



# 第3讲

## 语法分析-上下文无关文法

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年03月06日



# 简单计算器程序

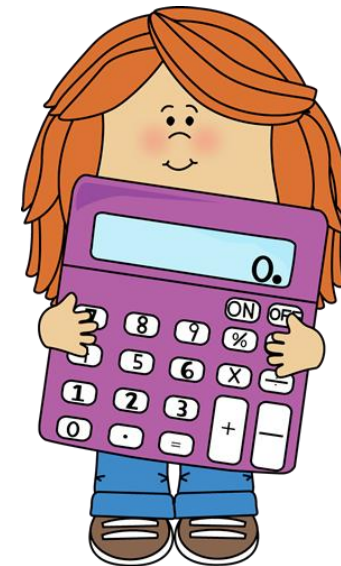


$1 + (2 - 3)$  合法

$1 + 2 - 3 +$  非法

$1\ 2 + - 3$  非法

$1 + 2 - a$  非法



语法分析的目的是教会计算机判断输入合法性



# 如何判定输入合法性呢？



- 首先要规定好合法的基本单元——词法分析

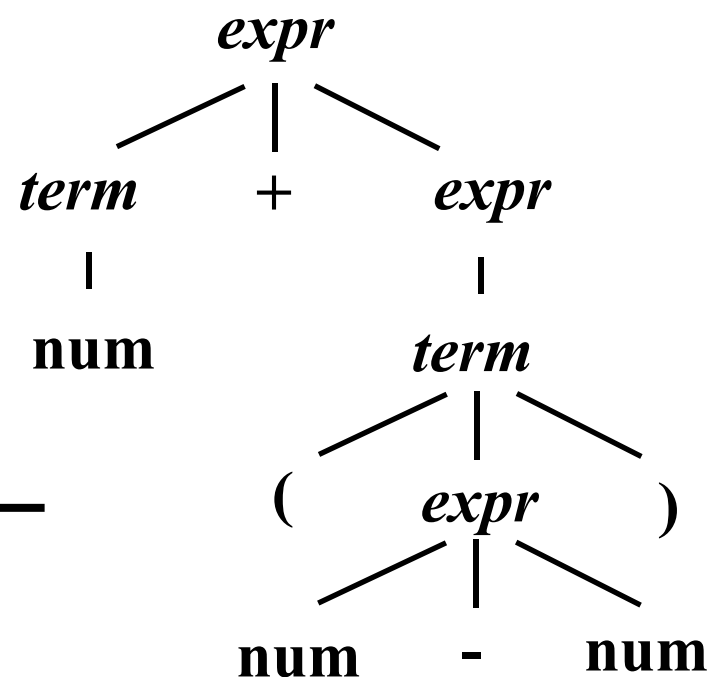
- 由0-9组成的数字(num)和符号+、-、(、)

- 其次要理解算术表达式的构成

- 大表达式(expr)可拆为若干子表达式
- 拆解过程是递归的
- 直至看到基本单元

1 + (2 - 3)

语法树



问题一：如何描述编程语言的语法结构？



# John Backus -1977图灵奖



- 提出了多种高级编程语言

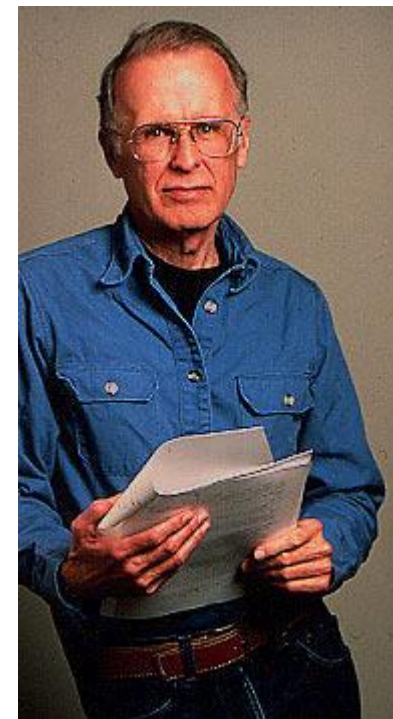
- Speedcoding -> FORTRAN -> ALGOL 58 -> ALGOL 60 -> FP

- 提出了编译技术的理论基础

- 巴科斯范式 (Backus-Naur Form)
- 上下文无关文法

- 对计算机科学影响巨大

- 诞生了许多理论研究成果
- 现代编译器还保留了FORTRAN I的大概架构



弗吉尼亚大学化学专业，哥伦比亚大学数学专业，曾服务于阿波罗登月计划



- 上下文无关文法的形式化定义：

$$(V_T, V_N, S, P)$$

$V_T$  : 终结符集合

➤ 终结符：是文法所定义的语言的基本符号，也称为“*token*”

➤ 例：  $V_T = \{ \text{num}, +, -, (, ) \}$



- 上下文无关文法的形式化定义：

$$(V_T, V_N, S, P)$$

$V_N$ ：非终结符集合

- 非空有限集合， $V_T \cap V_N = \emptyset$
- **非终结符**：表示语法成分的符号，存放中间结果，也称为“语法变量”
- 例： $V_N = \{ \textit{expr}, \textit{term} \}$



