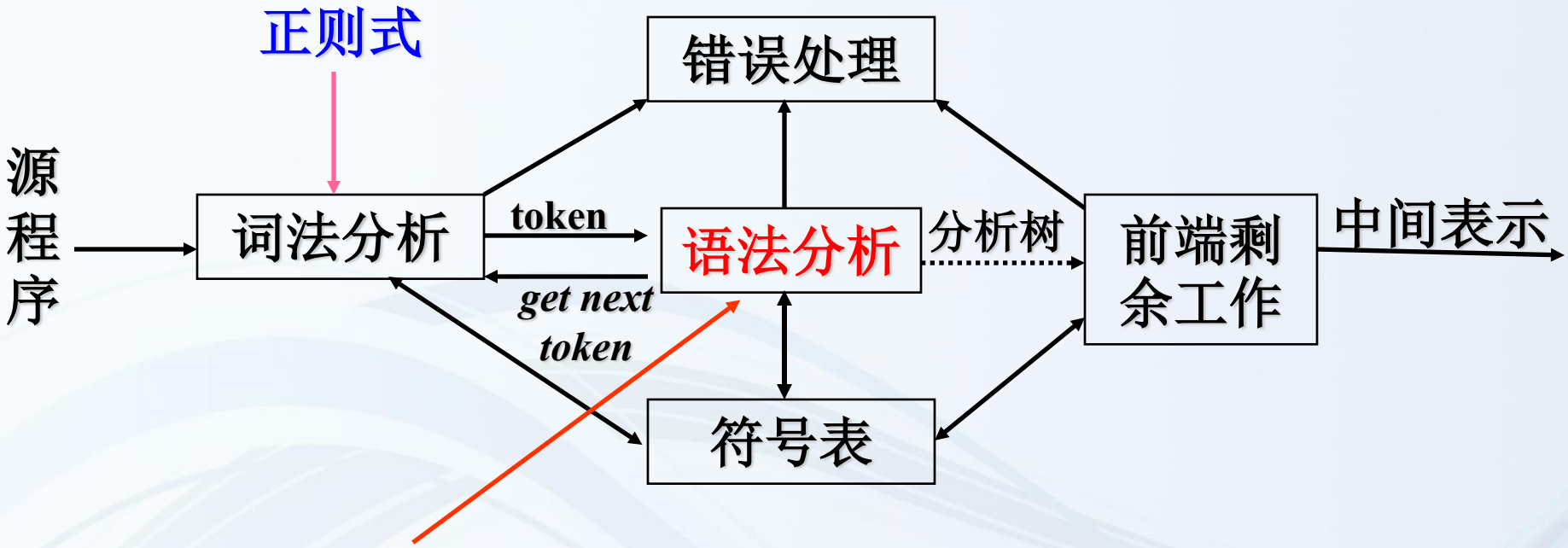
语法分析

4.1 引论



基于**文法**检查记号流的**构成**
生成**分析树**
发现、报告**语法错误**，并恢复

语法分析器的位置

语法分析和文法

- 为什么通过文法形式化地描述语言？
  - 精确易懂的表示方法
  - 编译器构建工具可以基于文法来自动构建编译器
  - 使语言的演化变得容易

❖为什么不继续使用正则式描述语言？

正则式的描述能力不足（正则式能描述正则语言，但不足以描述上下文无关语言）

证明：不存在识别$L=\{x^n y x^n | n=0,1,2,...\}$的DFA

反证：假设有一个状态数为K的DFA能够识别L

则对于K+1个不同的输入$x, x^2, x^3, ..., x^K, x^{K+1}$

必然存在a,b （1<=a<b<=K+1）

DFA在处理x^a和x^b后转向同一状态S

因为DFA接受$x^a y x^a$,所以在状态S继续输入$y x^a$将到达终态，

因此该DFA接受$x^b y x^a$。

由于a,b不等，所以此DFA识别的语言不是L。

所以原假设不成立，即不存在识别L的DFA。

4.2 上下文无关文法（复习）

❖文法

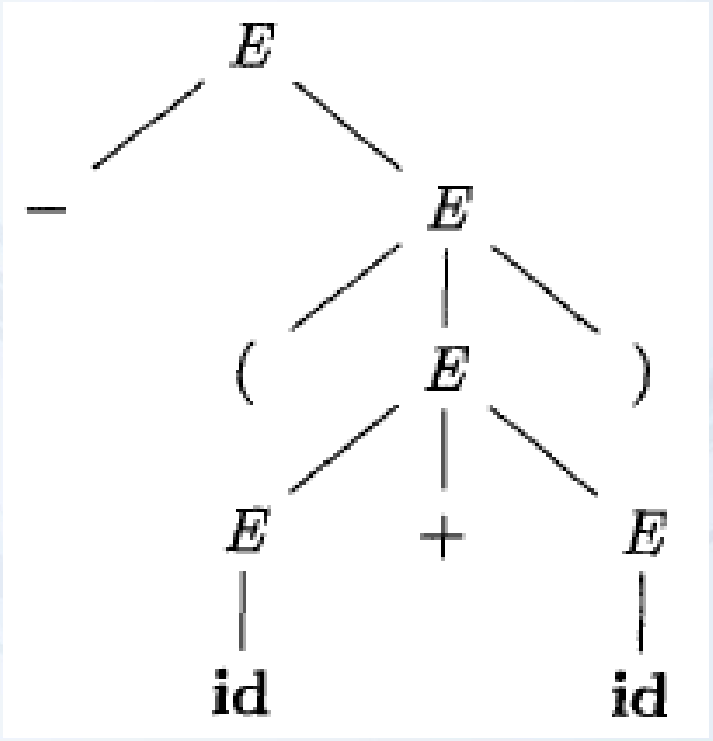
$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid id$

