编译原理

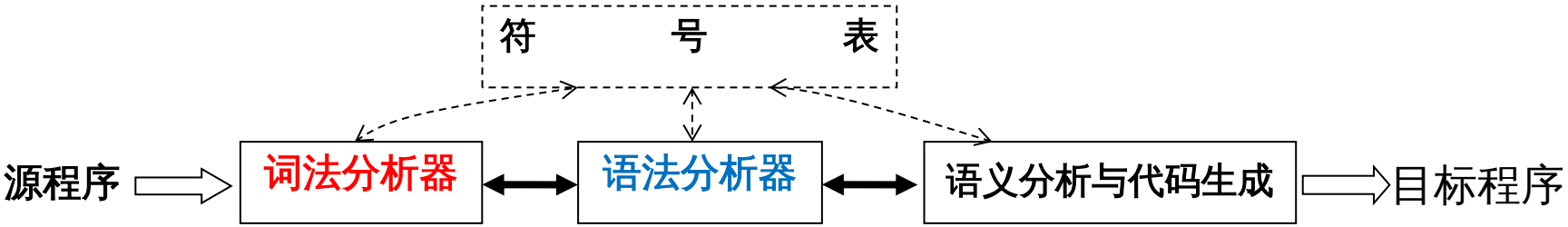
第四章
语法分析

哈尔滨工业大学　陈鄞　单丽莉

引言

- 语法分析器：
  - 功能：组词成句
  - 输入：词法单元序列
  - 输出：语法分析树

编译程序的组织



以语法分析器为中心

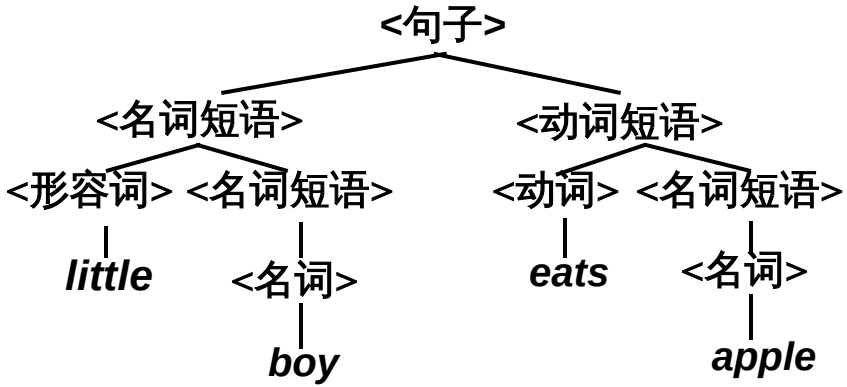
- 遍：编译程序对源程序或中间结果的完整扫描称为遍。
- 依据任务的复杂度、存储和效率需求
  - 可以将若干阶段组织为一遍处理

  例：如图以语法分器为中心的若干阶段的一遍处理

引言

- 语法分析的主要任务
  - 根据给定的文法，识别输入句子的各个成分，构造句子的分析树（显式或隐式）
- 大部分程序设计语言的语法构造可以用*CFG*来描述，*CFG*以 *token*作为终结符

