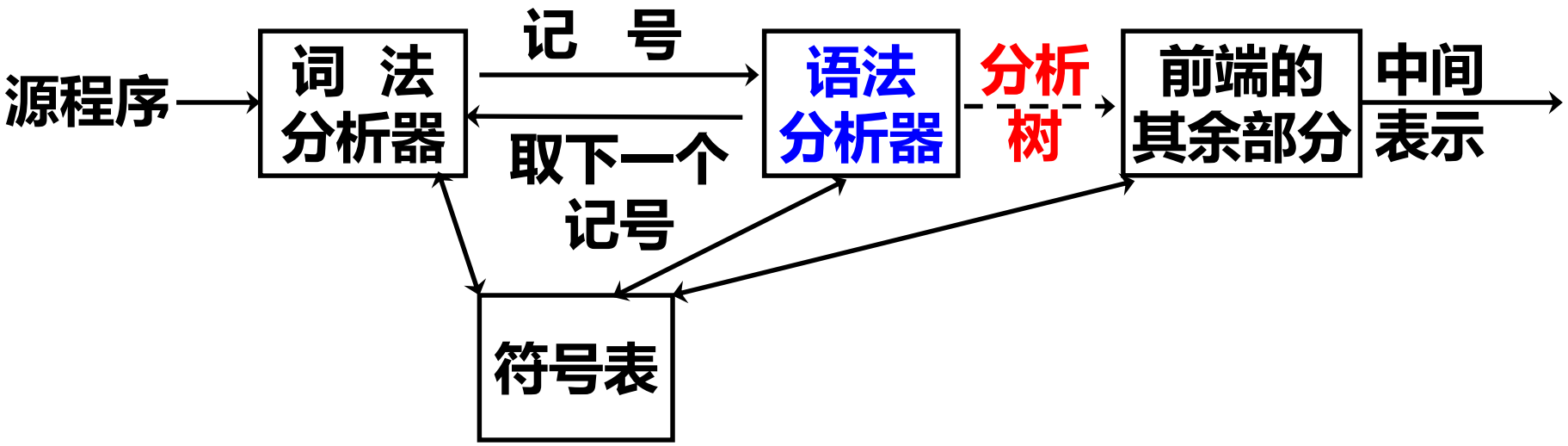
中国科学技术大学
University of Science and Technology of China

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
语法分析 I

计算机科学与技术学院
李 诚
17/09/2018

主要内容



- 正则表达式的局限
- 语法分析器简介
- 上下文无关文法
  - 定义、推导、二义性
  - 消除二义性

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* | *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：$a\,(ba)^5$, $a\,(ba)^*$

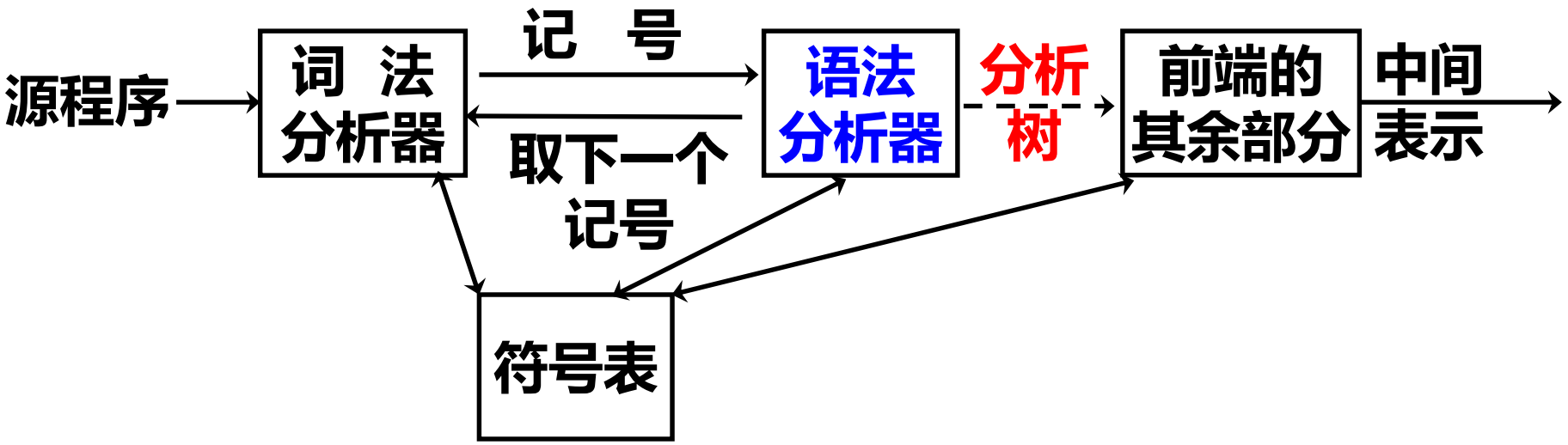
- ❖不能用于描述配对或嵌套的结构

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

  例2：{wcw | 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

主要内容



- 正则表达式的局限
- 语法分析器简介
- 上下文无关文法
  - 定义、推导、二义性
  - 消除二义性

语法分析器简介

- 输入：从词法分析器中获得的记号序列
- 输出：程序的语法树（syntax or parse tree）
  - 语法树表示了源程序的层次化语法结构
  - 语法树是一种中间代码形式