- 上下文无关文法的形式化定义：

$$(V_T, V_N, S, P)$$

**$S$  : 开始符号**

- 属于非终结符，是该文法中最大的语法成分，分析开始的地方
- 例： $S = \textit{expr}$



- 上下文无关文法的形式化定义：

$$(V_T, V_N, S, \textcolor{red}{P})$$

$P$  : 产生式集合

➤ **产生式**：描述了将终结符和非终结符组合成串的方法

➤ 例：  $P = \{ \textit{expr} \rightarrow \textit{term} \mid \textit{term} + \textit{expr} \mid \textit{term} - \textit{expr}$

$\textit{term} \rightarrow \textit{num} \mid (\textit{expr}) \}$





## □ 例：描述简单计算器的上下文无关文法

四元组：  $(V_T, V_N, S, P)$

$(\{\text{num}, +, -, (, )\}, \{expr, term\}, expr, P)$

$P = \{expr \rightarrow term \mid term + expr \mid term - expr$

$term \rightarrow num \mid (expr) \}$

问题二：给定文法，如何判定输入串属于文法规定的语言呢？



- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$$term \rightarrow num \mid (expr)$$

- 展示  $1 + (2 - 3)$  的构造过程

$expr$

$expr$

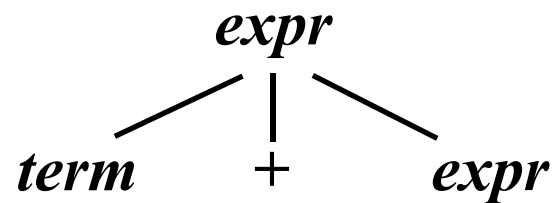


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$$term \rightarrow num \mid (expr)$$

- 展示  $1 + (2 - 3)$  的构造过程

$$expr \Rightarrow term + expr$$



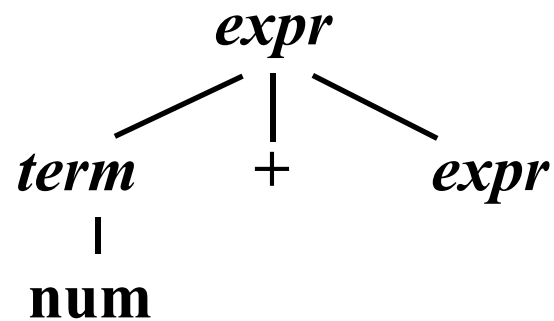


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$term \rightarrow \text{num} \mid (expr)$

- 展示  $1 + (2 - 3)$  的构造过程

$expr \Rightarrow term + expr$   
 $\Rightarrow num + expr$



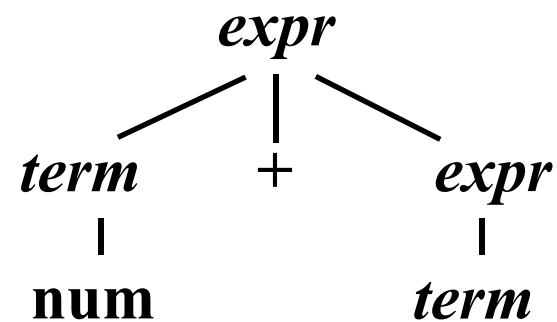


- 例：对于文法  $expr \rightarrow \textcolor{blue}{term} \mid term + expr \mid term - expr$

$$term \rightarrow num \mid (expr)$$

- 展示  $1 + (2 - 3)$  的构造过程

$$\begin{aligned} expr &\Rightarrow term + expr \\ &\Rightarrow num + expr \\ &\Rightarrow num + term \end{aligned}$$



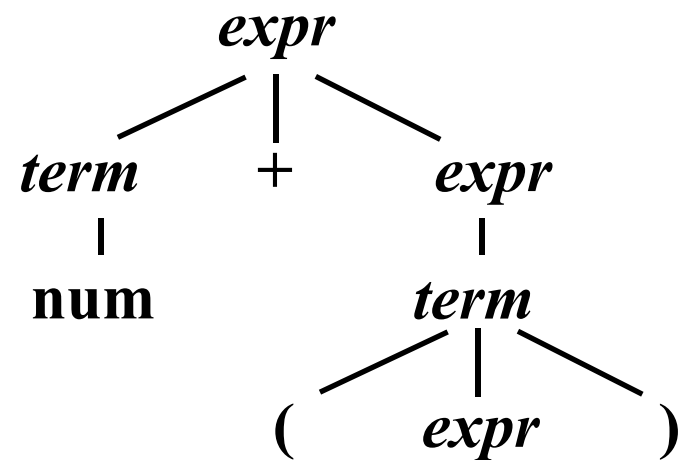


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$term \rightarrow num \mid (expr)$

- 展示  $1 + (2 - 3)$  的构造过程

$$\begin{aligned} expr &\Rightarrow term + expr \\ &\Rightarrow num + expr \\ &\Rightarrow num + term \\ &\Rightarrow num + (expr) \end{aligned}$$



