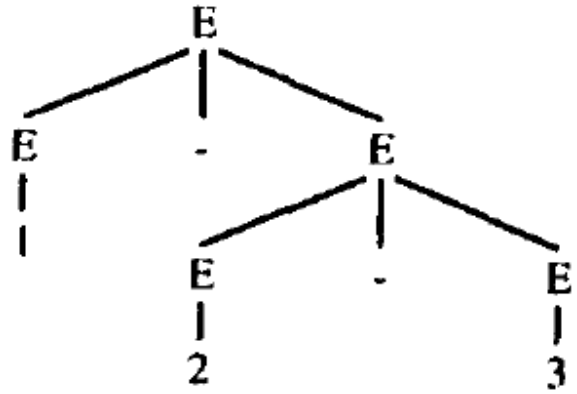
$1 - 2 - 3$ 的语法树?



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

$1 - 2 - 3$ 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?

“运算符结合性”导致的二义性

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,

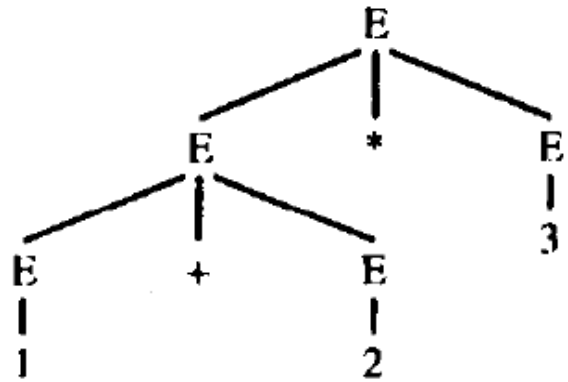
则文法 G 是二义性的。

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

$1 + 2 * 3$ 的语法树?

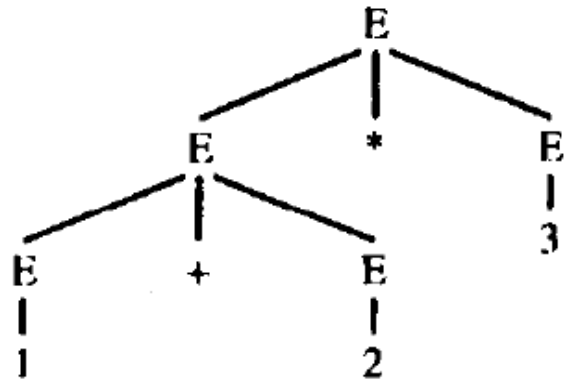
$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

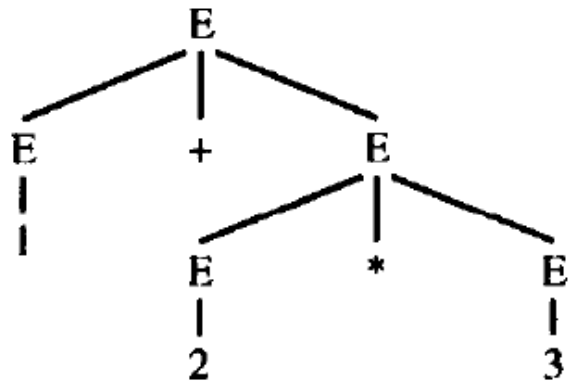
$1 + 2 * 3$ 的语法树?



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

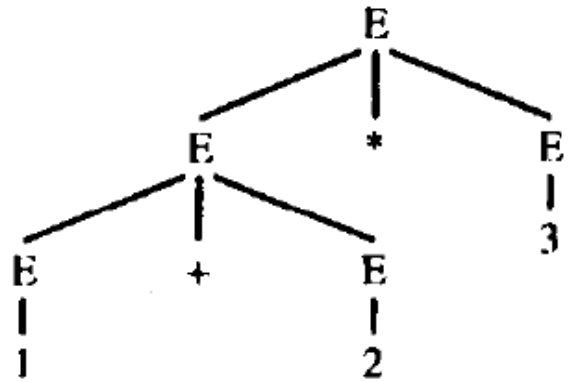
$1 + 2 * 3$ 的语法树?