❑COOL 语言

if x = y then 1 else 2 fi

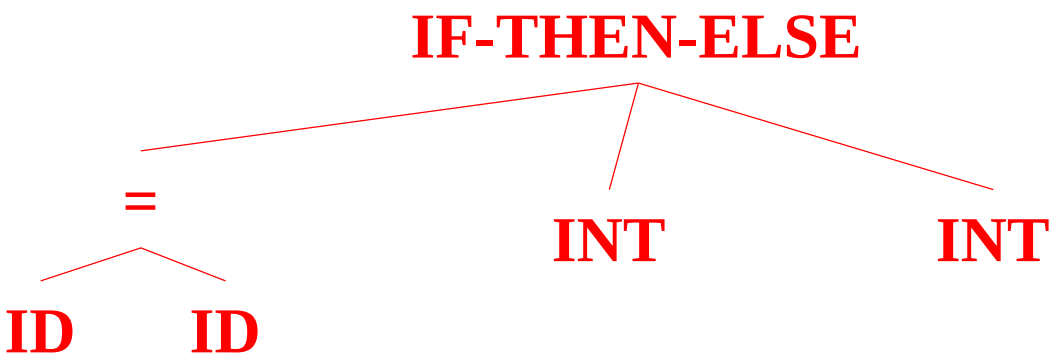
- COOL 语言

  if x = y then 1 else 2 fi

- 语法分析器的输入

  IF ID = ID THEN INT ELSE INT FI

- 语法分析器的输出



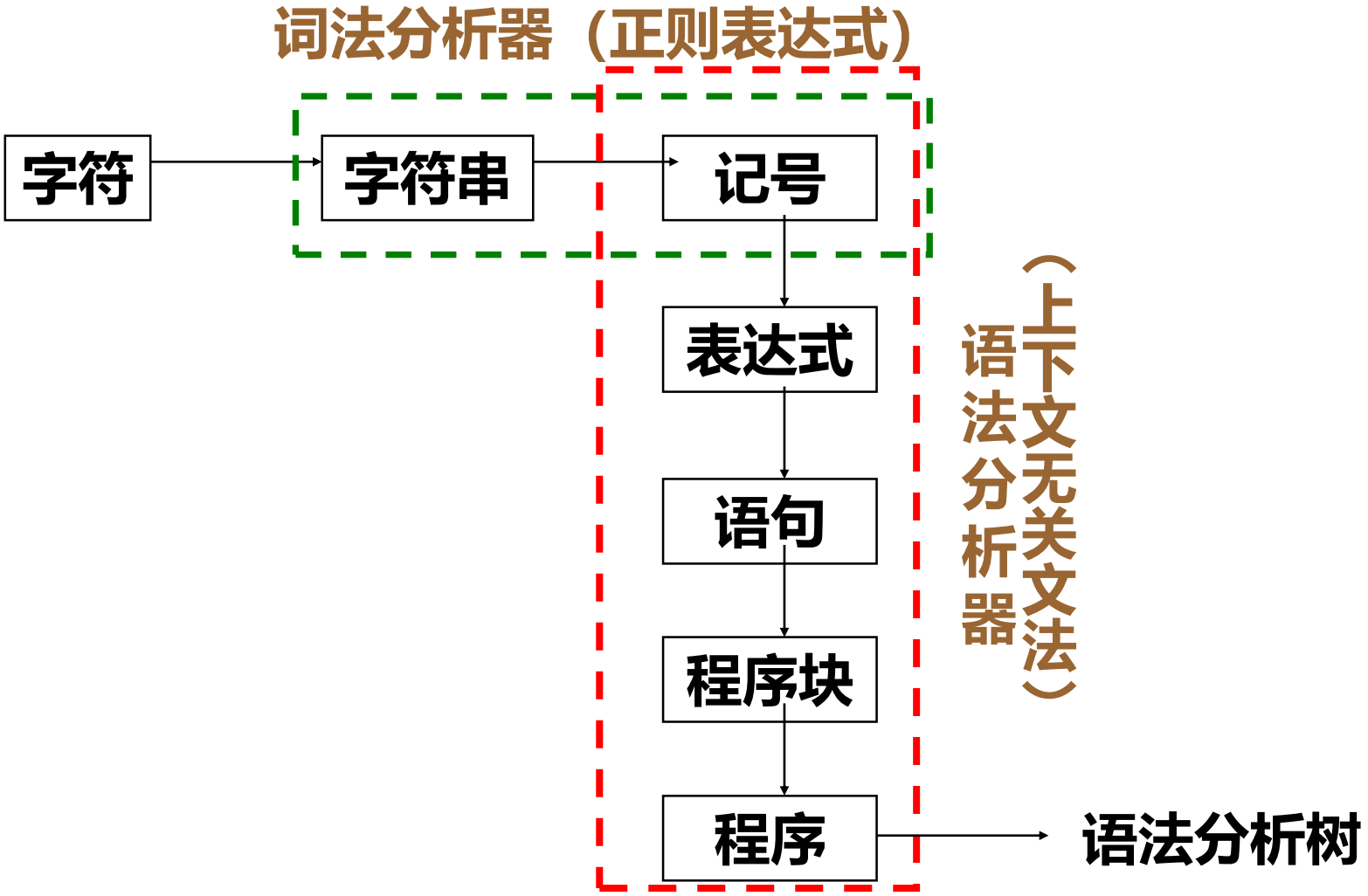
语法分析器简介

- 不是所有的记号序列都是合法(valid)的
- 语法分析器需要区分合法和非法的记号序列

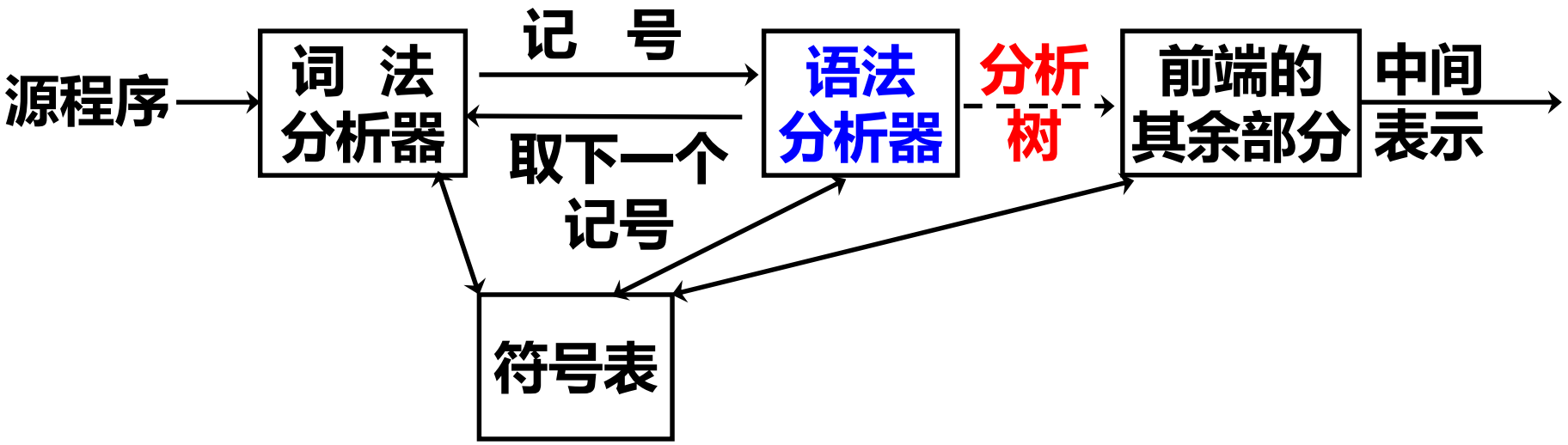
- 不是所有的记号序列都是合法(valid)的
- 语法分析器需要区分合法和非法的记号序列

- 因此，我们需要：
  - 一种可以描述合法记号序列的语言
  - 一种可以区分合法和非法的记号序列的方法

词法分析器与语法分析器



主要内容



- 正则表达式的局限
- 语法分析器简介
- **上下文无关文法**
  - 定义、推导、二义性
  - 消除二义性

上下文无关文法

- 上下文无关文法（Context-free Grammar，或CFG）是四元组（V_T, V_N, S, P）
  - V_T：终结符集合（终结符↔记号名）
  - V_N：非终结符集合（非空有限集合，$V_T \cap V_N = \phi$ ）
  - S：开始符号，非终结符中的一个
  - P：产生式集合
    - 产生式形式：$A \to \alpha$，$A \in V_N, \alpha \in (V_T \cup V_N)^*$