语法分析的种类

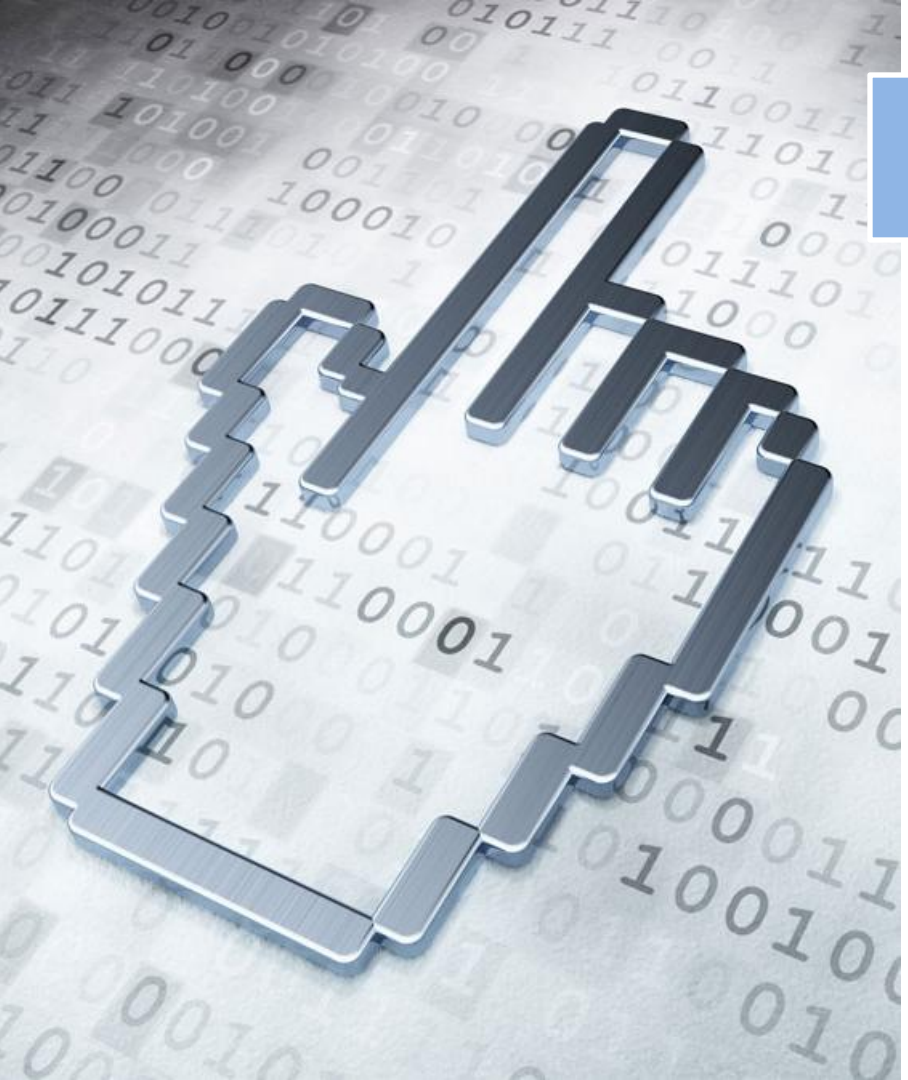


从左向右扫描输入，每次扫描一个符号

- 自顶向下的分析(*Top-Down Parsing*)
  - 从分析树的顶部（根节点）向底部（叶节点）构造分析树
  - 从文法开始符号*S*推导出串*w*
- 自底向上的分析(*Bottom-up Parsing*)
  - 从分析树的底部（叶节点）向顶部（根节点）构造分析树
  - 将一个串*w*归约为文法开始符号*S*

引言

- 通用的语法分析方法可对任意CFG进行语法分析，但可能需要回溯，效率极低（$\geq O(n^3)$），难于实践应用。
- 最高效的自顶向下和自底向上方法只需要线性时间，但只能处理某些CFG文法子类，其中的某些子类，特别是 *LL(k)* 和*LR(k)*文法，其表达能力足以描述现代程序设计语言的大部分语法构造，常用于实践。



本章内容

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具

4.1 自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
  - 例

文法

① $E \rightarrow E + E$

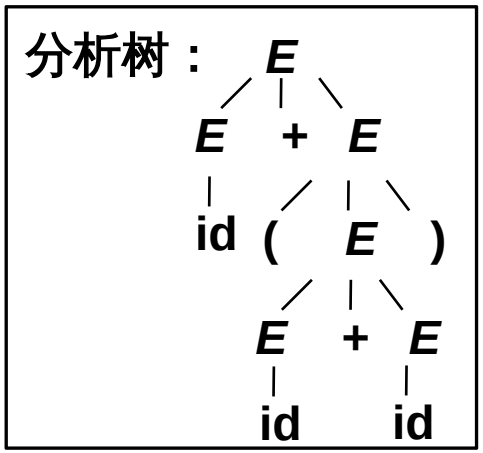
② $E \rightarrow E * E$

③ $E \rightarrow (E)$

④ $E \rightarrow id$

输入

id + (id + id)



