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

2024年09月11日

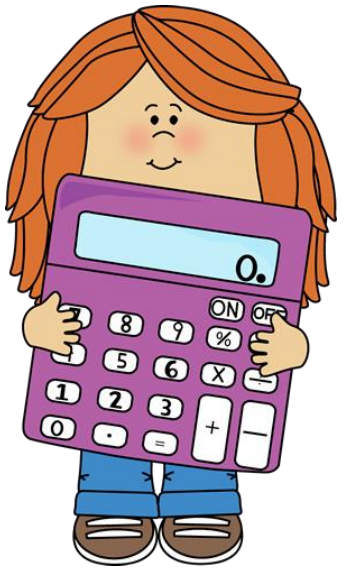
简单计算器程序

$1 + (2 - 3)$ 合法

$1 + 2 - 3 +$ 非法

$1\ 2 + - 3$ 非法

$1 + 2 - a$ 非法

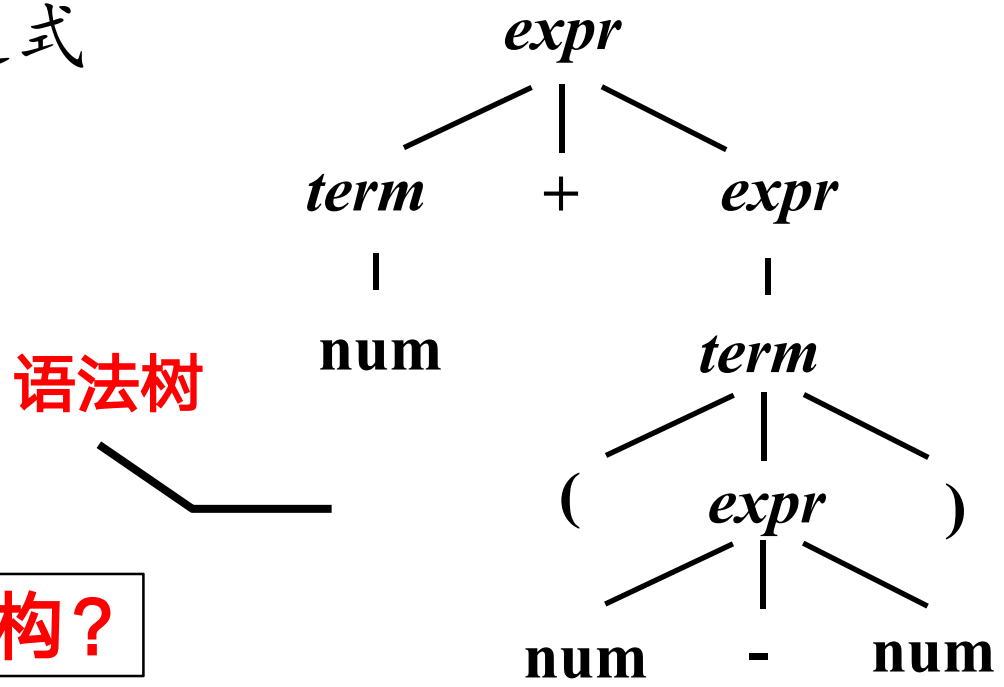


语法分析的目的是教会计算机判断输入合法性

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)

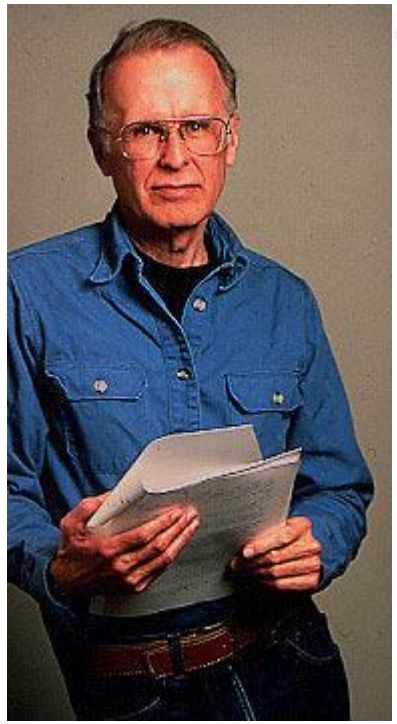


语法树

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——描述语言的语法结构

- 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——描述语言的语法结构

- 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，$V_T \cap V_N = \varnothing$
- 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：$V_N = \{ expr, term \}$