❖推导

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E)$

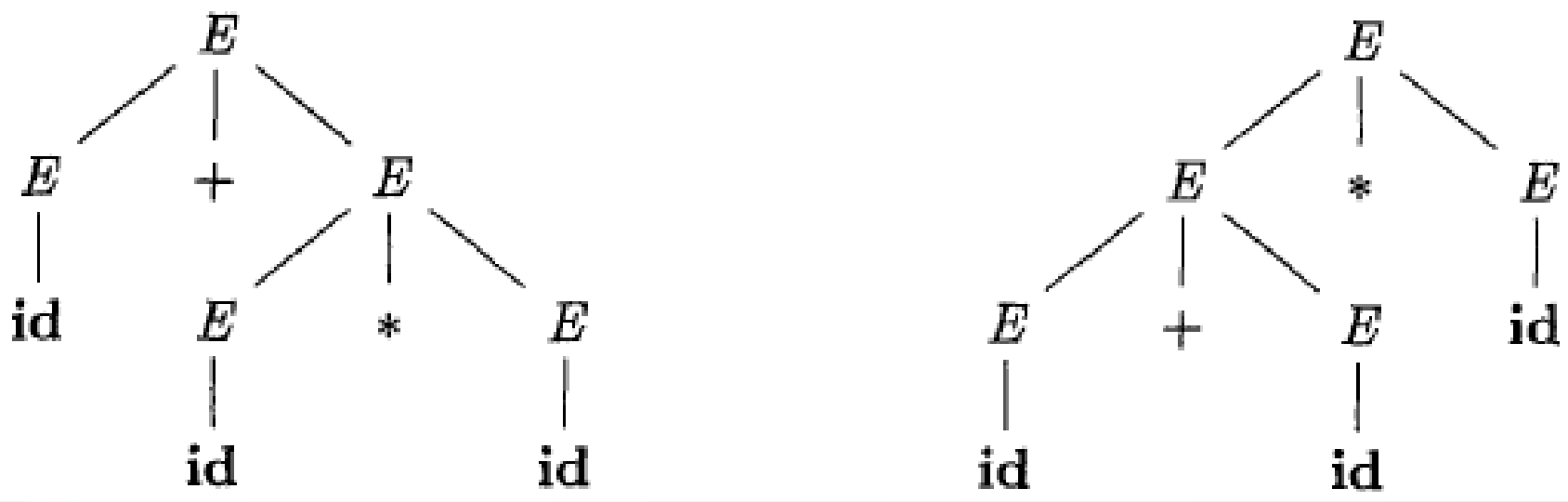
$\Rightarrow -(id+E) \Rightarrow -(id+id)$

❖语法分析树



文法的二义性（复习）

- 一个文法是二义的，如果：
  - 能为一个句子生成多棵语法分析树（推荐）
  - 能为一个句子生成多个最左推导
  - 能为一个句子生成多个最右推导



4.3 设计文法

二义性的去除

if E_1 then if E_2 then S_1 else S_2

```
if E1 then
    if E2 then
        S1
    else
        S2
```

vs

```
if E1 then
    if E2 then
        S1
else
        S2
```

stmt → if *expr* then *stmt*

| if *expr* then *stmt* else *stmt*

| other (any other statement)

二义性的去除（续）

原来的文法:

stmt → if *expr* then *stmt*

| if *expr* then *stmt* else *stmt*

| other (any other statement)

修改后的文法:

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

递归

❖直接递归(规则递归）：产生式体的左部和右部具有相同非终结符。

$A \rightarrow \alpha A \beta$

❖间接递归（文法递归）：文法中的某个非终结符，能建立一个推导过程，得到的符号串中又出现该非终结符本身。

$A \overset{+}{\Rightarrow} \alpha A \beta$

通过递归，我们能用有限的规则去定义无穷集合的语言

直接左递归(*direct left recursion*)

❖产生式体的最左边的符号和产生式头部相同。

❖左递归的消除：

将 $A \rightarrow A\alpha \mid \beta$

改成： $A \rightarrow \beta R$ $R \rightarrow \alpha R \mid \varepsilon$

例：消除文法expr → expr + term | term的左递归

解：引入非终结符R, 将文法改写为

expr → term R

R → + term R | ε

直接左递归的消除

for (each non-terminal A) {

(1) 将A为左部的规则组织成：

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \delta_1 \mid \delta_2 \mid \ldots \mid \delta_n$$

(2) 引入非终结符A’，将以上规则改写成：

$$A \rightarrow \delta_1 A' \mid \delta_2 A' \mid \ldots \mid \delta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$$

