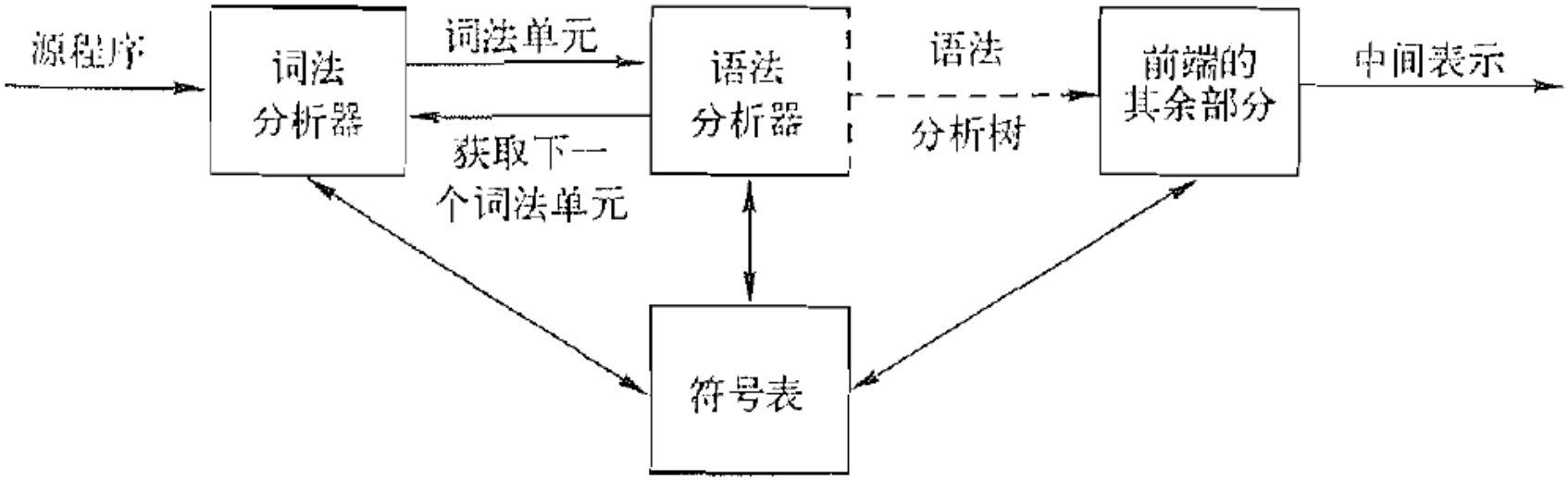
语法分析

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 22 日

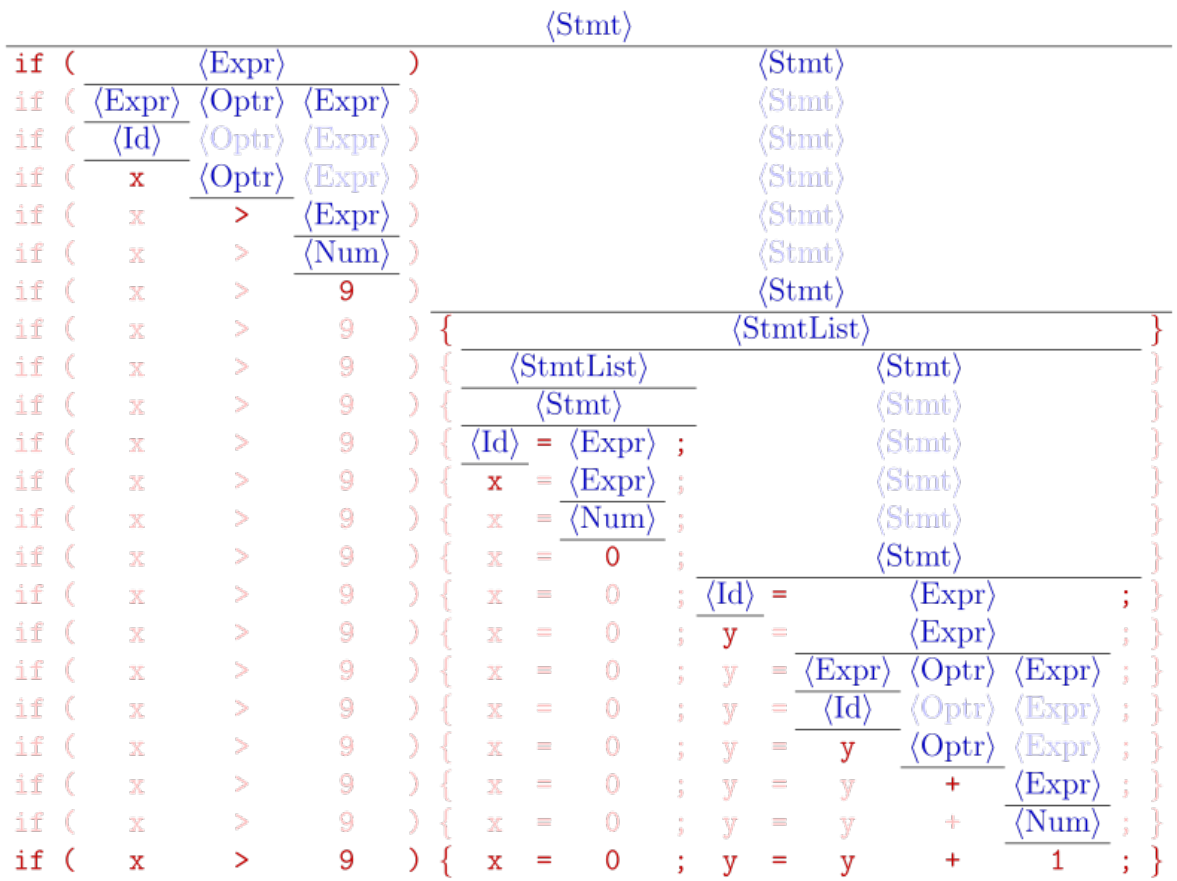
输入: 词法单元流 & **语言的语法规则**



输出: 语法分析树 (Parse Tree)