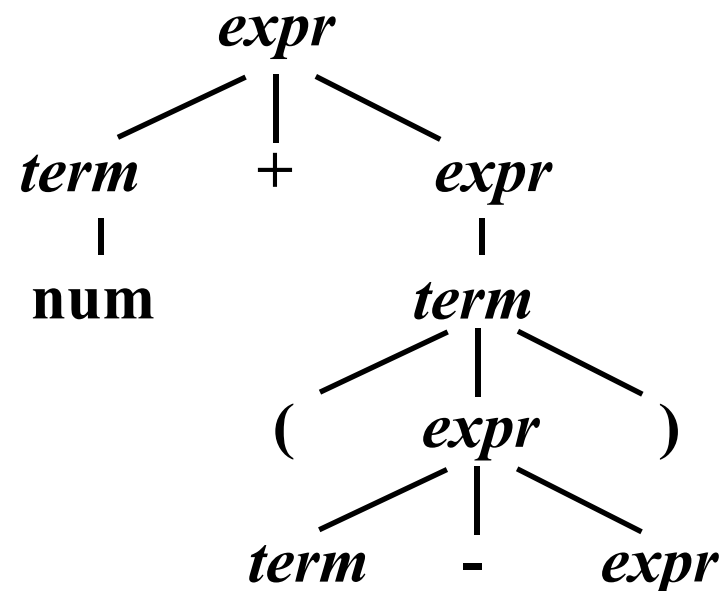


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid \textcolor{blue}{term - expr}$

$$term \rightarrow num \mid (expr)$$

- 展示  $1 + (2 - 3)$  的构造过程

$$\begin{aligned} expr &\Rightarrow term + expr \\ &\Rightarrow num + expr \\ &\Rightarrow num + term \\ &\Rightarrow num + (expr) \\ &\Rightarrow num + (term - expr) \end{aligned}$$



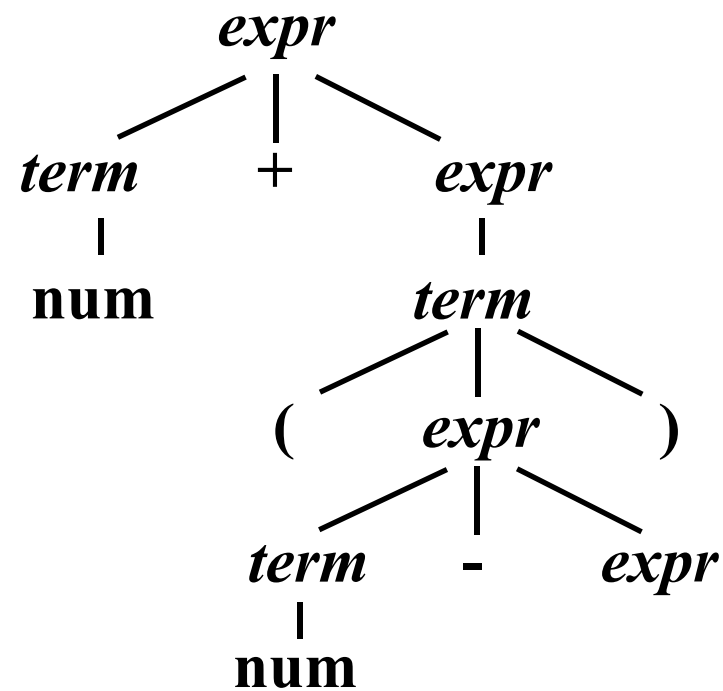


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$term \rightarrow \text{num} \mid (expr)$

- 展示  $1 + (2 - 3)$  的构造过程

$$\begin{aligned} expr &\Rightarrow term + expr \\ &\Rightarrow num + expr \\ &\Rightarrow num + term \\ &\Rightarrow num + (expr) \\ &\Rightarrow num + (term - expr) \\ &\Rightarrow num + (num - expr) \end{aligned}$$





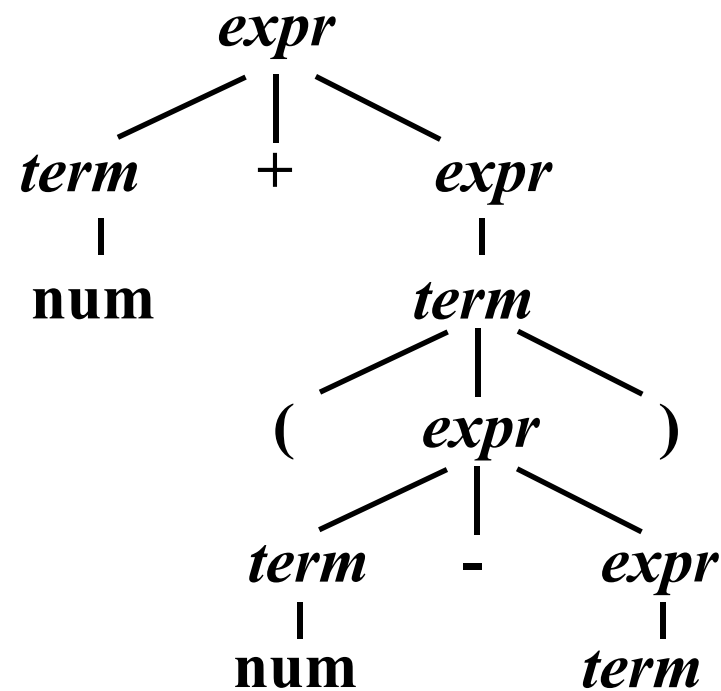


- 例：对于文法  $expr \rightarrow \textcolor{blue}{term} \mid term + expr \mid term - expr$

$$term \rightarrow num \mid (expr)$$

- 展示  $1 + (2 - 3)$  的构造过程

$$\begin{aligned} expr &\Rightarrow term + expr \\ &\Rightarrow num + expr \\ &\Rightarrow num + term \\ &\Rightarrow num + (expr) \\ &\Rightarrow num + (term - expr) \\ &\Rightarrow num + (num - expr) \\ &\Rightarrow num + (num - term) \end{aligned}$$