上下文无关文法

- 上下文无关文法（Context-free Grammar，或CFG）是四元组（V_T, V_N, S, P）
  - V_T：终结符集合（终结符↔记号名）
  - V_N：非终结符集合（非空有限集合，$V_T \cap V_N = \phi$ ）
  - S： 开始符号，非终结符中的一个
  - P： 产生式集合
    - 产生式形式：$A \to \alpha$， $A \in V_N, \alpha \in (V_T \cup V_N)^*$
- 例 ({id, +, *, –, (,)}, {*expr*, *op*}, *expr*, *P*)
  - *expr* → *expr op expr*
  - *expr* → (*expr*)
  - *expr* → – *expr*
  - *expr* → id
  - *op* → +
  - *op* → *

CFG-简化表示

- 简化表示：引入选择运算符

$$expr \rightarrow expr\ op\ expr \mid (expr) \mid - expr \mid \mathbf{id}$$

$$op \rightarrow + \mid *$$

- 简化表示

$$E \rightarrow E\ A\ E \mid (E\ ) \mid -E \mid \mathbf{id}$$

$$A \rightarrow + \mid *$$

思考题

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

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

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

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

❑ 记法:

- ❖$S \Rightarrow^{*} \alpha$: 0步或多步推导
- ❖$S \Rightarrow^{+} w$: 1步或多步推导

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

❑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

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

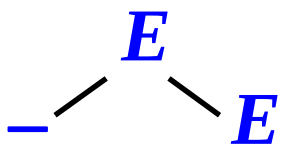
分析树(parse or syntax tree)

中国科学技术大学
University of Science and Technology of China

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathbf{id}$

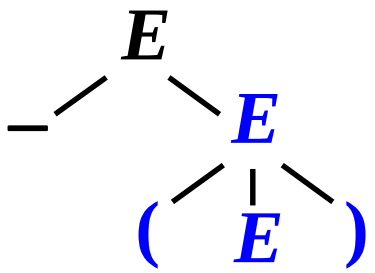
分析树(parse or syntax tree)

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathrm{id}$
  - -(id+id)最左推导的分析树



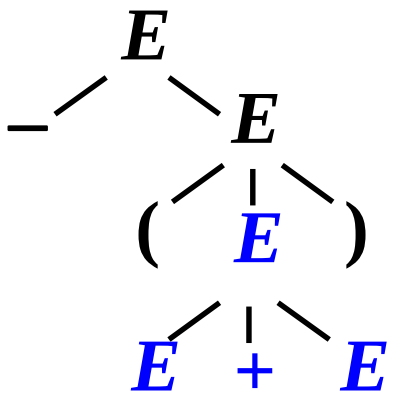
分析树(parse or syntax tree)

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathrm{id}$
  - -(id+id)最左推导的分析树



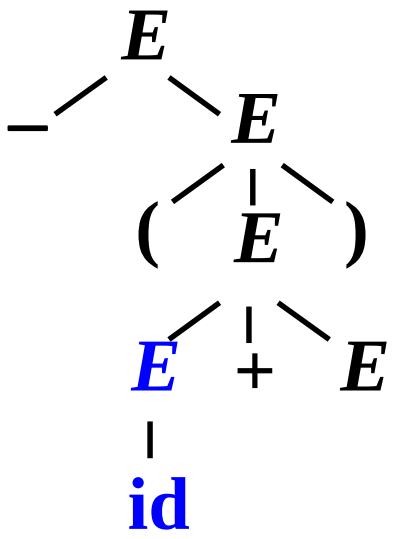
分析树(parse or syntax tree)

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathrm{id}$
  - -(id+id)最左推导的分析树



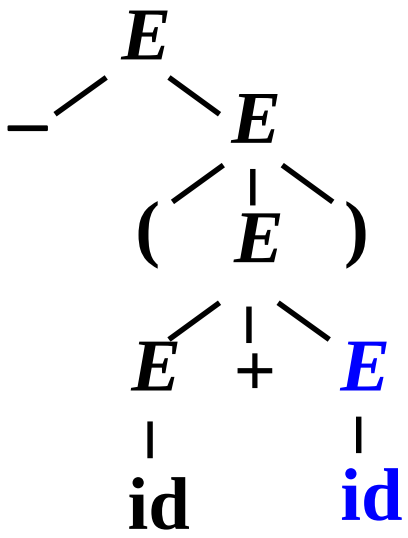
分析树(parse or syntax tree)

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathbf{id}$
  - -(id+id)最左推导的分析树