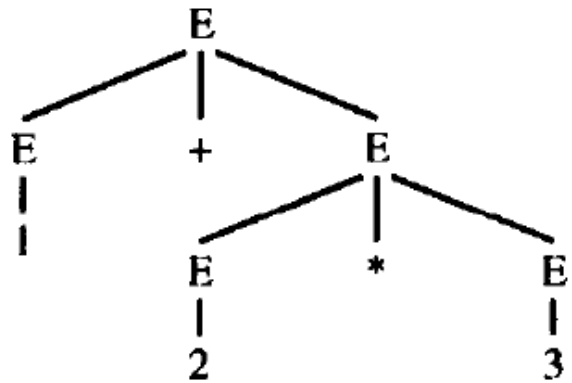




$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{num}$$

$1 + 2 * 3$ 的语法树?





“运算符优先级”导致的二义性

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

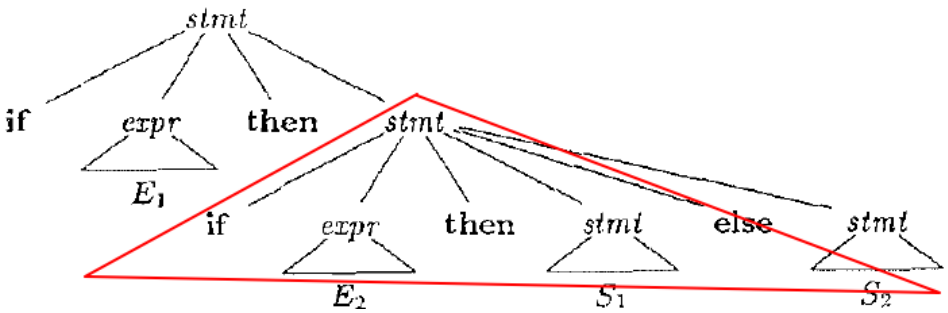
“悬空-else” 文法

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

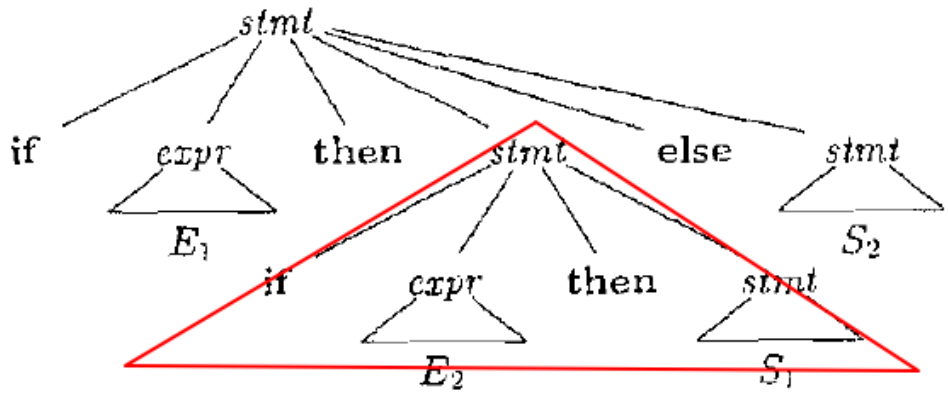
$$
\begin{array}{lcl}
\textit{stmt} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\
& | & \textbf{other}
\end{array}
$$