推导：E

$E + E$

$E + (E)$

$E + (E + E)$

$E + (id + E)$

$id + (id + E)$

$id + (id + id)$

- 自顶向下推导要做两个决策
  - 选择哪个非终结符推导？
  - 选择该非终结符的哪个候选式推导？

核心问题：
如何实现确定的分析？

最左推导(*Left-most Derivation*)

- 最左推导
  - 最左推导：总是选择每个句型的最左非终结符进行替换。
  - 最左句型：如果$S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$，则称$\alpha$是当前文法的最左句型

- 最右推导
  - 最右推导：总是选择每个句型的最右非终结符进行替换
  - 规范推导：在自底向上的分析中，总是采用最左归约的方式，因此把最左归约称为规范归约，而最右推导相应地称为规范推导

最左推导和最右推导的唯一性

$E \Rightarrow E+E$

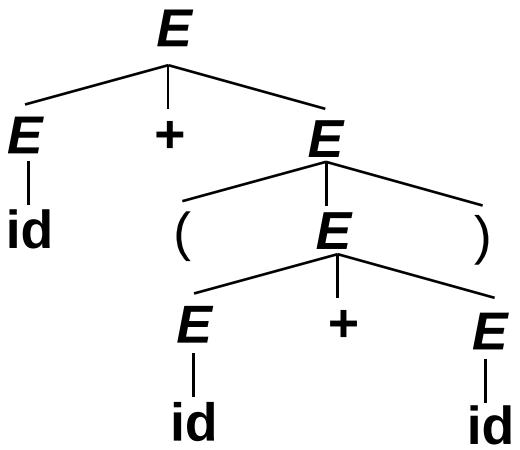
$\Rightarrow E+(E)$

$\Rightarrow E+(E+E)$

$\Rightarrow E+(id+E)$

$\Rightarrow id+(id+E)$

$\Rightarrow id+(id+id)$



$E \Rightarrow_{lm} E+E$

$\Rightarrow_{lm} id+E$

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

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

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

$\Rightarrow_{lm} id+(id+id)$

$E \Rightarrow E+E$

$\Rightarrow id+E$

$\Rightarrow id+(E)$

$\Rightarrow id+(E+E)$

$\Rightarrow id+(E+id)$

$\Rightarrow id+(id+id)$

正是利用这种唯一性，使得语法分析器可以进行确定的分析。

$E \Rightarrow_{rm} E+E$

$\Rightarrow_{rm} E+(E)$

$\Rightarrow_{rm} E+(E+E)$

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

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

$\Rightarrow_{rm} id+(id+id)$

自顶向下的语法分析采用最左推导方式

- 总是选择每个句型的最左非终结符进行替换
- 根据输入流中的下一个终结符，选择最左非终结符的一个候选式

自顶向下分析的两个选择：

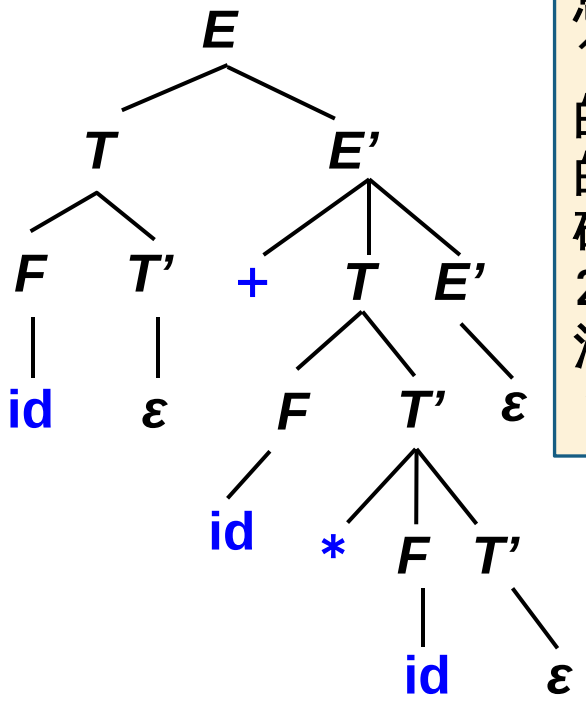
- 最左推导解决了非终结符的选择的确定性
- 如何选择候选式呢？

例

- 文法
  - ① $E \rightarrow T\ E'$
  - ② $E' \rightarrow +\ T\ E' \mid \varepsilon$
  - ③ $T \rightarrow F\ T'$
  - ④ $T' \rightarrow *\ F\ T' \mid \varepsilon$
  - ⑤ $F \rightarrow (\ E\ ) \mid id$
- 输入

  id + id * id$

  ↑ ↑ ↑ ↑ ↑ ↑



思考：
1．如果具有相同左部的候选式能生成相同的首终结符是否还能确定地选择候选式？
2．符合什么条件的文法才能做确定的选择？

- 候选式的选择：