语法分析举例

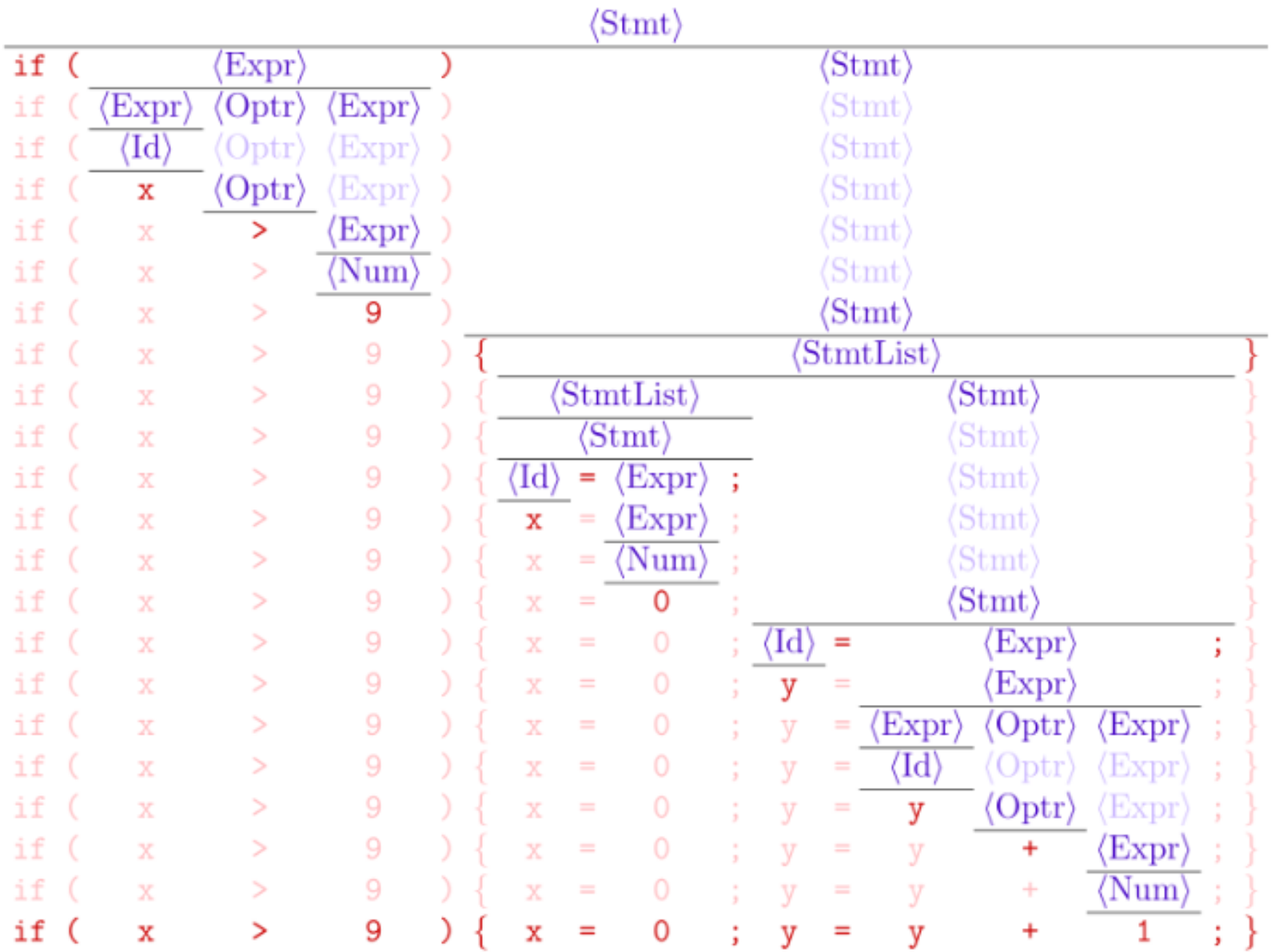
⟨Stmt⟩ → ⟨Id⟩ = ⟨Expr⟩ ;
⟨Stmt⟩ → { ⟨StmtList⟩ }
⟨Stmt⟩ → if (⟨Expr⟩) ⟨Stmt⟩
⟨StmtList⟩ → ⟨Stmt⟩
⟨StmtList⟩ → ⟨StmtList⟩ ⟨Stmt⟩
⟨Expr⟩ → ⟨Id⟩
⟨Expr⟩ → ⟨Num⟩
⟨Expr⟩ → ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
⟨Id⟩ → x
⟨Id⟩ → y
⟨Num⟩ → 0
⟨Num⟩ → 1
⟨Num⟩ → 9
⟨Optr⟩ → >
⟨Optr⟩ → +