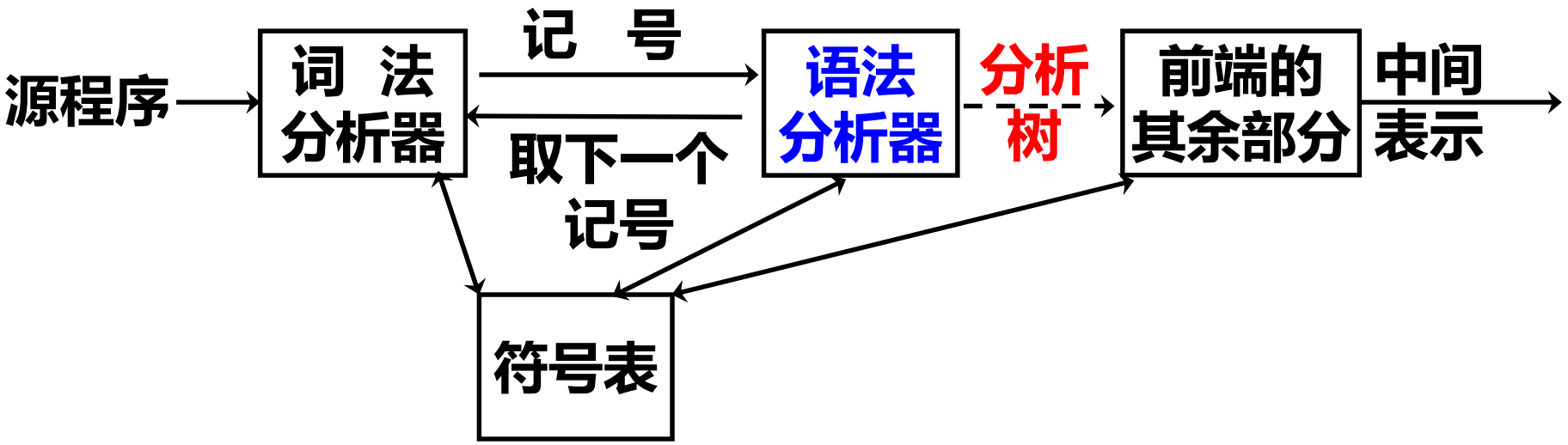


分析树(parse or syntax tree)

- 语法分析树是推导的图形表示形式
- 例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathbf{id}$
  - -(id+id)最左推导的分析树



主要内容



- 正则表达式的局限
- 语法分析器简介
- **上下文无关文法**
  - 定义、推导、二义性
  - 消除二义性

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

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

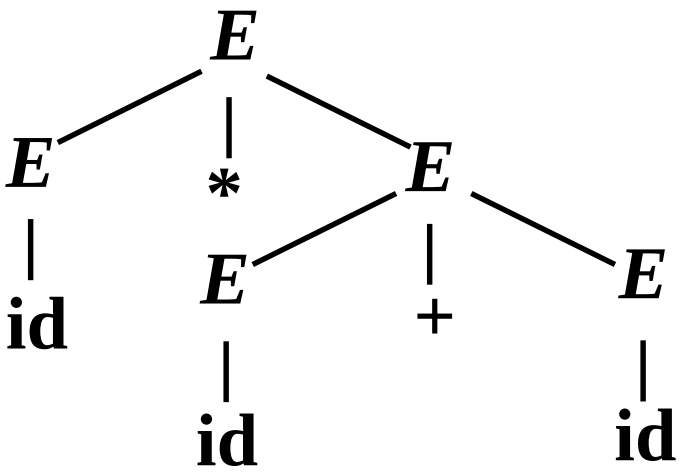
文法的二义性

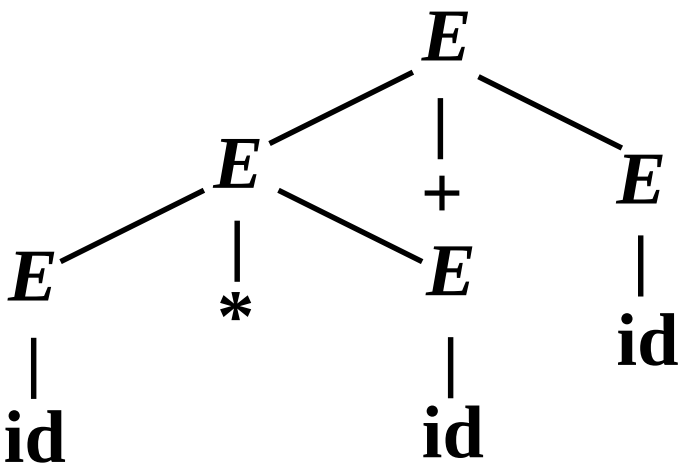
❑例 $E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid \mathrm{id}$