上下文无关文法——描述语言的语法结构

- 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：$S = expr$

上下文无关文法——描述语言的语法结构

- 上下文无关文法的形式化定义：

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

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

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

举例

- 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)

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

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

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

问题二：给定文法，如何判定输入串属于文法规定的语言呢？

图形化演示构造过程

- 例：对于文法 *expr* → *term* | *term* + *expr* | *term* - *expr*

  term → num | (*expr*)

  - 展示1 + (2 – 3) 的构造过程

expr

expr

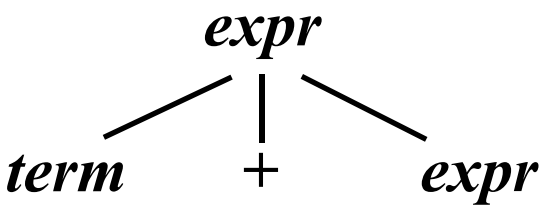
图形化演示构造过程

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

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

- 展示1 + (2 – 3) 的构造过程

$$expr \Rightarrow term + expr$$



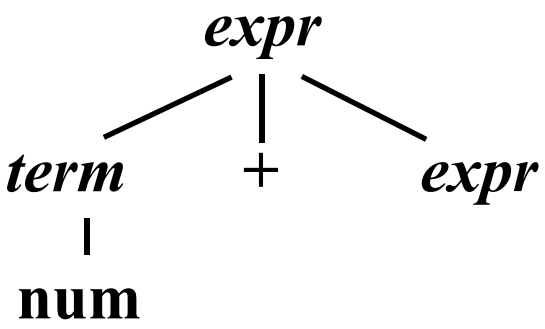
图形化演示构造过程

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

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

- 展示1 + (2 – 3) 的构造过程

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



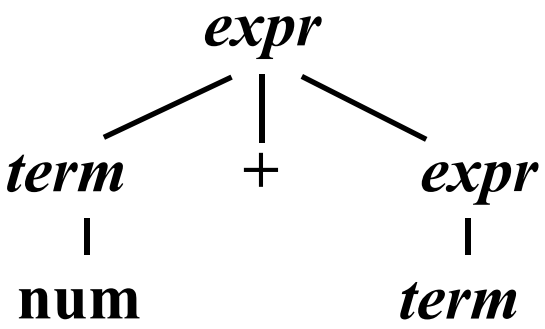
图形化演示构造过程

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

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

- 展示1 + (2 – 3) 的构造过程

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



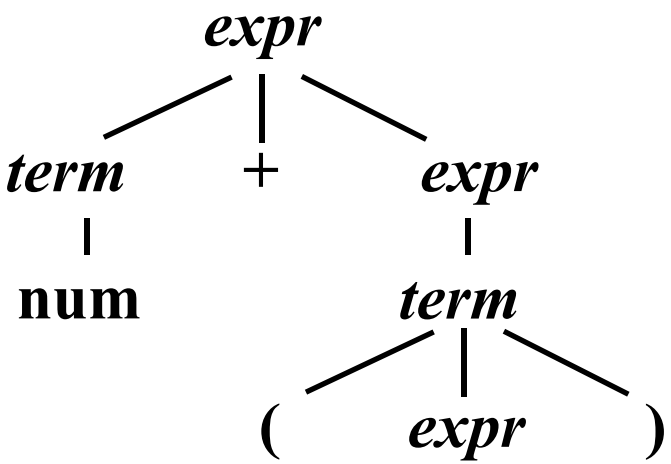
图形化演示构造过程

- 例：对于文法 *expr* → *term* | *term* + *expr* | *term* - *expr*

  term → num | (*expr*)

  - 展示1 + (2 – 3) 的构造过程

expr ⇒ *term* + *expr*
⇒ num + *expr*
⇒ num + *term*
⇒ num + (*expr*)