语法分析阶段的主题之一：**上下文无关文法**

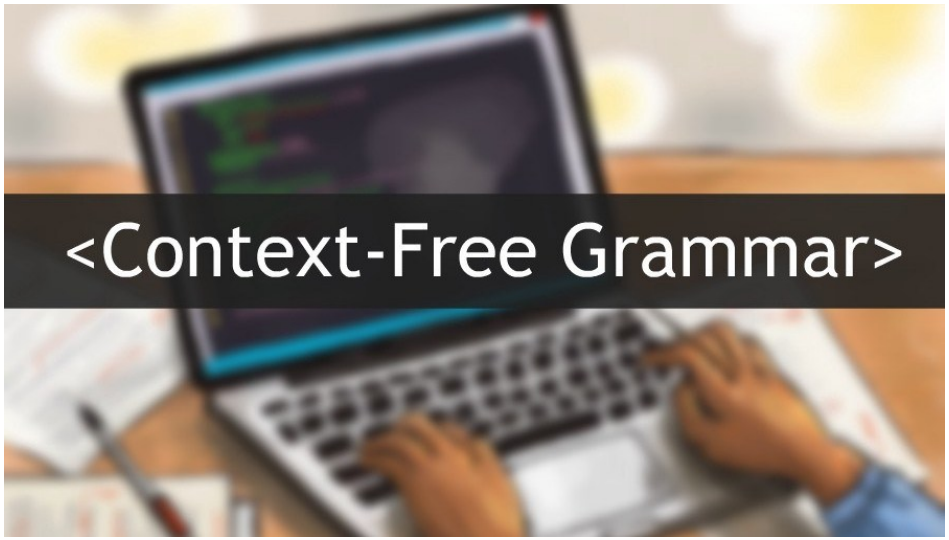
⟨Stmt⟩ → ⟨Id⟩ = ⟨Expr⟩ ;
⟨Stmt⟩ → { ⟨StmtList⟩ }
⟨Stmt⟩ → if (⟨Expr⟩) ⟨Stmt⟩
⟨StmtList⟩ → ⟨Stmt⟩
⟨StmtList⟩ → ⟨StmtList⟩ ⟨Stmt⟩
⟨Expr⟩ → ⟨Id⟩
⟨Expr⟩ → ⟨Num⟩
⟨Expr⟩ → ⟨Expr⟩ ⟨Optr⟩ ⟨Expr⟩
⟨Id⟩ → x
⟨Id⟩ → y
⟨Num⟩ → 0
⟨Num⟩ → 1
⟨Num⟩ → 9
⟨Optr⟩ → >
⟨Optr⟩ → +

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。

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

$$S \to aSb$$

$$S \to \epsilon$$

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

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

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

条件语句文法

悬空 (Dangling)-else 文法

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$S \rightarrow \text{begin } S\ L$

$S \rightarrow \text{print } E$

$L \rightarrow \text{end}$

$L \rightarrow ;\ S\ L$

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

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 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*。

推导 (Derivation)

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

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)

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

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

推导 (Derivation)

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

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

$E \implies -E$: 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

推导 (Derivation)

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

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

$E \implies -E$: 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 $S \overset{*}{\Rightarrow} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**句型**。

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

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

Definition (Sentential Form; 句型)

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

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

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

Definition (Sentence; 句子)

如果 $S \overset{*}{\Rightarrow} w$, 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。

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

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?

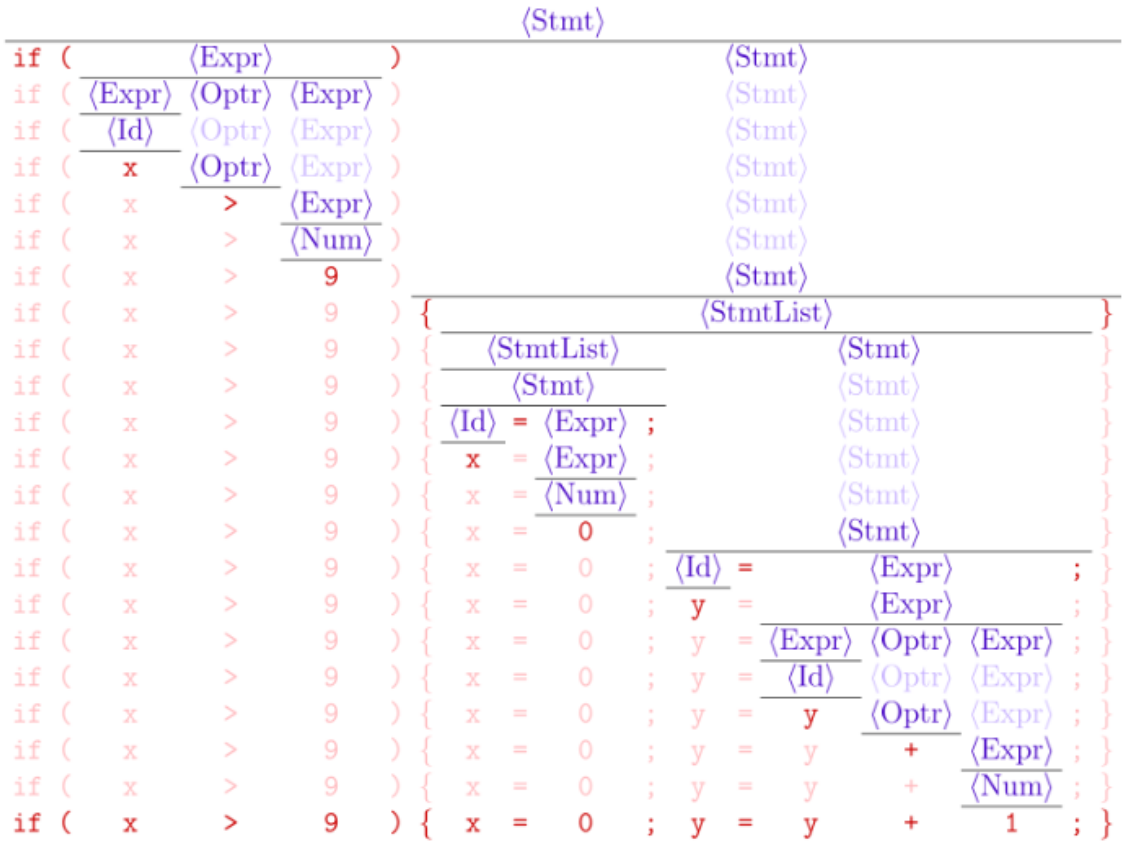
给定字符串 $x \in T^*$, $x \in L(G)$?