  启发式策略：各候选式生成的首终结符与当前的输入符号是否匹配。

自顶向下分析过程中的问题

1.二义性问题

- 对于文法*G*，如果*G*是二义性文法。那么，*L*(*G*)中存在一个具有两个或两个以上最左(或最右)推导的句子。
- 设文法*G*是二义性的，*w*　*L*(*G*)且*w*存在两个最左推导，则在对*w*进行自顶向下的语法分析时，语法分析程序将无法确定采用*w*的哪个最左推导。
- 解决方法：改造文法或在分析时使用“消歧规则”

自顶向下分析过程中的问题

2. 左公因子引起的回溯问题

- 例：文法*G*

  $S \rightarrow aAd \mid aBe$

  $A \rightarrow c$

  $B \rightarrow b$

- 输入

  a b c $

  ↑

最左推导：

S　aAd

S　acd

a与b不匹配

失败?

回退

S　aBe

S　abe

c与e不匹配

失败!

- 同一非终结符的多个候选式存在共同前缀时，根据当前输入符号也无法进行唯一的候选式选择，很可能导致回溯现象，严重影响分析效率。
- 解决方法：改造文法，提取左公因子。

对上下文无关文法的改造

2. 提取左公因子(*Left Factoring*)

- 文法*G*

  $S \rightarrow aAd \mid aBe$

  $A \rightarrow c$

  $B \rightarrow b$

- 文法*G '*

  $S \rightarrow aS'$

  $S' \rightarrow Ad \mid Be$

  $A \rightarrow c$

  $B \rightarrow b$

思想：通过提取左公因子改写产生式来推迟决定，待读入了足够多的输入时，便能做出确定的选择。

提取左公因子算法

- 输入：文法G
- 输出：等价的提取了左公因子的文法
- 方法：

对于每个非终结符A，找出它的两个或多个选项之间的最长公共前缀α。如果$\alpha \neq \varepsilon$，即存在一个非平凡的(*nontrivial*)公共前缀，那么将所有A-产生式

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$$

替换为

$$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

其中，γ_i 表示所有不以α开头的产生式体；A'是一个新的非终结符。不断应用这个

自顶向下分析过程中的问题

3. 左递归引起的问题

- 文法G

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

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

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

- 输入

  id + id * id $

  ↑

E

$E + T$

$E + T + T$

$E + T + T + T$

…

左递归文法可能会使自顶向下分析陷入无限循环

解决办法：消除左递归

对上下文无关文法的改造

1.消除直接左递归

- 思想：左递归转换为右递归
- 引入新的变量A'，将左递归产生式：

$A \rightarrow A\alpha \mid \beta$　　($\alpha \neq \varepsilon$, β不以A开头)

替换为

$A \rightarrow \beta A'$　　$A' \rightarrow \alpha A' \mid \varepsilon$

$r=\beta\alpha^*$

A　$A\alpha$
$A\alpha\alpha$
$A\alpha\alpha\alpha$
…
$A\alpha \ldots\alpha$
$\beta\ \underline{\alpha \ldots\alpha}$