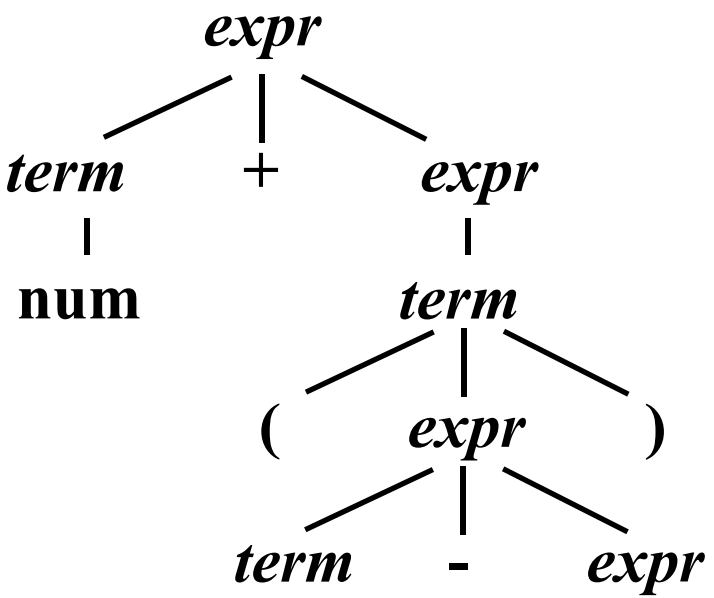
图形化演示构造过程

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

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

- 展示1 + (2 – 3) 的构造过程

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



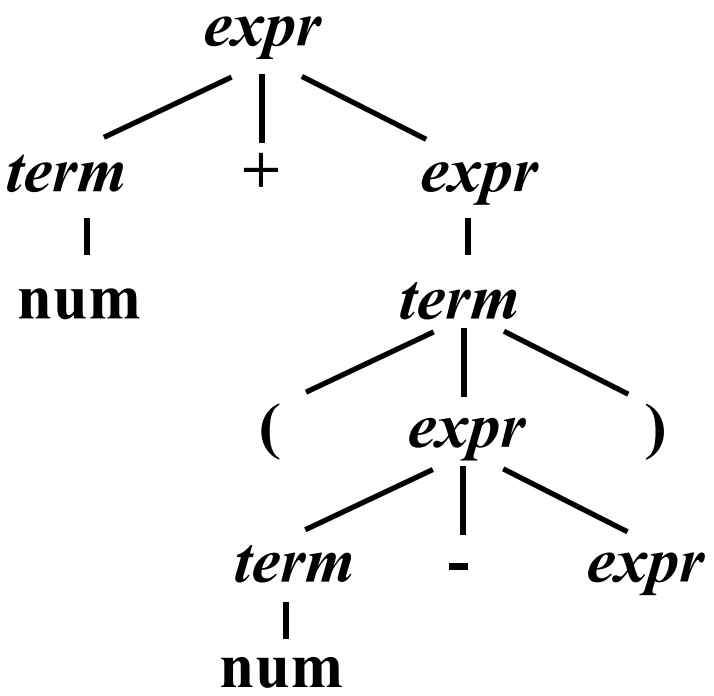
图形化演示构造过程

- 例：对于文法 *expr* → *term* | *term* + *expr* | *term* - *expr*

  term → **num** | (*expr*)

  - 展示1 + (2 – 3) 的构造过程

expr ⇒ *term* + *expr*
⇒ **num** + *expr*
⇒ **num** + *term*
⇒ **num** + (*expr*)
⇒ **num** + (*term* – *expr*)
⇒ **num** + (**num** – *expr*)