(即, 检查 x 是否符合文法 G)

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

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) =$

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) = \{$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$\}$

$$S \to aSb$$

$$S \to \epsilon$$

$L(G) =$

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) = \{\text{良匹配括号串}\}$

$$S \to aSb$$

$$S \to \epsilon$$

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

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to a$$

$$S \to b$$

$$S \to \epsilon$$

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to a$$

$$S \to b$$

$$S \to \epsilon$$

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

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

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

$$S \to bSbb \mid A$$

$$A \to aA \mid \epsilon$$

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

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

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

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

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

$$S \to T \mid U$$

$$T \to VaT \mid VaV$$

$$U \to VbU \mid VbV$$

$$V \to aVbV \mid bVaV \mid \epsilon$$

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

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

练习 (非作业): 证明之

L-System (注: 这不是上下文无关文法, 但精神高度一致)

variables : A B

constants : + −

start : A

rules : (A → B−A−B), (B → A+B+A)

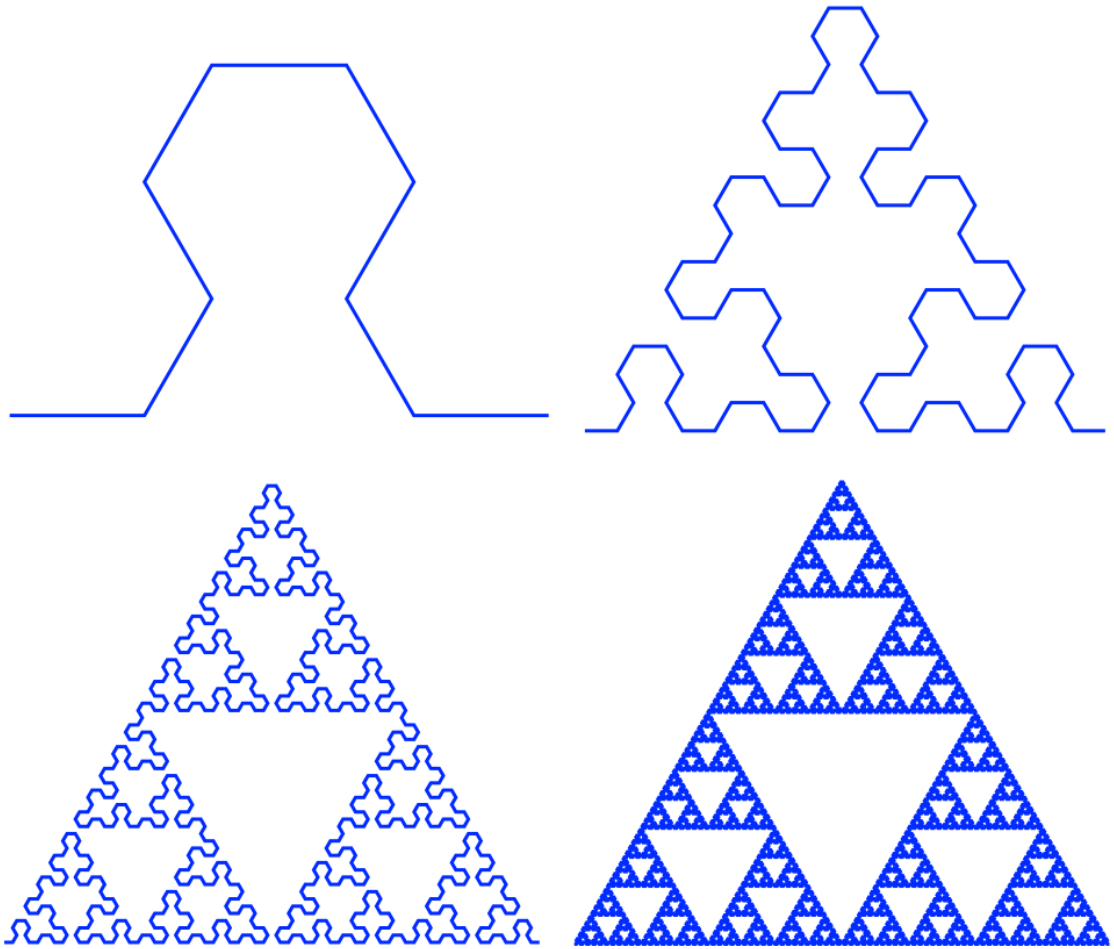
angle : 60°

A, B：向前移动并画线

$+$：左转

$-$：右转

每一步都**并行地**应用**所有**规则



variables : X Y

constants : F + −

start : FX

rules : (X → X+YF+), (Y → −FX−Y)