“悬空-else” 文法

if E_1 then if E_2 then S_1 else S_2



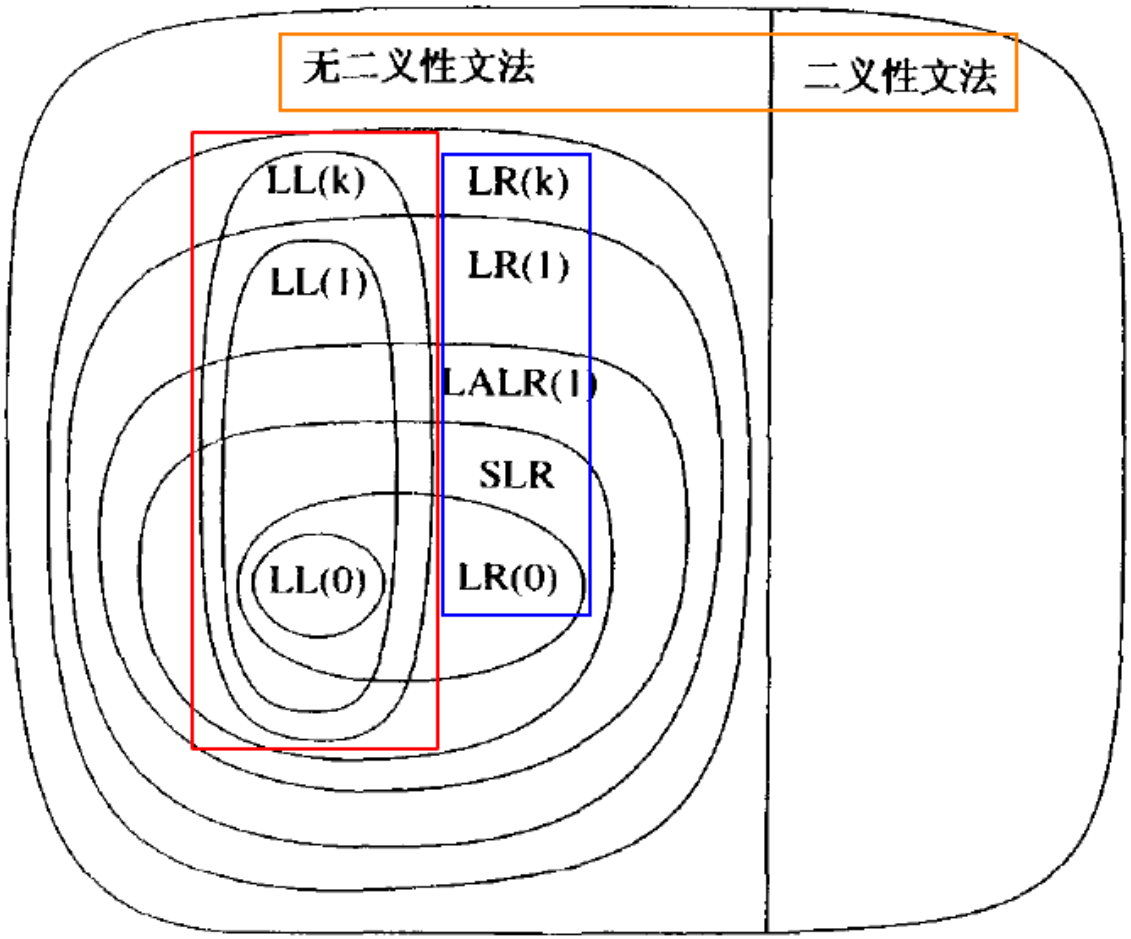
if E_1 then (if E_2 then S_1 else S_2)