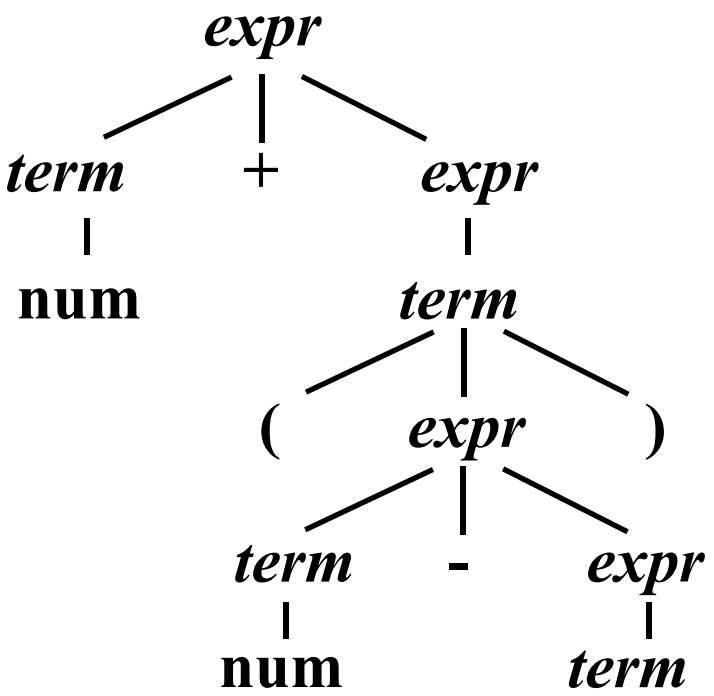
图形化演示构造过程

- 例：对于文法 *expr* → *term* | *term* + *expr* | *term* - *expr*

term → num | (*expr*)

- 展示1 + (2 – 3) 的构造过程

expr ⇒ *term* + *expr*
⇒ num + *expr*
⇒ num + *term*
⇒ num + (*expr*)
⇒ num + (*term* – *expr*)
⇒ num + (num – *expr*)
⇒ num + (num – *term*)



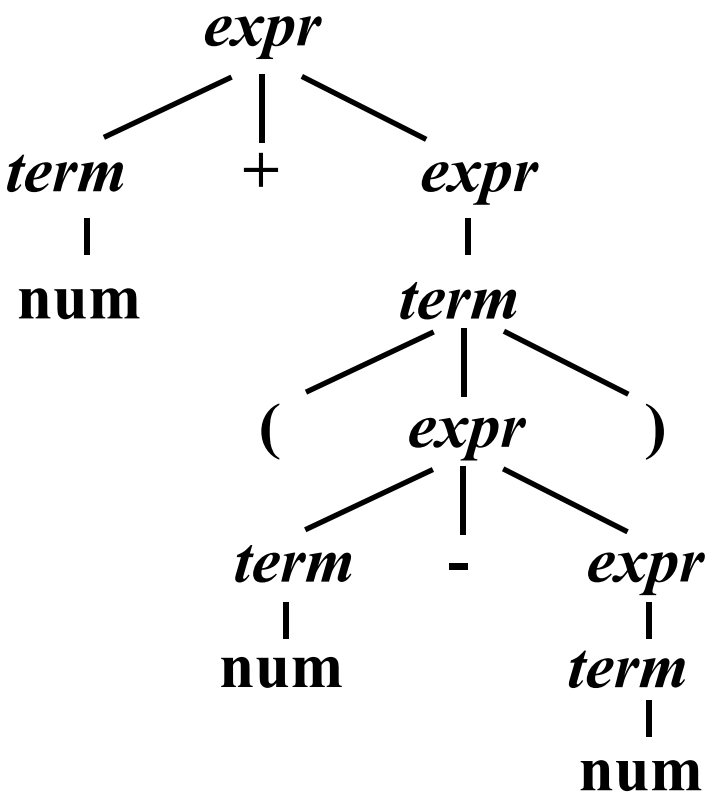
图形化演示构造过程

- 例：对于文法 *expr* → *term* | *term* + *expr* | *term* - *expr*

  term → **num** | (*expr*)

  - 展示1 + (2 – 3) 的构造过程

expr ⇒ *term* + *expr*
⇒ num + *expr*
⇒ num + *term*
⇒ num + (*expr*)
⇒ num + (*term* – *expr*)
⇒ num + (num – *expr*)
⇒ num + (num – *term*)
⇒ num + (num – num)



上下文无关文法的推导

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

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

- **记法：**
  - $S \Rightarrow^* \alpha$：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 $E \to E+E \mid E*E \mid (E) \mid -E \mid \mathrm{id}$

- **最左推导**（leftmost derivation）
  - 每步代换最左边的非终结符

$$E \Rightarrow_{lm} -E \Rightarrow_{lm} -(E) \Rightarrow_{lm} -(E+E)$$
$$\Rightarrow_{lm} -(\mathrm{id}+E) \Rightarrow_{lm} -(\mathrm{id}+\mathrm{id})$$

- **最右推导**（rightmost or canonical derivation，规范推导）
  - 每步代换最右边的非终结符

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

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γ仅依据A的产生式推导，而无需依赖A的上下文α和β

语言、文法、句型、句子

- **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)：$S \Rightarrow^* \alpha$，S是开始符号，α是由终结符和/或非终结符组成的串，则α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 $E \to E+E \mid E*E \mid (E) \mid -E \mid \mathrm{id}$