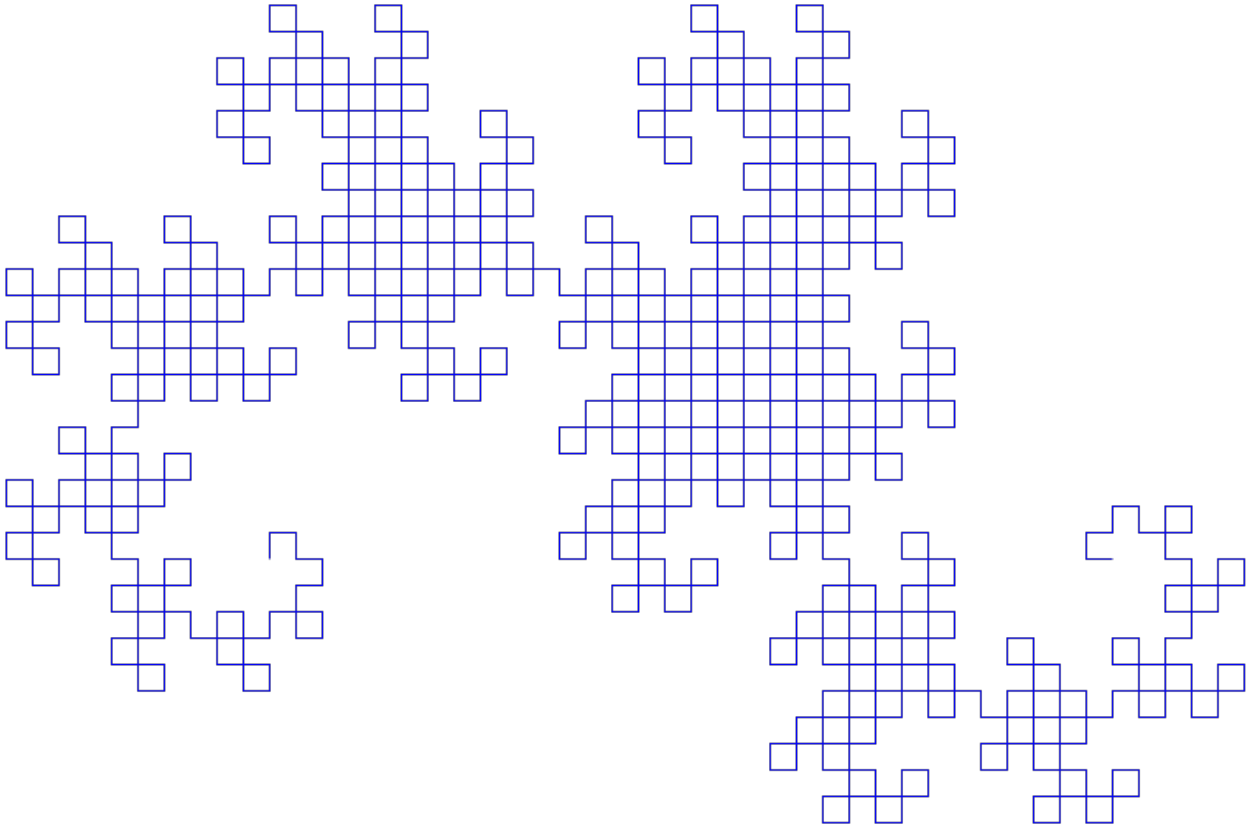
angle : 90°

F：向前移动并画线

$+$：右转

$-$：左转

每一步都**并行地**应用**所有**规则



最左 (leftmost) 推导与最右 (rightmost) 推导

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

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

最左 (leftmost) 推导与最右 (rightmost) 推导

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

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

$E \underset{\text{lm}}{\Longrightarrow} -E$: 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

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

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

$E \underset{\text{lm}}{\Longrightarrow} -E$: 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 $S \overset{*}{\underset{\text{lm}}{\Longrightarrow}} \alpha$, 且 $\alpha \in (T \cup N)^*$, 则称 α 是文法 G 的一个**最左句型**。

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

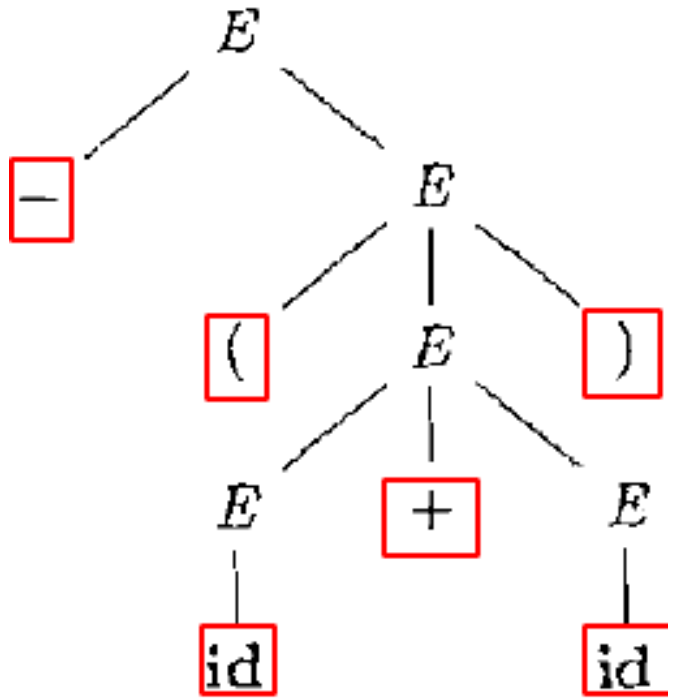
Definition (Right-sentential Form; 最右句型)

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

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

语法分析树

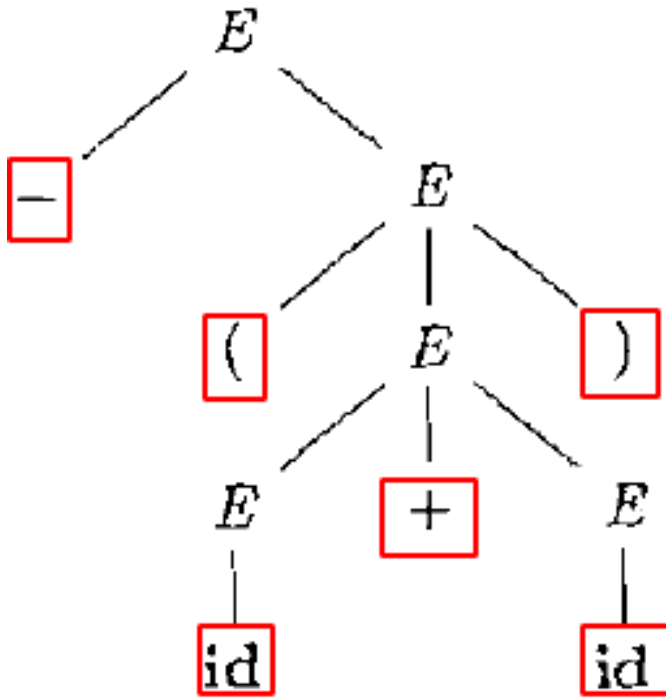
语法分析树是静态的, 它不关心动态的推导顺序



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

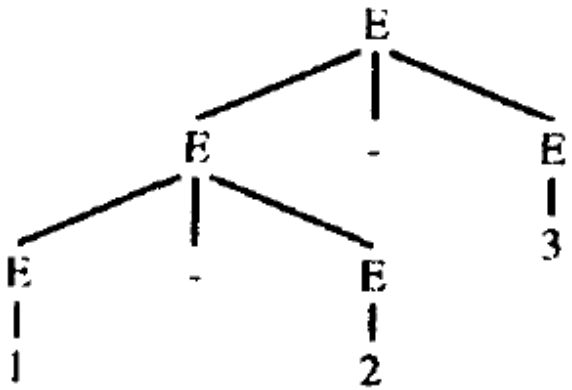
但是, 一棵语法分析树与**最左 (最右) 推导**一一对应

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

$1 - 2 - 3$ 的语法树?