}

间接左递归(*indirect left recursion*)

❖定义：文法G存在推导$A \overset{+}{\Rightarrow} A\alpha$

由多步推导带来的左递归

$$\left.\begin{array}{l} S \to Aa \mid b \\ A \to Sd \mid \varepsilon \end{array}\right\} \quad S \Rightarrow Aa \Rightarrow Sda$$

例:以下文法加入哪一条规则可导致左递归(多选)

S→A　　A→B|C　　B→(C)

C→B+C|D　　D→1|0

(A) C→C+B

(B) D→B

(C) C→1C

(D) B→C

(E) A→D

(F) D→A

间接左递归的消除

设定非终结符顺序，假设依次记为$A_1, A_2, \ldots, A_n$.

for (each i from 1 to n) {

(1) 消除与A_i相关的间接左递归：

for (each j from 1 to i-1) {

向$A_i \rightarrow A_j\gamma$ 中代入以A_j为头的规则，得 $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$ ，

其中 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ 为所有以A_j为头的规则

}

(2) 消除与A_i相关的直接左递归

}

输入文法有环（$A \overset{+}{\Rightarrow} A$）或ε规则（$A \rightarrow \varepsilon$）时，无法保证消除左递归。

消除间接左递归（例）

消除文法G[S]的左递归：S→Aa|b　　A→Ac|Sd|ε

解：设定非终结符顺序为S, A

S不存在左递归。

消除A存在的间接左递归，变为:A→Ac | Aad | bd | ε

消除A存在的直接左递归：引入非终结符A’,变为：

A→bdA’ | A’　　A’→cA’ | adA’ | ε

所得文法为：

S→Aa | b　　A→bdA’ | A’　　A’→ cA’ | adA’| ε

消除间接左递归（例2）

消除文法G[S]的左递归：S→Qc|c　　Q→Rb|b　　R→Sa|a

解：设定非终结符顺序为S,Q,R

S和Q不存在左递归。

R存在间接左递归R→Sa，将S代入得：R→Qca|ca|a

R存在间接左递归R→Qca，将Q代入得：R→Rbca|bca|ca|a

R存在直接左递归R→Rbca，引入R’，消除左递归得：

R→bcaR’ | caR’ | aR’　　R’→bcaR’| ε

所得文法为：

S→Qc|c　　Q→Rb|b　　R→bcaR’ | caR’ | aR’

R’→bcaR’| ε

提取左公共因子

某个非终结符的多个产生式具有相同前缀时，称存在左公共因子

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

| if *expr* then *stmt*

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

左公共因子

转化为:

$A \rightarrow \alpha A'$　　$A' \rightarrow \beta_1 \mid \beta_2$

Choose the alternative that correctly left factors "if" statements in the given grammar

```
EXPR → if BOOL then { EXPR }
     | if BOOL then { EXPR } else { EXPR }
     | ...
BOOL → true | false
```

A

```
EXPR → if true then { EXPR }
     | if false then { EXPR }
     | if true then { EXPR } else { EXPR }
     | if false then { EXPR } else { EXPR }
     | ...
```

B

```
EXPR  → if BOOL EXPR'
      | ...
EXPR' → then { EXPR }
      | then { EXPR } else { EXPR }
BOOL  → true | false
```

C

```
EXPR  → EXPR' | EXPR' else { EXPR }
EXPR' → if BOOL then { EXPR }
      | ...
BOOL  → true | false
```

D

```
EXPR  → if BOOL then { EXPR } EXPR'
      | ...
EXPR' → else { EXPR } | ε
BOOL  → true | false
```

提取左公共因子（算法）

对每个非终结符A

找到最长公共前缀α（并非针对所有产生式）

若 $\alpha \neq \varepsilon$

将$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$ （γ代表所有不以α开头的产生式组）

替换为： $A \rightarrow \alpha A' \mid \gamma$ $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

反复进行以上替换，直至$\alpha = \varepsilon$

提取左公共因子（例）

改造以下文法G，以利于无回溯的自顶向下分析：

$S \rightarrow abcd \mid abce \mid abf$

解：S为左部的产生式体的最长公共前缀为abc, 引入A， 将文法变为：$S \rightarrow abcA \mid abf$　　$A \rightarrow d \mid e$

S为左部的产生式体的最长公共前缀为ab,引入B，得：

$S \rightarrow abB$　　$B \rightarrow cA \mid f$

所得文法为：

$S \rightarrow abB$　　$A \rightarrow d \mid e$　　$B \rightarrow cA \mid f$

改造以下文法，使其可能用于无回溯的自顶向下分析。

$S \rightarrow SS+ \mid SS* \mid a$

解：（1）提取左公共因子SS得：

$S \rightarrow SSA \mid a$　　$A \rightarrow + \mid *$

（2）设定文法符号顺序S, A。

仅S存在直接左递归，消除左递归得：$S \rightarrow aR$　$R \rightarrow SAR \mid \varepsilon$

最终文法为：

$S \rightarrow aR$　　$R \rightarrow SAR \mid \varepsilon$　　$A \rightarrow + \mid *$