褐红色标出的均是句型

- **最左推导**（leftmost derivation）
  - 每步代换最左边的非终结符

$$E \Rightarrow_{lm} -E \Rightarrow_{lm} -(E) \Rightarrow_{lm} -(E+E)$$
$$\Rightarrow_{lm} -(\mathrm{id}+E) \Rightarrow_{lm} -(\mathrm{id}+\mathrm{id})$$

- **最右推导**（rightmost or canonical derivation,规范推导）
  - 每步代换最右边的非终结符

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

最左推导和最右推导

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

- **最左推导**（leftmost derivation）
  - 每步代换最左边的非终结符

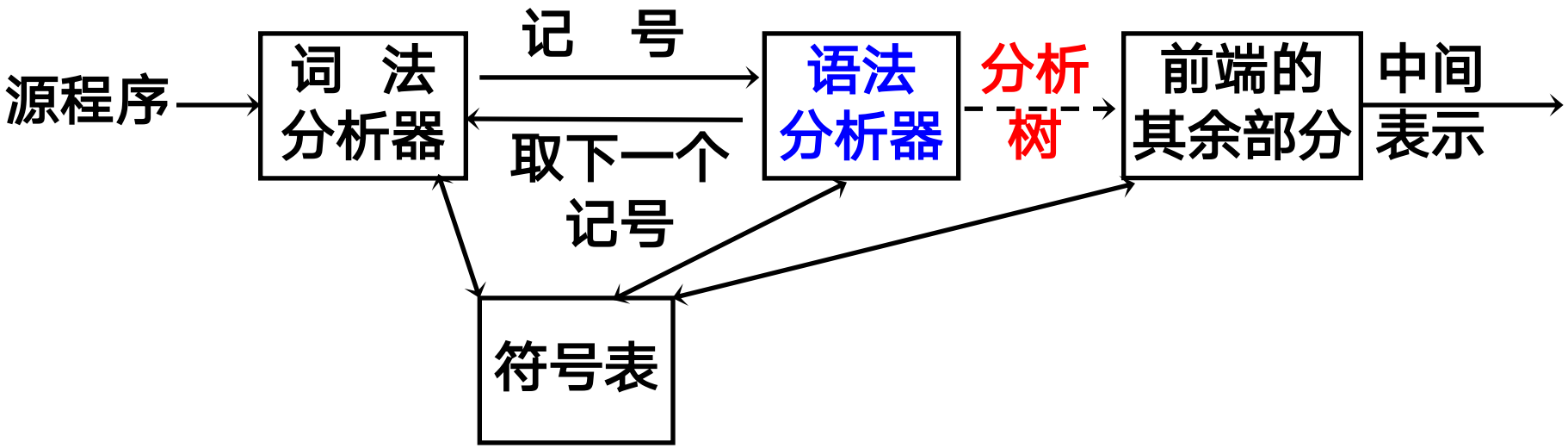
$$E \Rightarrow_{lm} -E \Rightarrow_{lm} -(E) \Rightarrow_{lm} -(E+E)$$
$$\Rightarrow_{lm} -(\mathrm{id}+E) \Rightarrow_{lm} -(\mathrm{id}+\mathrm{id})$$

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

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

主要内容



- 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

正则表达式的局限

- **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：$\{wcw \mid w$是a和b的串$\}$

正则表达式的局限

- **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：$\{wcw \mid w$是a和b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