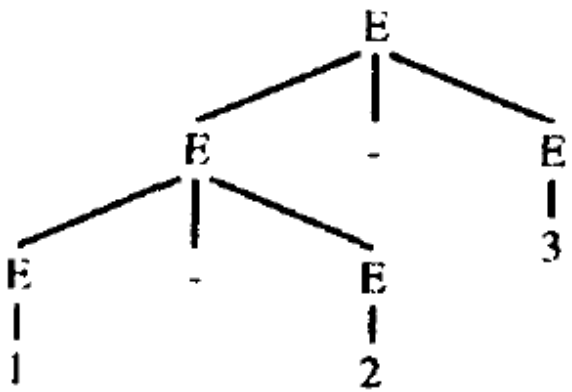
$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{number}$$

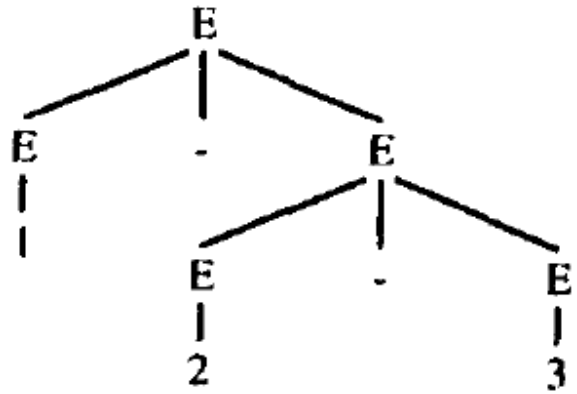
$1 - 2 - 3$ 的语法树?



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

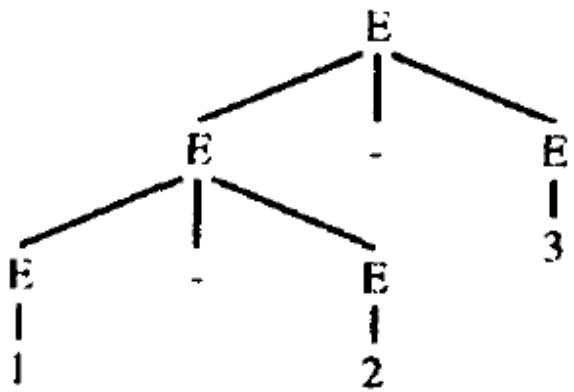
$1 - 2 - 3$ 的语法树?

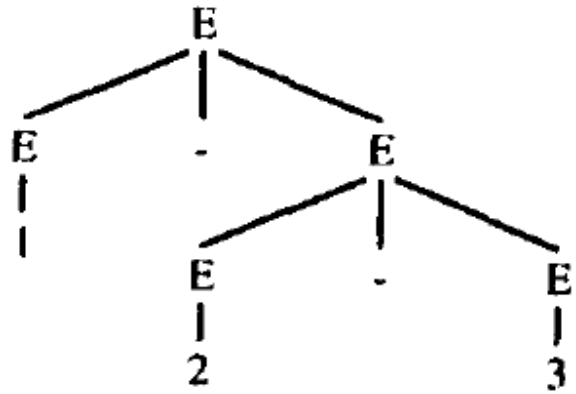




$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{number}$$

$1 - 2 - 3$ 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,

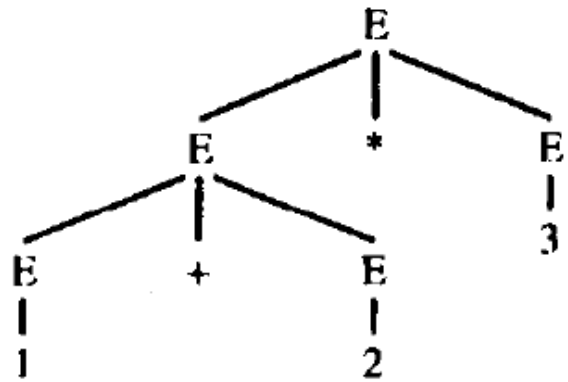
则文法 G 是二义性的。

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

$1 + 2 * 3$ 的语法树?

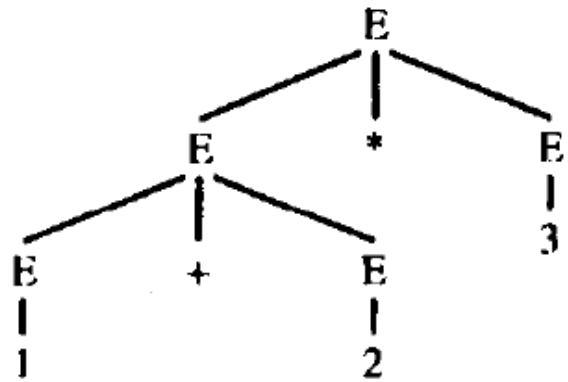
$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \textbf{id} \mid \textbf{number}$$

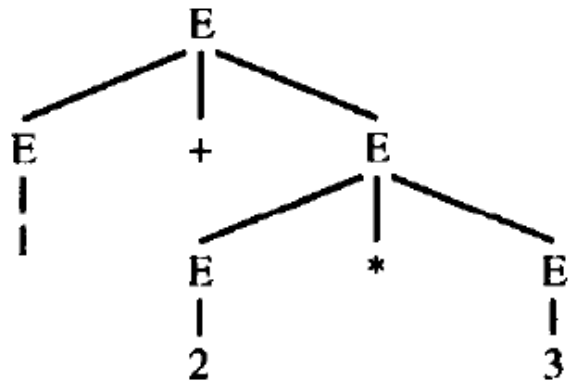
$1 + 2 * 3$ 的语法树?