❖id * id + id有两棵不同的分析树

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

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





文法的二义性

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

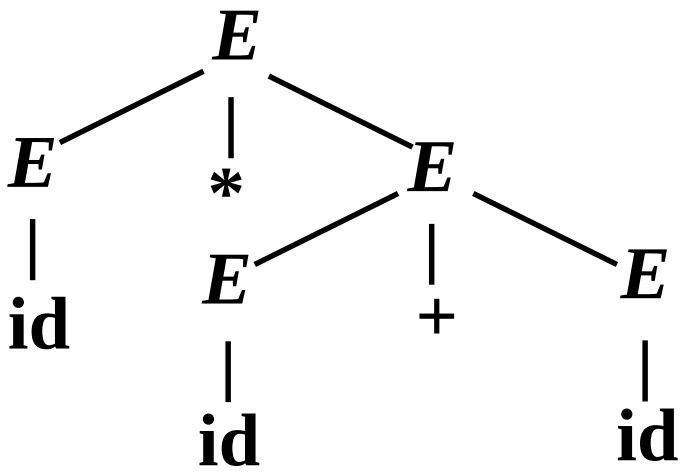
❖id * id + id有两棵不同的分析树

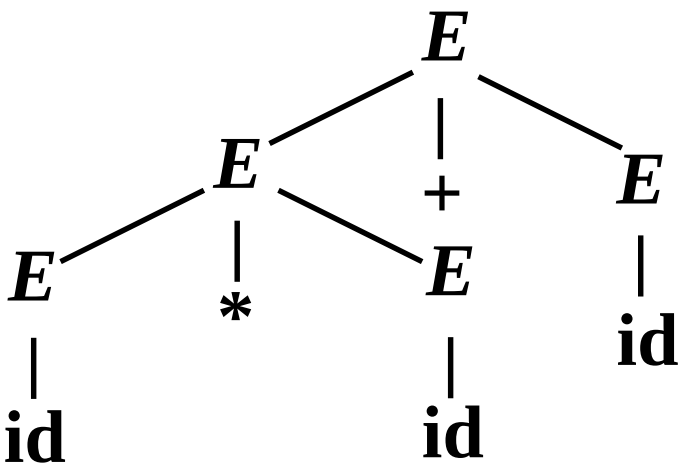
$E \Rightarrow E * E$
$\Rightarrow \text{id} * E$
$\Rightarrow \text{id} * E + E$
$\Rightarrow \text{id} * \text{id} + E$
$\Rightarrow \text{id} * \text{id} + \text{id}$

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

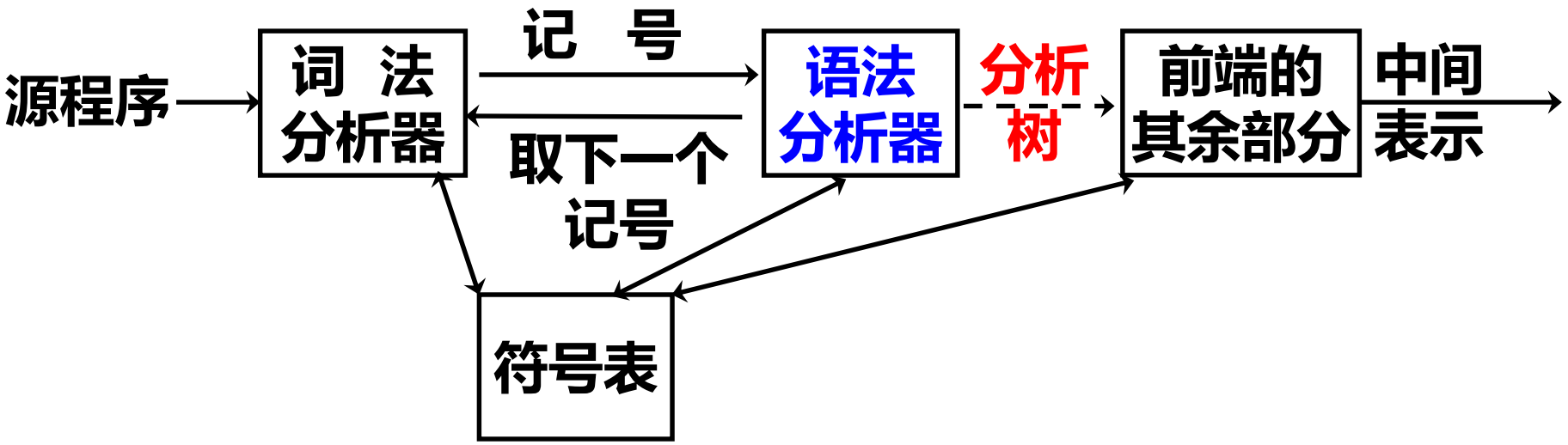
$E \Rightarrow E + E$
$\Rightarrow E * E + E$
$\Rightarrow \text{id} * E + E$
$\Rightarrow \text{id} * \text{id} + E$
$\Rightarrow \text{id} * \text{id} + \text{id}$