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

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

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

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

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

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

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

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

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

左结合文法

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

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

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

左结合文法

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

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

右结合文法

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

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

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

左结合文法

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

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

右结合文法

使用**左（右）递归**实现**左（右）结合**

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

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

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

括号最先，先乘后加文法

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

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

无二义性的表达式文法

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

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

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

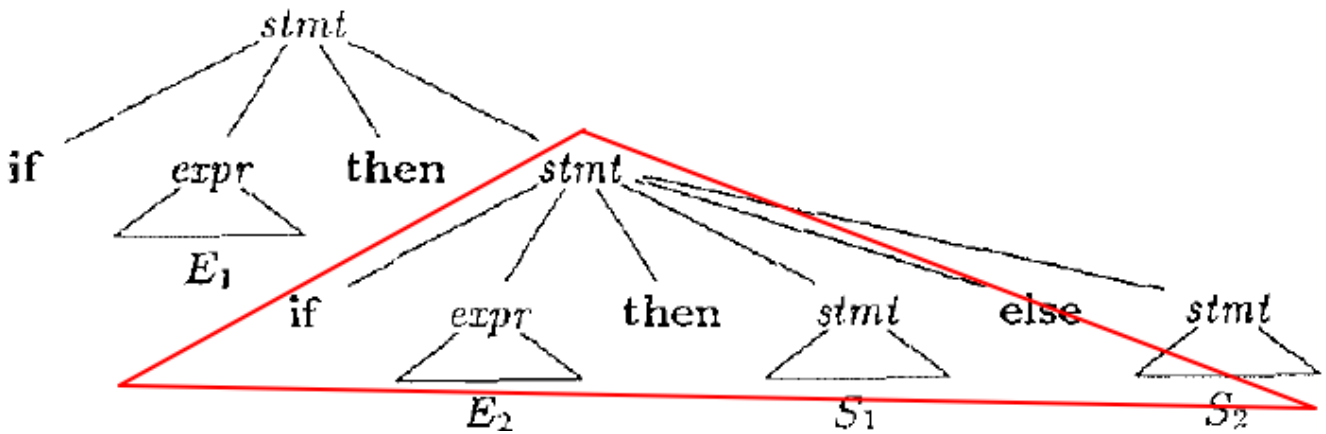
无二义性的表达式文法

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

将运算的 “先后” 顺序信息编码到语法树的 “层次” 结构中

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

if E_1 then if E_2 then S_1 else S_2

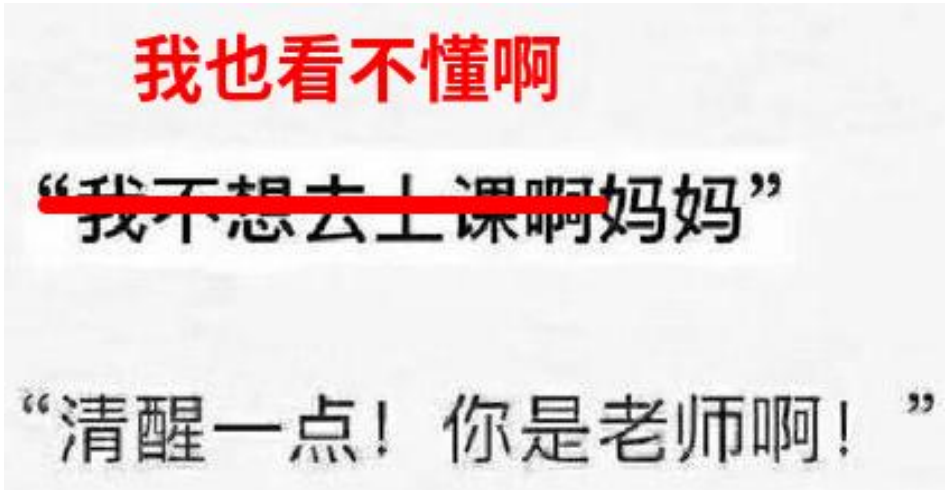


“每个else与**最近的尚未匹配的**then匹配”

$$
\begin{array}{rcl}
stmt & \longrightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

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

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

$$
\begin{array}{rcl}
stmt & \longrightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

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

$$
\begin{array}{rcl}
stmt & \longrightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

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

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

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

$$
\begin{array}{rcl}
\textit{stmt} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\
& | & \textbf{other}
\end{array}
$$

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

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

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

对推导步数作数学归纳

G' 是无二义性的

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

G' 是**无二义性**的

stmt	→	*matched_stmt*
	|	*open_stmt*
matched_stmt	→	**if** *expr* **then** *matched_stmt* **else** *matched_stmt*
	|	**other**
open_stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *matched_stmt* **else** *open_stmt*