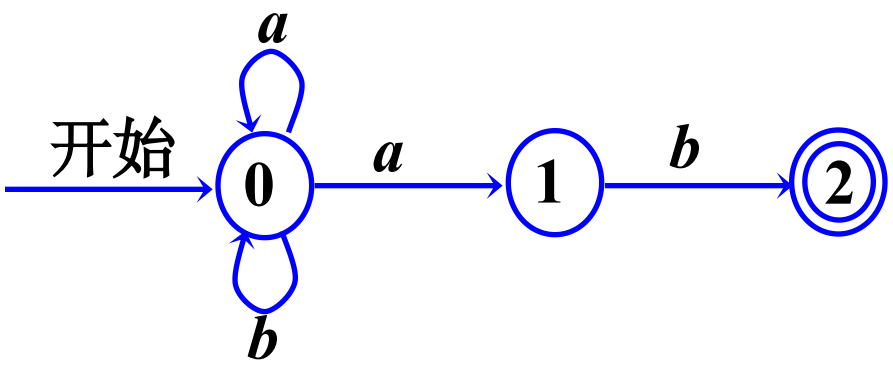
思考题

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

正则表达式与CFG的区别

- 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

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

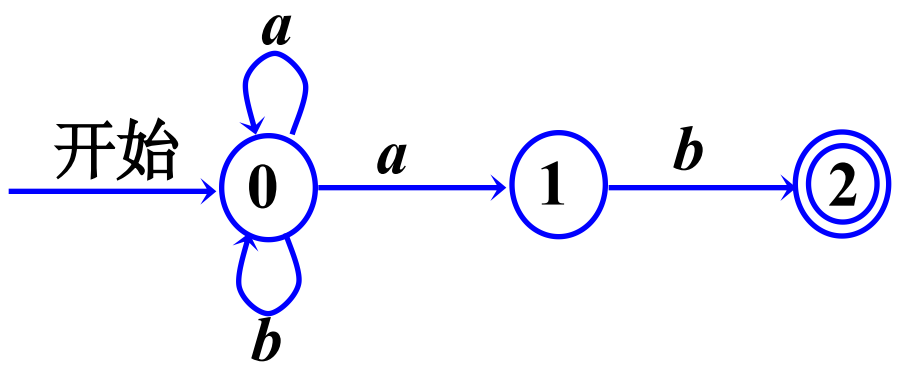
$A_1 \rightarrow b\,A_2$

$A_2 \rightarrow \varepsilon$

正则表达式与CFG的区别

- NFA →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Ai
  - 如果状态i有一个a转换到状态j,引入产生式Ai→aAj,如果i是接受状态，则引入Ai→ ε

$$A_0 \rightarrow a\,A_0 \mid b\,A_0 \mid a\,A_1$$
$$A_1 \rightarrow b\,A_2$$
$$A_2 \rightarrow \varepsilon$$



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**
  - $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

文法的二义性

- 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。

文法的二义性

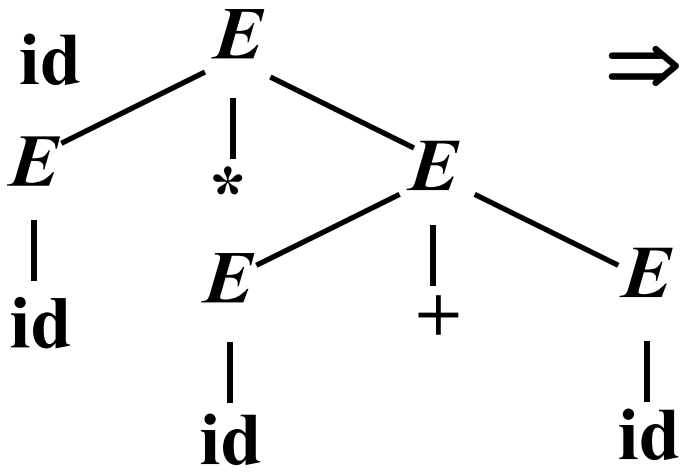
- 例 $E \to E + E \mid E * E \mid (E) \mid - E \mid \mathbf{id}$
  - id * id + id有两个不同的最左推导

$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow \mathbf{id} * E \\
&\Rightarrow \mathbf{id} * E + E \\
&\Rightarrow \mathbf{id} * \mathbf{id} + E \\
&\Rightarrow \mathbf{id} * \mathbf{id} + \mathbf{id}
\end{aligned}
\qquad
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow E * E + E \\
&\Rightarrow \mathbf{id} * E + E \\
&\Rightarrow \mathbf{id} * \mathbf{id} + E \\
&\Rightarrow \mathbf{id} * \mathbf{id} + \mathbf{id}
\end{aligned}
$$