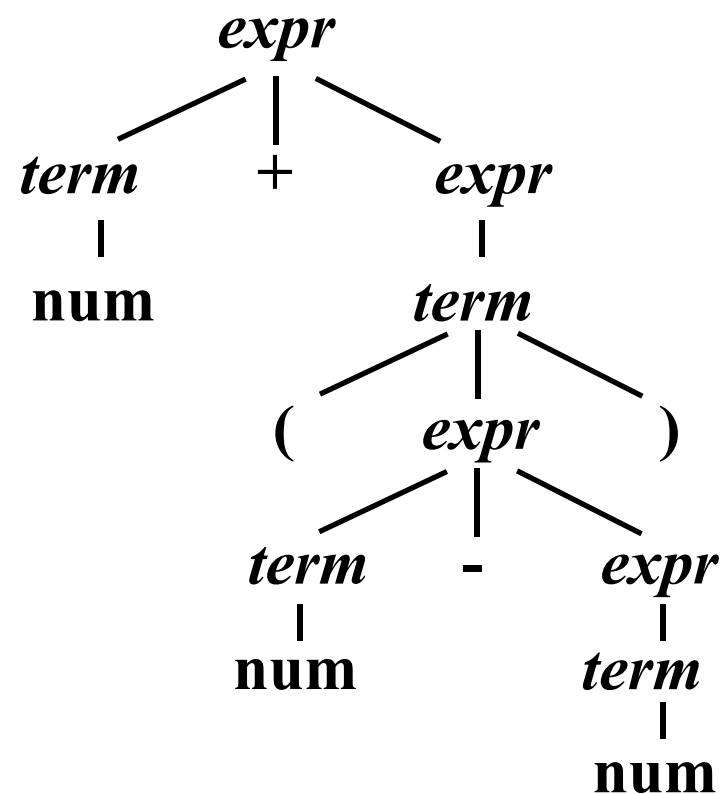


- 例：对于文法  $expr \rightarrow term \mid term + expr \mid term - expr$

$term \rightarrow \text{num} \mid (expr)$

- 展示  $1 + (2 - 3)$  的构造过程

$expr \Rightarrow term + expr$   
 $\Rightarrow num + expr$   
 $\Rightarrow num + term$   
 $\Rightarrow num + (expr)$   
 $\Rightarrow num + (term - expr)$   
 $\Rightarrow num + (num - expr)$   
 $\Rightarrow num + (num - term)$   
 $\Rightarrow num + (num - num)$





- **推导 (Derivation)**

- 是从文法推出文法所描述的语言中所包含的合法串集合的动作
- 把产生式看成重写规则，把符号串中的非终结符用其产生式右部的串来代替

- **例**  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\text{id} + E) \Rightarrow -(\text{id} + \text{id})$$

- **记法:**

- $S \Rightarrow^* a$ : 0步或多步推导
- $S \Rightarrow^+ w$ : 1步或多步推导



# 最左推导和最右推导



- 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

- 最左推导 (leftmost derivation)

- 每步代换最左边的非终结符

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

- 最右推导 (rightmost or canonical derivation, 规范推导)

- 每步代换最右边的非终结符

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



# 思考题？



- **上下文无关**是什么意思？

- 上下文无关指的是在文法推导的每一步

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

符号串  $\gamma$  仅依据  $A$  的产生式推导，而无需依赖  $A$  的上下文  $\alpha$  和  $\beta$



## • 上下文无关语言

- 上下文无关**文法** $G$ 产生的**语言**: 从**开始符号** $S$ 出发, 经 $\Rightarrow^+$ 推导所能到达的所有仅由终结符组成的串
- 句型(sentential form):  $S \Rightarrow^* a$ ,  $S$ 是开始符号,  $a$ 是由**终结符和/或非终结符**组成的串, 则 $a$ 是文法 $G$ 的句型
- 句子(sentence): 仅由**终结符**组成的句型

## • 等价的文法

- 它们产生同样的语言



# 最左推导和最右推导



- 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

- 最左推导 (leftmost derivation)

- 每步代换最左边的非终结符

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

- 最右推导 (rightmost or canonical derivation, 规范推导)

- 每步代换最右边的非终结符

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

褐红色标出的均是句型



# 最左推导和最右推导



- 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

- 最左推导 (leftmost derivation)