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





主要内容



- 正则表达式的局限
- 语法分析器简介
- **上下文无关文法**
  - 定义、推导、二义性
  - **消除二义性**

❑表达式产生二义性的原因

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

+，*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

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

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

$E \rightarrow E + E$
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

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

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

$E \rightarrow E + E$ 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

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

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

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

$E \rightarrow E + E$
从不同的E推导得到不同的树

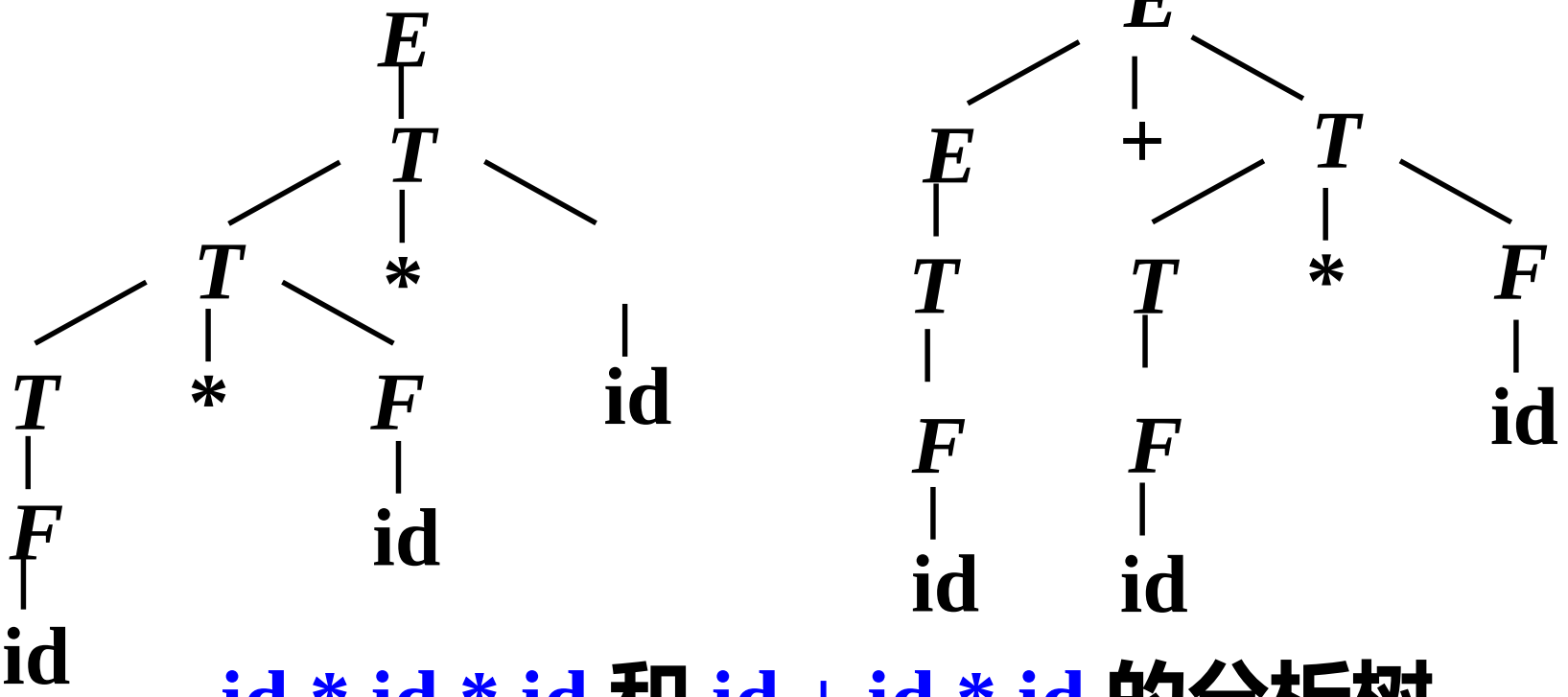
根据算符不同的优先级，引入新的非终结符

消除二义性

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

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



id * id * id 和 id + id * id 的分析树

❑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

❑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*

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*

❑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

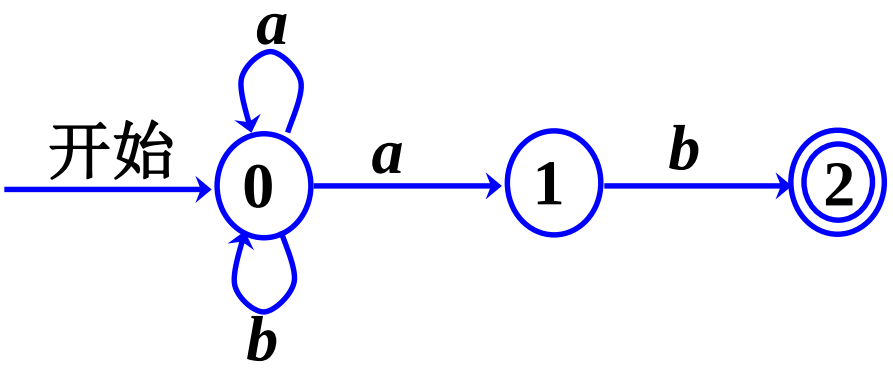
❑上下文无关文法的缺点

- ❖文法只能描述编程语言的大部分语法

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

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