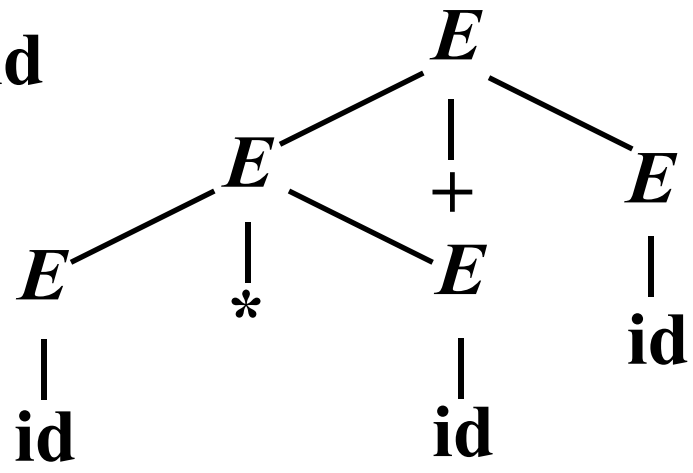
文法的二义性

- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathbf{id}$
  - id * id + id有两棵不同的分析树

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



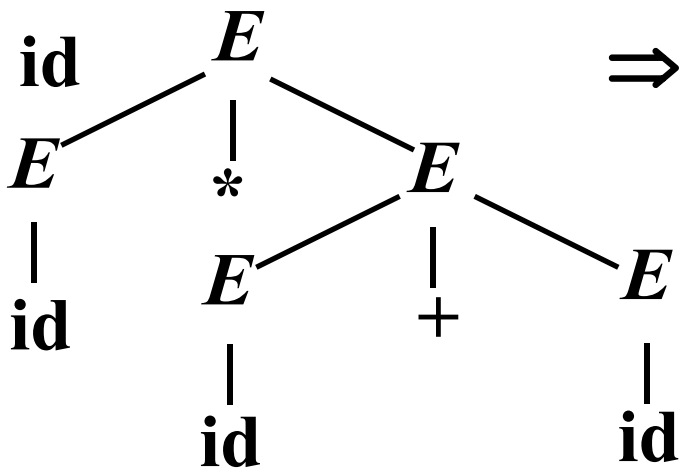
$E \Rightarrow E + E$
$\Rightarrow E * E + E$
$\Rightarrow \mathbf{id} * E + E$
$\Rightarrow \mathbf{id} * \mathbf{id} + E$
$\Rightarrow \mathbf{id} * \mathbf{id} + \mathbf{id}$



文法的二义性

- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathrm{id}$
  - id * id + id有两棵不同的分析树

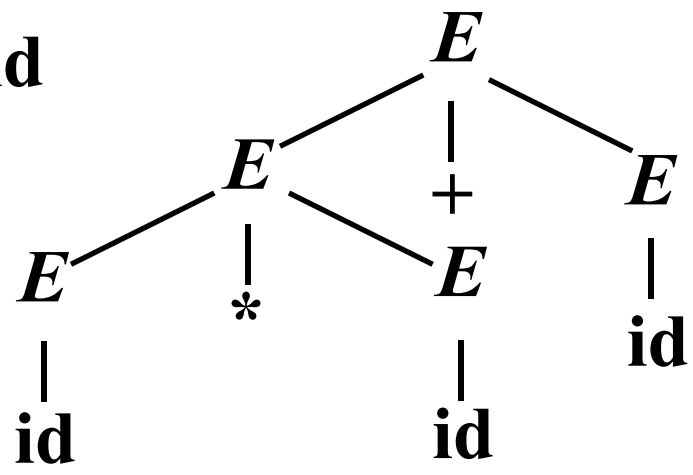
$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow \mathrm{id} * E \\
&\Rightarrow \mathrm{id} * E + E \\
&\Rightarrow \mathrm{id} * \mathrm{id} + E \\
&\Rightarrow \mathrm{id} * \mathrm{id} + \mathrm{id}
\end{aligned}
$$



3*4+5
->3*9
->27
Wrong!

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

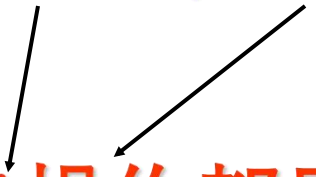
3*4+5
->12+5
->17
Right!