- 每步代换最左边的非终结符

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

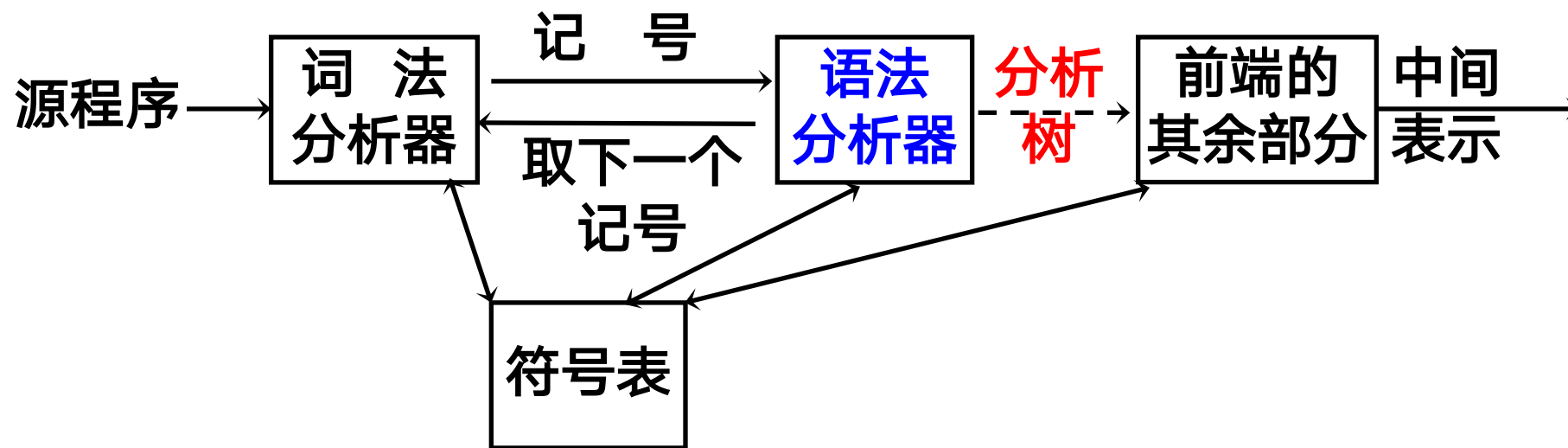
褐红色标出的均是句子

- 最右推导 (rightmost or canonical derivation, 规范推导)