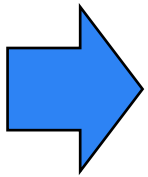
A'　$\alpha A'$
$\alpha\alpha A'$
$\alpha\alpha\alpha A'$
…
$\alpha \ldots\alpha A'$
$\underline{\alpha \ldots\alpha}$

消除直接左递归

$$A \rightarrow A\alpha \mid \beta \Rightarrow \quad A \rightarrow \beta A' \qquad A' \rightarrow \alpha A'$$

例： $\mid \varepsilon$

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$



$$E \rightarrow T\ E'$$
$$E' \rightarrow + T\ E' \mid \varepsilon$$
$$T \rightarrow F\ T'$$
$$T' \rightarrow * F\ T' \mid \varepsilon$$
$$F \rightarrow (E) \mid id$$

消除直接左递归的一般形式

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

($\alpha_i \neq \varepsilon$, β 不以A开头)

$$A \quad \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$

$$A' \quad \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$$

消除左递归是要付出代价的——引进了一些非终结符和ε_产生式

消除间接左递归

- 两个非终结符　　$S \rightarrow A\,a \mid b$

  $A \rightarrow A\,c \mid S\,d \mid \varepsilon$

  S　Aa

  Sda

  - 将S的定义代入A-产生式，得：

    $A \rightarrow A\,c \mid A\,a\,d \mid b\,d \mid \varepsilon$

  - 消除A-产生式的直接左递归，得：

    $A \rightarrow b\,d\,A' \mid A'$

    $A' \rightarrow c\,A' \mid a\,d\,A' \mid \varepsilon$

  - 得改造后文法：$S \rightarrow A\,a \mid b$

    $A \rightarrow b\,d\,A' \mid A'$　　$A' \rightarrow c\,A' \mid a\,d\,A' \mid \varepsilon$

消除间接左递归

- 多个非终结符

$$A->Ba|a \quad B->Ab|Cb|b \quad C->Ac|c$$

1. 将非终结符排序 *A B C* (改造后的文法与排序有关)
2. 首先处理*A*产生式
   ① 将排在A之前的非终结符依次代入A产生式：无需代入
   ② 消除A产生式的直接左递归:无直接左递归
3. 再处理*B*产生式
   ① 将排在B之前的非终结符依次代入B产生式: 将A的产生式代入B
   ② 消除B产生式的直接左递归
4. 最后处理*C*产生式
   ① 将排在C之前的非终结符依次代入C产生式:将A的产生式代入C，再将B的产生式代入C
   ② 消除C产生式的直接左递归

消除间接左递归

$$A->Ba|a \quad B->Ab|Cb|b \quad C->Ac|c$$

例：将非终符排序 *A B C*，消除间接左递归之后：

$A->Ba|a$

$B-> abB'|CbB'|bB'$
$B' -> abB' | \varepsilon$

$C -> abB'acC'| bB' acC'| acC'| cC'$
$C' -> bB'acC' | \varepsilon$

练习：按*A*、*C*、*B*的排序，消除间接左递归

消除左递归算法

- 输入：不含循环推导（即形如$A \overset{+}{\Rightarrow} A$的推导）和ε-产生式的文法$G$
- 输出：等价的无左递归文法
- 方法： 思想：采用代入法将间接左递归变为直接左递归，再消除直接左递归

1) 按照某个顺序将非终结符号排序为A_1，A_2，…，A_n
2) for (从1到n的每个i) {
3) for (从1到i -1的每个j) {
4) 将每个形如$A_i \rightarrow A_j\,\gamma$的产生式替换为产生式组 $A_i \rightarrow \delta_1\,\gamma | \delta_2\,\gamma | \ldots | \delta_k\,\gamma$ ，其中$A_j \rightarrow \delta_1 | \delta_2 | \ldots | \delta_k$ ，是所有的A_j 产生式
5) }
6) 消除A_i 产生式之间的直接左递归
7) }

i=3时，将A3产生式体中所有首部的A1或A2分别替换为A1或A2产生式的右部

i=3时，消除A3产生式的直接左递归

课后练习——消除左递归

请消除下列文法的左递归：

- **分别将非终结符排序为：*S, Q, R* 和 *R, Q, S***
  - ***S->Qc|c*** ***Q->Rb|b*** ***R->Sa|a***