消除二义性

- 表达式产生二义性的原因

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



+，*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

消除二义性

- 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

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

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)
- 新的非二义文法

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

E → 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

- 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

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)
- 新的非二义文法

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

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

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

E → E + E
从不同的E推导得到不同的树

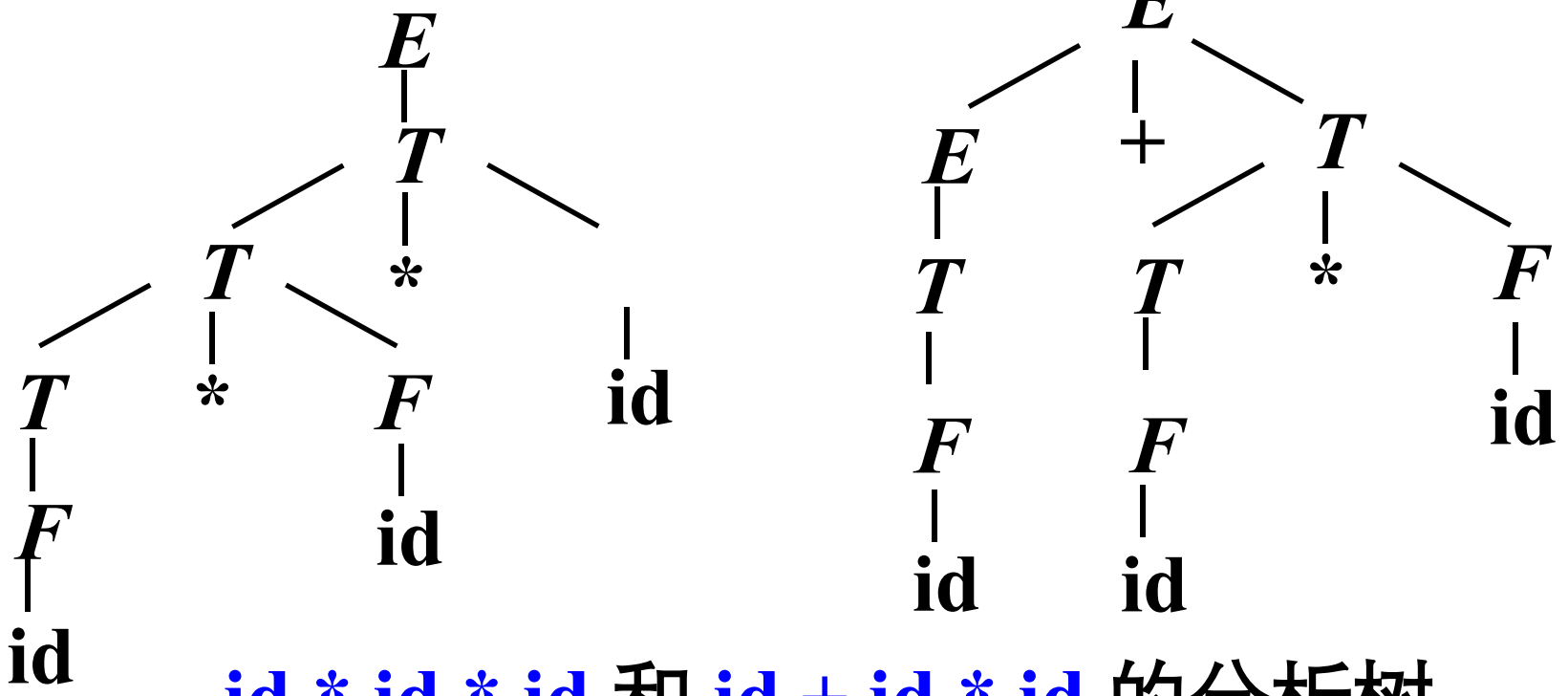
根据算符不同的优先级，引入新的非终结符

消除二义性

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

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



id * id * id 和 id + id * id 的分析树

思考题

- 悬空else文法

stmt → 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

思考题

- 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*

思考题

- 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*

- 两个最左推导：

stmt ⇒ if *expr* then *stmt*
⇒ if *expr* then if *expr* then *stmt* else *stmt*

stmt ⇒ if *expr* then *stmt* else *stmt*
⇒ if *expr* then if *expr* then *stmt* else *stmt*

消除二义性

- 无二义的文法
  - 每个else与最近的尚未匹配的then匹配

stmt → *matched _stmt*
| *unmatched_stmt*

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

unmatched_stmt → if *expr* then *stmt*
| if *expr* then *matched_stmt*
else *unmatched_stmt*

语言与文法

- **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

分离词法分析器的理由

- 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。

分离词法分析器的理由

- 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

2024年09月11日