- 每步代换最右边的非终结符

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





- 语法分析器简介
- 上下文无关文法CFG：定义、推导
- 思考与拓展
  - 正则表达式与CFG的联系与区别
  - CFG二义性及消除方法



- 正则表达式的表达能力

- 定义一些简单的语言，能表示给定结构的固定次数的重复或者没有指定次数的重复

例:  $a(ba)^5, a(ba)^*$

- 不能用于描述配对或嵌套的结构

例1: 配对括号串的集合，如不能表达  $(^n)^n, n \geq 0$

例2:  $\{wcw \mid w \text{ 是 } a \text{ 和 } b \text{ 的串}\}$



- 正则表达式的表达能力

- 定义一些简单的语言，能表示给定结构的固定次数的重复或者没有指定次数的重复

例:  $a(ba)^5, a(ba)^*$

- 不能用于描述配对或嵌套的结构

例1: 配对括号串的集合，如不能表达  $(^n)^n, n \geq 0$

例2:  $\{wcw \mid w \text{ 是 } a \text{ 和 } b \text{ 的串}\}$

原因: 有限自动机无法记录访问同一状态的次数



# 思考题



- 请写出语言  $\{(n)^n \mid n \geq 0\}$  的CFG文法



# 思考题



• 请写出语言  $\{(n)^n \mid n \geq 0\}$  的CFG文法

•  $S \rightarrow (S) \mid \varepsilon$



- 都能表示语言
- 能用正则表达式表示的语言都能用CFG表示

- 正则表达式

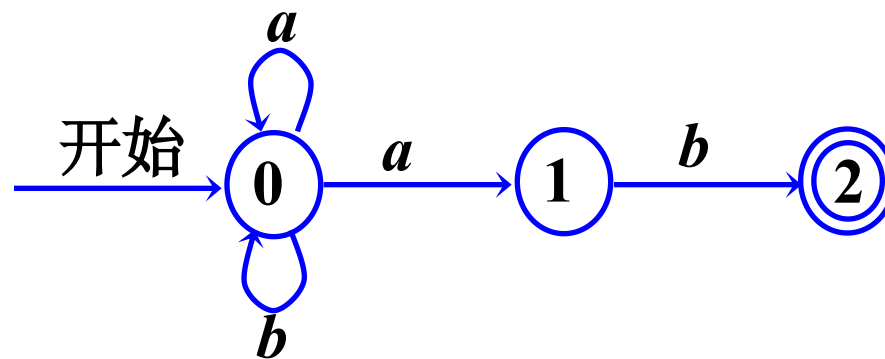
$(a|b)^*ab$

- CFG文法

$A_0 \rightarrow a A_0 \mid b A_0 \mid a A_1$

$A_1 \rightarrow b A_2$

$A_2 \rightarrow \varepsilon$





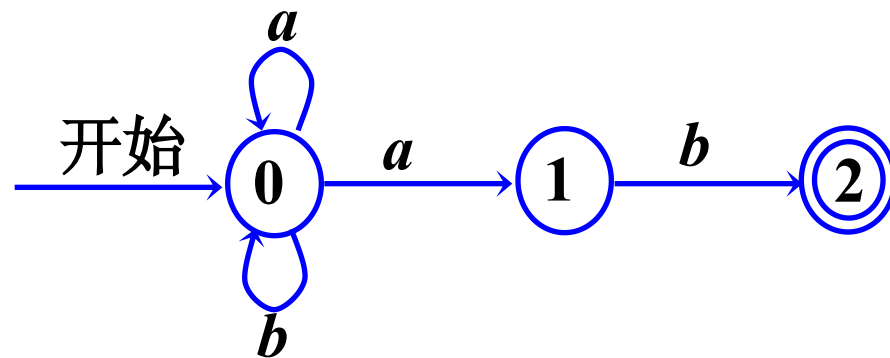
## • NFA $\rightarrow$ 上下文无关文法

- 确定终结符集合
- 为每个状态引入一个非终结符  $A_i$
- 如果状态  $i$  有一个  $a$  转换到状态  $j$ , 引入产生式  $A_i \rightarrow aA_j$ , 如果  $i$  是接受状态, 则引入  $A_i \rightarrow \varepsilon$

$$A_0 \rightarrow a A_0 \mid b A_0 \mid a A_1$$

$$A_1 \rightarrow b A_2$$

$$A_2 \rightarrow \varepsilon$$





# 思考题



- 请为描述所有由0或1组成的回文字符串的语言设计CFG文法





# 思考题



- 请为描述所有由0或1组成的回文字符串的语言设计CFG文法
  - $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$