自顶向下语法分析的通用形式

- 递归下降分析 (*Recursive-Descent Parsing*)
  - 由一组过程组成，每个过程对应一个非终结符
  - 从文法开始符号*S*对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果*S*对应的过程体恰好扫描了整个输入串，则成功完成语法分析

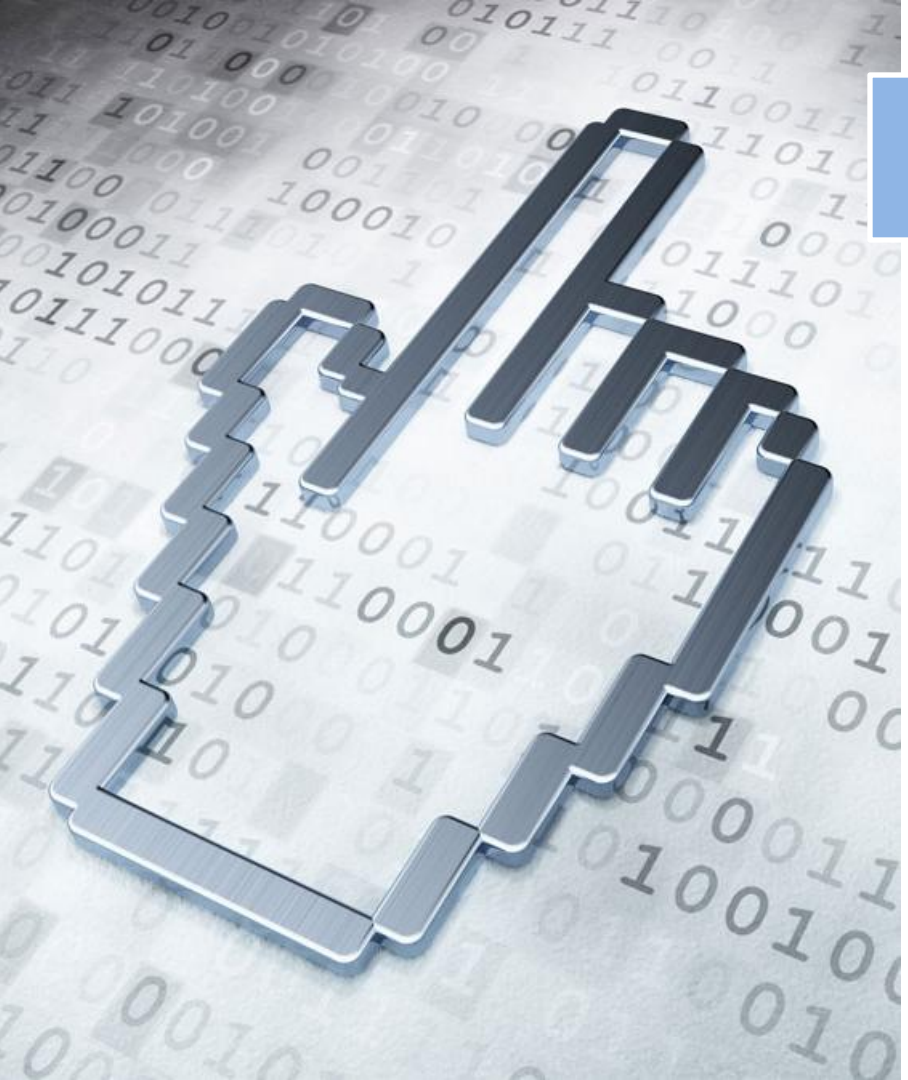
```
    void A( ) {
1)    选择一个A产生式， A →X1 X2 … Xk ;
2)    for ( i = 1 to k ) {
3)        if ( Xi是一个非终结符号)
4)            调用过程 Xi ( ) ;
5)        else  if ( Xi 等于当前的输入符号a)
6)            读入下一个输入符号;
7)        else /* 发生了一个错误 */ ;
        }
    }
```