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

$1 + 2 * 3$ 的语法树?





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

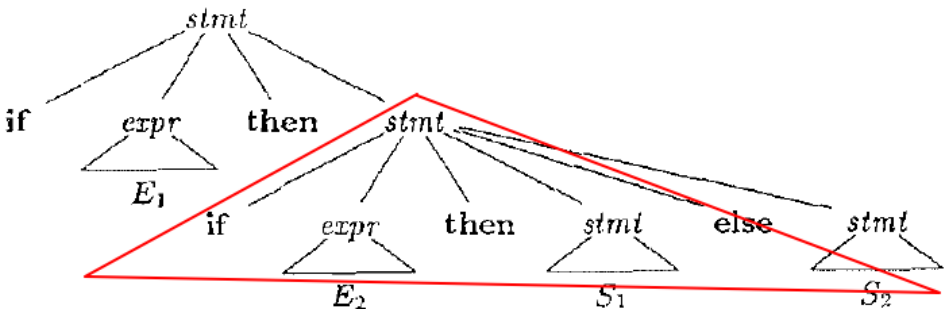
“悬空-else” 文法

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

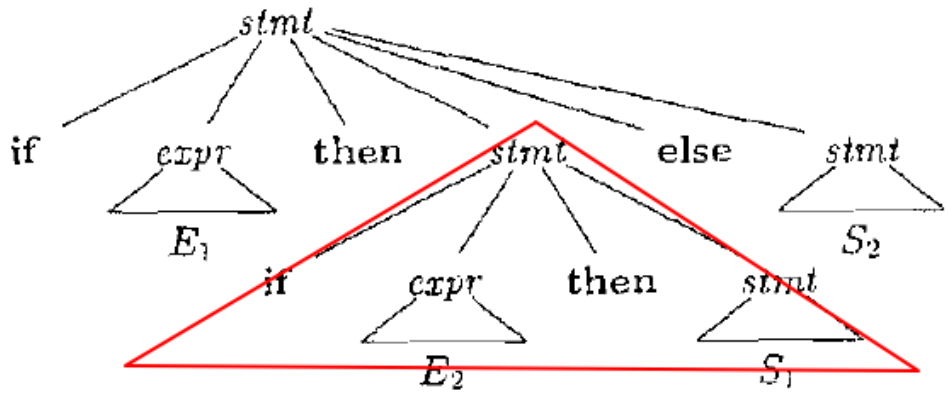
$$
\begin{array}{lcl}
\textit{stmt} & \longrightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\
& | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\
& | & \textbf{other}
\end{array}
$$

“悬空-else” 文法

if E_1 then if E_2 then S_1 else S_2



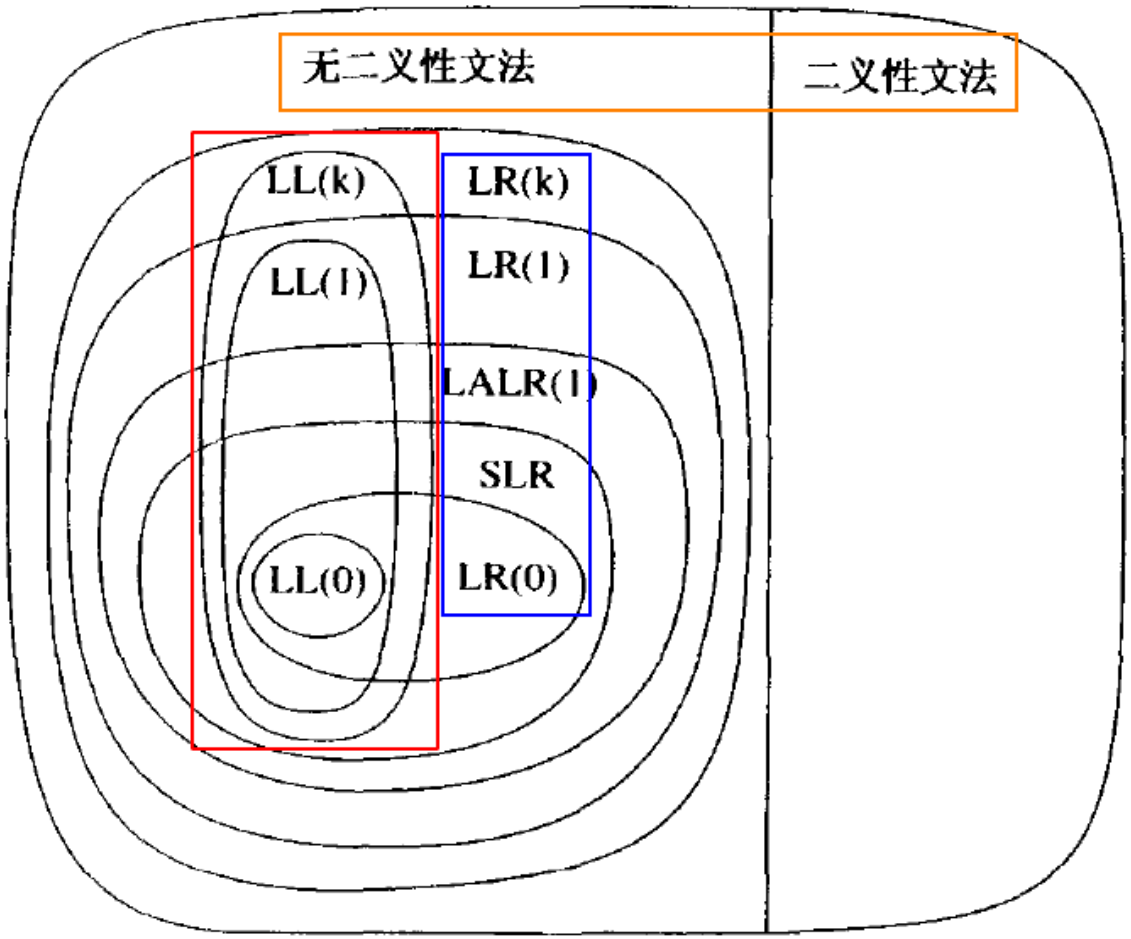
if E_1 then (if E_2 then S_1 else S_2)