- 文法的某些句子存在**不止一种**最左(最右)推导, 或者**不止一棵**分析树, 则该文法是二义的。



• 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

•  $\text{id} * \text{id} + \text{id}$  有两个不同的最左推导

$$E \Rightarrow E * E$$

$$\Rightarrow \text{id} * E$$

$$\Rightarrow \text{id} * E + E$$

$$\Rightarrow \text{id} * \text{id} + E$$

$$\Rightarrow \text{id} * \text{id} + \text{id}$$

$$E \Rightarrow E + E$$

$$\Rightarrow E * E + E$$

$$\Rightarrow \text{id} * E + E$$

$$\Rightarrow \text{id} * \text{id} + E$$

$$\Rightarrow \text{id} * \text{id} + \text{id}$$



• 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

•  $\text{id} * \text{id} + \text{id}$  有两棵不同的分析树

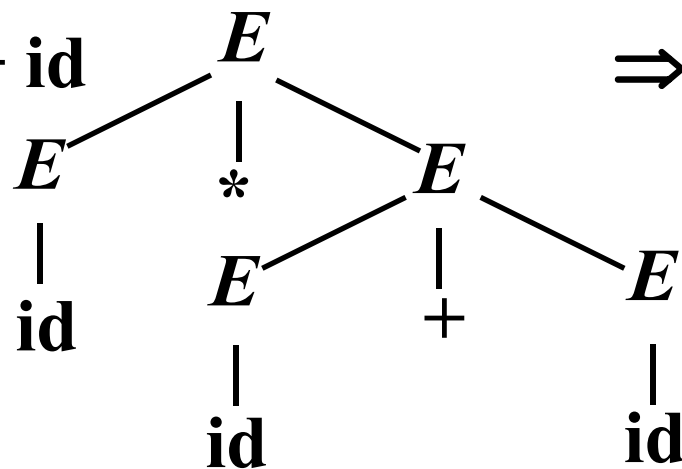
$E \Rightarrow E * E$

$\Rightarrow \text{id} * E$

$\Rightarrow \text{id} * E + E$

$\Rightarrow \text{id} * \text{id} + E$

$\Rightarrow \text{id} * \text{id} + \text{id}$



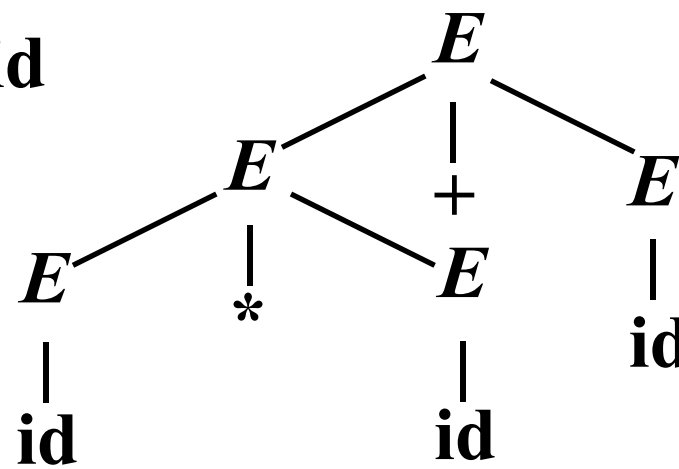
$E \Rightarrow E + E$

$\Rightarrow E * E + E$

$\Rightarrow \text{id} * E + E$

$\Rightarrow \text{id} * \text{id} + E$

$\Rightarrow \text{id} * \text{id} + \text{id}$





• 例  $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$

•  $\text{id} * \text{id} + \text{id}$  有两棵不同的分析树

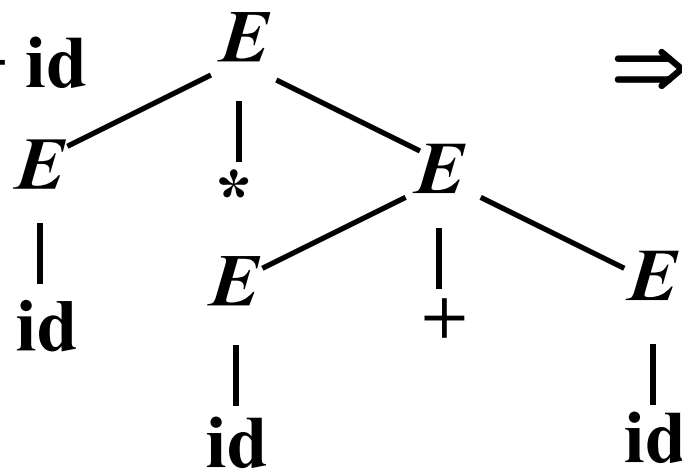
$E \Rightarrow E * E$

$\Rightarrow \text{id} * E$

$\Rightarrow \text{id} * E + E$

$\Rightarrow \text{id} * \text{id} + E$

$\Rightarrow \text{id} * \text{id} + \text{id}$



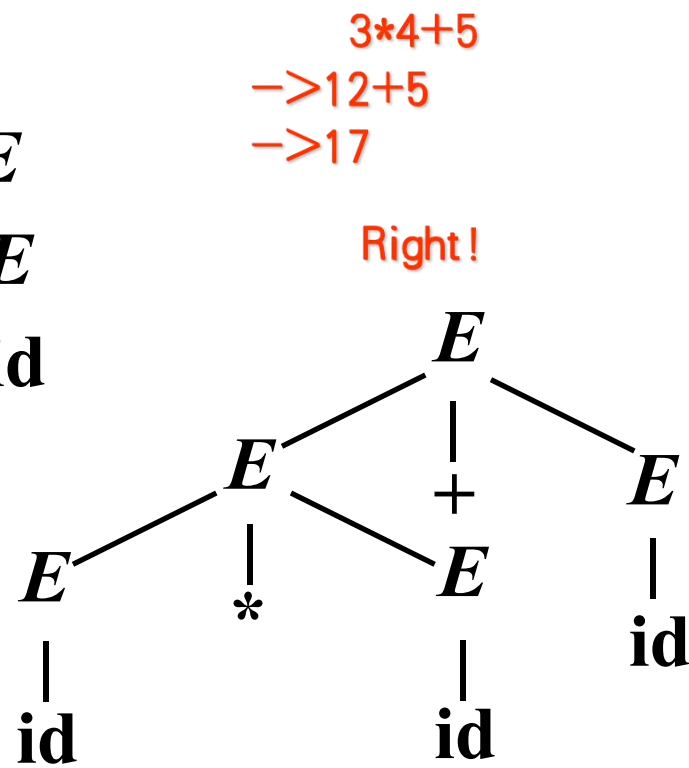
$E \Rightarrow E + E$

$\Rightarrow E * E + E$

$\Rightarrow \text{id} * E + E$

$\Rightarrow \text{id} * \text{id} + E$

$\Rightarrow \text{id} * \text{id} + \text{id}$



$3 * 4 + 5$   
 $\rightarrow 3 * 9$   
 $\rightarrow 27$

$3 * 4 + 5$   
 $\rightarrow 12 + 5$   
 $\rightarrow 17$

Wrong!

Right!



- 表达式产生二义性的原因

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \text{id}$$

+, \*操作都是左结合的, 并且在运算中有不同的优先级, 但是在这个文法中没有得到体现



- 表达式产生二义性的原因
- 没有一般性的方法，但，可通过**定义运算优先级和结合律**来消除二义性



- 用一种层次观点看待表达式

- id \* id \* (id+id) + id \* id + id
- id \* id \* (id+id)

$E \rightarrow E + E$   
从不同的E推导  
得到不同的树

根据算符不同的  
优先级，引入新  
的非终结符





- 用一种层次观点看待表达式

- id \* id \* (id+id) + id \* id + id
- id \* id \* (id+id)

- 新的非二义文法

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

$E \rightarrow E + E$   
从不同的E推导  
得到不同的树

根据算符不同的  
优先级，引入新  
的非终结符



- 用一种层次观点看待表达式

- id \* id \* (id+id) + id \* id + id
- id \* id \* (id+id)

- 新的非二义文法

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

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

$E \rightarrow E + E$   
从不同的E推导  
得到不同的树

根据算符不同的  
优先级，引入新  
的非终结符



- 用一种层次观点看待表达式

- id \* id \* (id+id) + id \* id + id
- id \* id \* (id+id)