如何有效地选择候选式？

不确定分析：可能需要回溯(*backtracking*)，导致效率较低

预测分析（*Predictive Parsing*）

- 预测分析技术是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来指导产生式的选择，可以实现确定的分析。
  - 可以对某些文法构造出向前看*k*个输入符号的预测分析器，不需要回溯，该类文法有时也称为***LL(k)* 文法类**



提纲

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具

4.2 预测分析法

- 4.2.1 LL(1) 文法
- 4.2.2 递归的预测分析法
- 4.2.3 非递归的预测分析法
- 4.2.4 预测分析中的错误恢复

4.2.1 LL(1) 文法

假如允许S_文法包含ϵ产生式，将会产生什么问题？

- 预测分析法的工作过程
  - 从文法开始符号出发，在每一步推导过程中根据当前句型的最左非终结符*A*和当前输入符号*a*，选择正确的*A*-产生式。为保证分析的确定性，选出的候选式必须是唯一的。
- *S*_文法（简单的确定性文法，*Korenjak & Hopcroft*，19

每个产生式的右部都以终结符开始

同一非终结符的各个候选式的首终结符都不同

S_文法不含ε产生式

例：文法

① $S \rightarrow aBCD$
② $B \rightarrow bC$
③ $B \rightarrow dB$
④ $B \rightarrow \varepsilon$
⑤ $C \rightarrow c$
⑥ $C \rightarrow a$
⑦ $D \rightarrow e$

➢ 输入 a d adde e

➢ 推导

S	S
aBCD	aBCD
adBCD	adBCD
adCD	adCD
adaD	失败！
adae	

可以紧跟 *B*后面出现的终结符：*c*、*a*

- 什么时候使用ε产生式？
  - 如果当前某非终结符*A*与当前输入符*a*不匹配时，若存在$A \rightarrow \varepsilon$，可以通过检查*a*是否可以出现在 *A*的后面，来决定是否使用产生式 $A \rightarrow \varepsilon$（若文法中无 $A \rightarrow \varepsilon$ ，或*a*不可以出现在*A*的后面，则应报错）

非终结符的后继符号集

- 非终结符*A*的后继符号集
  - 可能在某个句型中紧跟在*A*后边的终结符*a*的集合，记为*FOLLOW*(*A*)

$$FOLLOW(A) = \{ a \mid S \overset{*}{\Rightarrow} \alpha A a \beta,\ a \in V_T,\ \alpha, \beta \in (V_T \cup V_N)^* \}$$

如果 *A*是某个句型的最右符号，则将结束符“$”添加到*FOLLOW*(*A*)中

例

(1) *S* → *aBC*
(2) *B* → *bC*
(3) *B* → *dB*
(4) *B* → ε
(5) *C* → *c*
(6) *C* → *a*
(7) *D* → *e*

输入符号
b
d
{ *a*, *c* }
e 失败

FOLLOW(*B*) = { *a*, *c* }

产生式的可选集

- 产生式$A \rightarrow \beta$的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为$SELECT(A \rightarrow \beta)$
  - $SELECT(A \rightarrow a\beta) = \{a\}$
  - $SELECT(A \rightarrow \varepsilon) = FOLLOW(A)$
- 确定预测分析的充要条件：

  具有相同左部的产生式的可选集互不相交

任意形式的产生式的可选集如何求呢？

串首终结符集

- **串首终结符集**
  - 串首第一个符号，并且是终结符。简称首终结符。

给定一个文法符号串α， α的**串首终结符集$FIRST(\alpha)$**被定义为可以从α推导出的所有串首终结符构成的集合。如果$\alpha \overset{*}{\Rightarrow} \varepsilon$，那么$\varepsilon$也在$FIRST(\alpha)$中