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

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

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

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

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

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

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

$$E \to E + T$$

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

左结合文法

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

$$E \to E + T$$

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

左结合文法

$$E \to T + E$$

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

右结合文法

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

$$E \to E + T$$

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

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**

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

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

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

括号最先，先乘后加文法

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

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

无二义性的表达式文法

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

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

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

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

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

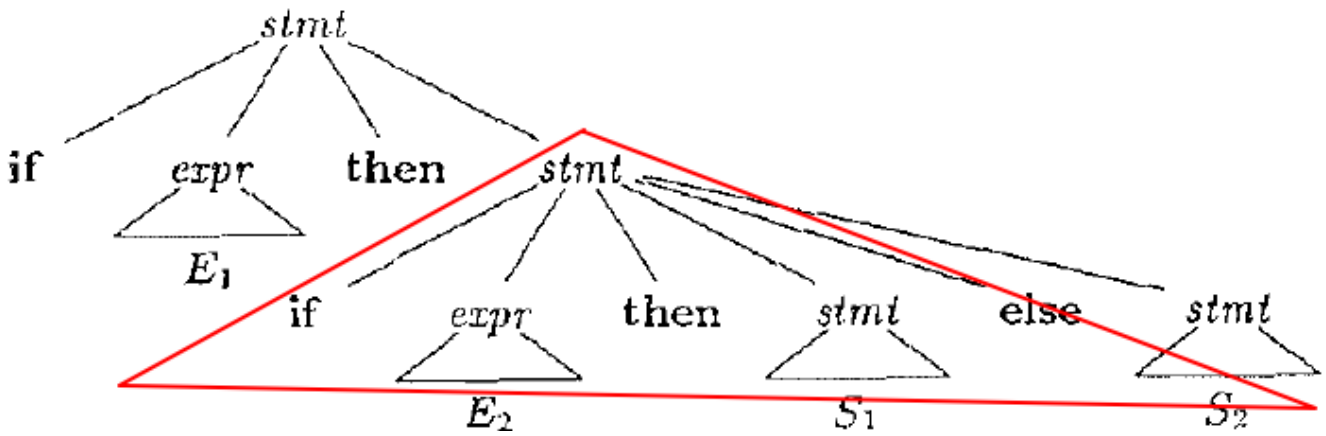
无二义性的表达式文法

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

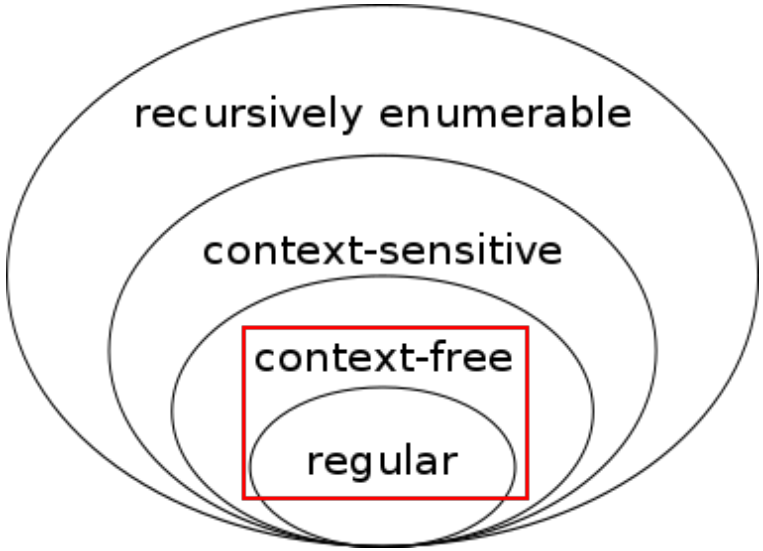


“每个else与**最近的尚未匹配的**then匹配”

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*

图 4-10　**if-then-else** 语句的无二义性方法

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

$$S \to aSb$$

$$S \to \epsilon$$

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

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。

Proof.

使用数学归纳法

假设存在正则表达式 r: $L(r) = L$

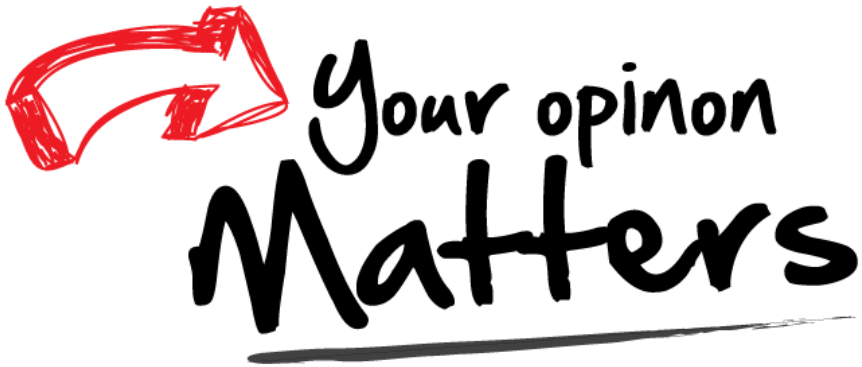
存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

□

Pumping Lemma for Regular Languages

Pumping Lemma for Context-free Languages


Thank
You!



Office 926

hfwei@nju.edu.cn