- 新的非二义文法

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

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

$$F \rightarrow \text{id} \mid (E)$$

$E \rightarrow E + E$   
从不同的E推导  
得到不同的树

根据算符不同的  
优先级, 引入新  
的非终结符



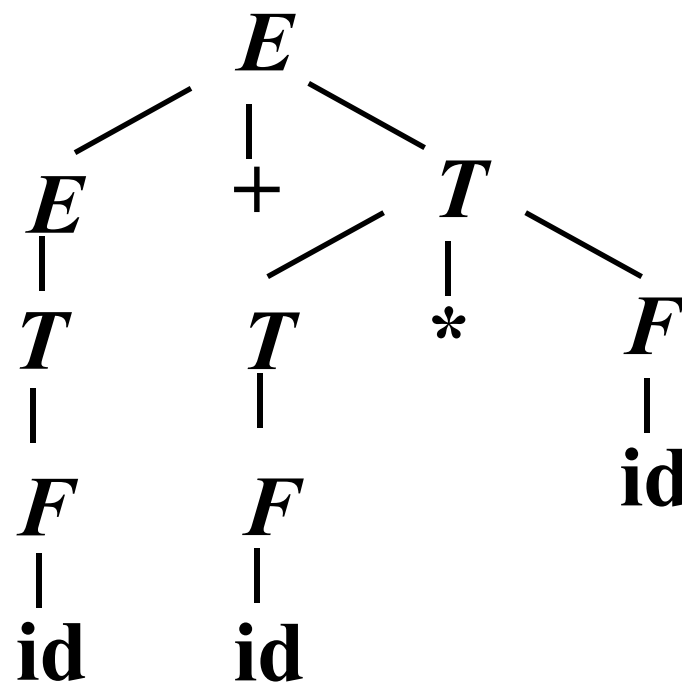
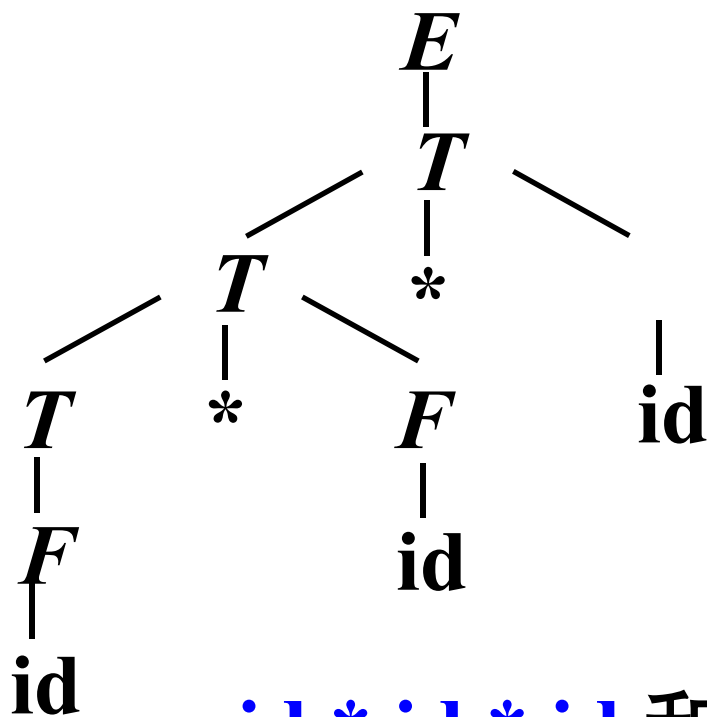
# 消除二义性



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

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

$$F \rightarrow \text{id} \mid (E)$$



$\text{id} * \text{id} * \text{id}$  和  $\text{id} + \text{id} * \text{id}$  的分析树



- 悬空else文法

*stmt*  $\rightarrow$  if *expr* then *stmt*  
          | if *expr* then *stmt* else *stmt*  
          | other

- 判断该文法有无二义性
- 如果存在二义性，如何消除



- 悬空else文法

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

- 句型: if *expr* then if *expr* then *stmt* else *stmt*



- 悬空else文法

*stmt*  $\rightarrow$  if *expr* then *stmt*  
          | if *expr* then *stmt* else *stmt*  
          | other

- 句型: if *expr* then if *expr* then *stmt* else *stmt*

- 两个最左推导:

*stmt*  $\Rightarrow$  if *expr* then *stmt*  
           $\Rightarrow$  if *expr* then if *expr* then *stmt* else *stmt*  
*stmt*  $\Rightarrow$  if *expr* then *stmt* else *stmt*  
           $\Rightarrow$  if *expr* then if *expr* then *stmt* else *stmt*



- 每个else与最近的尚未匹配的then匹配

$$stmt \rightarrow \begin{matrix} matched\_stmt \\ | \\ unmatched\_stmt \end{matrix}$$
$$\begin{array}{l} matched\_stmt \rightarrow \text{if } expr \text{ then } matched\_stmt \\ \qquad \qquad \qquad \text{else } matched\_stmt \\ | \text{ other} \end{array}$$

*unmatched\_stmt* → if *expr* then *stmt*  
| if *expr* **then** *matched\_stmt*  
**else** *unmatched\_stmt*





- 上下文无关文法的优点

- 文法给出了精确的，易于理解的语法说明
- 自动产生高效的分析器
- 可以给语言定义出层次结构
- 以文法为基础的语言的实现便于语言的修改

- 上下文无关文法的缺点

- 文法只能描述编程语言的大部分语法



- **为什么要用正则表达式定义词法**

- 词法规则非常简单，不必用上下文无关文法。
- 对于词法记号，正则表达式描述简洁且易于理解。
- 从正则表达式构造出的词法分析器效率高。



- 为什么要用正则表达式定义词法

- 词法规则非常简单，不必用上下文无关文法。
- 对于词法记号，正则表达式描述简洁且易于理解。
- 从正则表达式构造出的词法分析器效率高。

- 分离词法分析和语法分析的好处（**软件工程视角**）

- 简化设计
- 编译器的效率会改进
- 编译器的可移植性加强
- 便于编译器前端的模块划分

# 一起努力 打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年03月06日