每个句子对应的**语法分析树**是唯一的

G' 是**无二义性**的

stmt	→	*matched_stmt*
	|	*open_stmt*
matched_stmt	→	**if** *expr* **then** *matched_stmt* **else** *matched_stmt*
	|	**other**
open_stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *matched_stmt* **else** *open_stmt*

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

stmt	→	*matched_stmt*
	|	*open_stmt*
matched_stmt	→	**if** *expr* **then** *matched_stmt* **else** *matched_stmt*
	|	**other**
open_stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *matched_stmt* **else** *open_stmt*

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

stmt	→	*matched_stmt*
	|	*open_stmt*
matched_stmt	→	**if** *expr* **then** *matched_stmt* **else** *matched_stmt*
	|	**other**
open_stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *matched_stmt* **else** *open_stmt*

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

stmt	→	*matched_stmt*
	|	*open_stmt*
matched_stmt	→	**if** *expr* **then** *matched_stmt* **else** *matched_stmt*
	|	**other**
open_stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *matched_stmt* **else** *open_stmt*

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

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

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

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

Cymbol.g4

左递归文法 (Left Recursion)

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

左递归文法 (Left Recursion)

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

$$E \to TE'$$

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

将左递归转为**右递归**

左递归文法 (Left Recursion)

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

$$E \to TE'$$

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

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)

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

其中, β_i 都不以 A 开头

$$A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$

$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

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

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

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

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

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

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

$$E \to TE'$$

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

$$T \to FT'$$

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

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

间接左递归

$$S \to Aa \mid b$$

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

$$S \Longrightarrow Aa \Longrightarrow Sba$$

间接左递归

$$S \to Aa \mid b$$

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

$$S \Longrightarrow Aa \Longrightarrow Sba$$

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

$$A_k \to A_l\alpha \Longrightarrow l > k$$

$$S \to Aa \mid b$$

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

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

$$S \to Aa \mid b$$

$$A \to bdA' \mid A'$$

$$A' \to cA' \mid adA' \mid \epsilon$$

$$A_k \to A_l\alpha \implies l > k$$

$$(0)\ S \to Sa \mid Tbc \mid Td$$

$$(1)\ T \to Se \mid gh$$

$$(0)\ S \to Sa \mid Tbc \mid Td$$

$$(1)\ T \to Se \mid gh$$

$$S \to T(bc \mid d)S'$$

$$S' \to aS' \mid \epsilon$$

$$T \to ghT'$$

$$T' \to (bc \mid d)S'eT' \mid \epsilon$$

ANTLR4 可以处理直接左递归文法, **不要**改写文法

`Expr.g4`

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

$$E \rightarrow b$$

提取左公因子

$$S \rightarrow i\ E\ t\ S\ S' \mid a$$

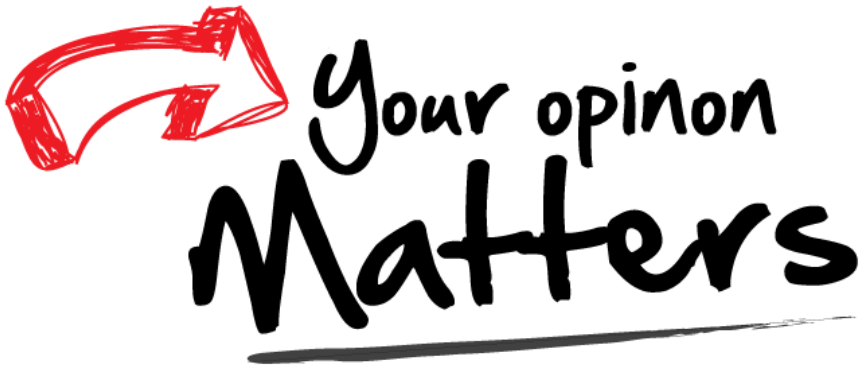
$$S' \rightarrow e\ S \mid \epsilon$$

$$E \rightarrow b$$

ANTLR4 可以处理有左公因子的文法, **不要**改写文法

`IfStat.g4`


Thank
You!



Office 926

hfwei@nju.edu.cn