$A_1 \rightarrow b\,A_2$

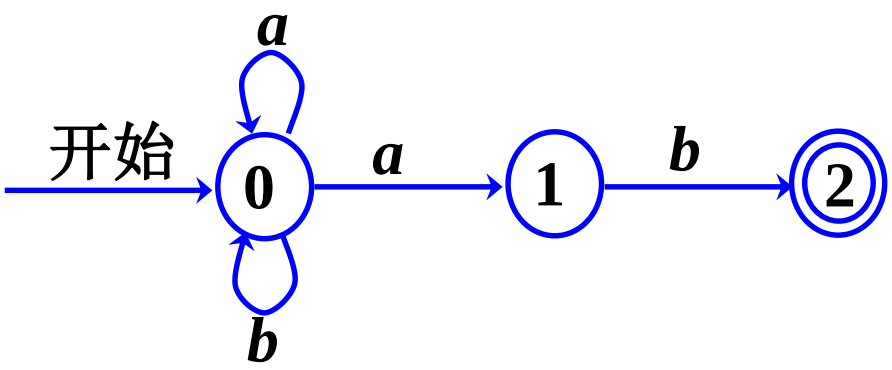
$A_2 \rightarrow \varepsilon$

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

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



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

上下文有关文法

- 上下文有关文法(context-sensitive grammar，或CSG)是一种形式文法，其中任何产生式规则的**左手端和右手端都可以被终结符和非终结符构成的上下文**所围绕。

- 例 $L_3 = \{a^n b^n c^n \mid n \geq 1\}$的上下文有关文法

$S \rightarrow aSBC$　　$S \rightarrow aBC$　　$CB \rightarrow BC$

$aB \rightarrow ab$　　$bB \rightarrow bb$　　$bC \rightarrow bc$

$cC \rightarrow cc$

上下文有关文法

❑例 $L_3 = \{a^n b^n c^n \mid n \geq 1\}$的上下文有关文法

$S \rightarrow aSBC$　　$S \rightarrow aBC$　　$CB \rightarrow BC$

$aB \rightarrow ab$　　$bB \rightarrow bb$　　$bC \rightarrow bc$

$cC \rightarrow cc$

$a^n b^n c^n$的推导过程如下：

❑例 $L_3 = \{a^n b^n c^n \mid n \geq 1\}$的上下文有关文法

$S \to aSBC$　　$S \to aBC$　　$CB \to BC$

$aB \to ab$　　$bB \to bb$　　$bC \to bc$

$cC \to cc$

$a^n b^n c^n$的推导过程如下：

$S \Rightarrow^* a^{n-1}S(BC)^{n-1}$	用$S \to aSBC$　n-1次
$S \Rightarrow^+ a^n(BC)^n$	用$S \to aBC$　1次
$S \Rightarrow^+ a^n B^n C^n$	用$CB \to BC$　交换相邻的CB
$S \Rightarrow^+ a^n bB^{n-1}C^n$	用$aB \to ab$　1次
$S \Rightarrow^+ a^n b^n C^n$	用$bB \to bb$　n-1次
$S \Rightarrow^+ a^n b^n cC^{n-1}$	用$bC \to bc$　1次
$S \Rightarrow^+ a^n b^n c^n$	用$cC \to cc$　n-1次

《编译原理与技术》
语法分析 I

Most of the difference between people that succeed and people that don't is the people don't give up!

—— Steven Jobs