- 对于 $\alpha \in (V_T \cup V_N)^+$，

  $FIRST(\alpha) = \{ a \mid \alpha \overset{*}{\Rightarrow} a\beta，a \in V_T，\beta \in (V_T \cup V_N)^* \}$；
- 如果 $\alpha \overset{*}{\Rightarrow} \varepsilon$，那么 $FIRST(\alpha) = FIRST(\alpha) \cup \{\varepsilon\}$

计算文法符号X的FIRST(X)

- FIRST(X)：可以从X推导出的所有串首终结符构成的集合 *
- 如果 X ε，那么 FIRST(X) = FIRST(X) ∪{ ε }
  - 例

非终结符的FIRST集有依赖关系，需迭代计算

$S \to AB$，$S \to bC$

$A \to \varepsilon$，$A \to b$

$B \to \varepsilon$，$B \to aD$

$C \to AD$，$C \to b$

D

FIRST (S) = { b , a , ε }

FIRST (A) = { b , ε }

FIRST (B) = { a , ε }

FIRST (C) = { b , a , }

FIRST (D) = { a , ε }

要特别注意 空符号ε 成为串首终结符的条件，不能直接并入。

算法——计算文法符号X的FIRST(X)

- 不断应用下列规则，直到没有新的终结符或ε可以被加入到任何FIRST集合中为止
  - 若$X \in V_T$，那么$FIRST(X) = \{X\}$
  - 若$X \in V_N$，且有$X \to \varepsilon \in P$，那么将ε加入到$FIRST(X)$中
  - 若$X \in V_N$，且有$A \to a\beta \in P$，$a \in V_T$，那么将a加入到$FIRST(X)$中
  - 若$X \in V_N$，且有$X \to Y_1 \ldots Y_k \in P$ $(k \geq 1)$，且$Y_1 \in V_N$

  那么如果对于某个i，a在$FIRST(Y_i)$中且ε在所有的$FIRST(Y_1)$，…，$FIRST(Y_{i-1})$中(即$Y_1 \ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$)，就把a加入到$FIRST(X)$中。

  如果对于所有的$j = 1, 2, \ldots, k$，ε在$FIRST(Y_j)$中，那么将ε加入到$FIRST(X)$

计算串$X_1X_2 \ldots X_n$的*FIRST*集合

$S \to AB$，$S \to bC$

$A \to \varepsilon$，$A \to b$

$B \to \varepsilon$，$B \to aD$

$C \to AD$，$C \to b$

$D \to aS$，$D \to c$

FIRST(*S*) = { *b*, *a* , *ε* }

FIRST(*A*) = { *b* , *ε* }

FIRST(*B*) = { *a* , *ε* }

FIRST(*C*) = { *b* , *a* , }

FIRST(*D*) = { *a* , *c* }

- 求：*FIRST*(*AB*)

= (FIRST(*A*) −{ *ε* })U (FIRST(*B*) −{ *ε* }) U { *ε* }

= { *a*, *b*, *ε* }

由于*ε* ∈ FIRST(*A*)，即 $A \overset{*}{\Rightarrow} \varepsilon$

由于*ε* ∈ FIRST(*A*) 且*ε* ∈ FIRST(*B*) 即 $AB \overset{*}{\Rightarrow} \varepsilon$

计算串$X_1X_2\ldots X_n$的*FIRST*集合

- 向*FIRST*($X_1X_2\ldots X_n$)加入*FIRST*(X_1)中所有的非ε符号
- 如果ε在*FIRST*(X_1)中，再加入*FIRST*(X_2)中的所有非ε符号；如果ε在*FIRST*(X_1)和*FIRST*(X_2)中，再加入*FIRST*(X_3)中的所有非ε符号，以此类推
- 最后，如果对所有的i，ε都在*FIRST*(X_i)中，那么将ε加入到*FIRST*($X_1X_2\ldots X_n$)中

产生式$A \to \alpha$的可选集

- 产生式$A \to \alpha$的**可选集**是指可以选用该产生式进行推导时，对应的当前输入符号（向前看符号）的集合，记为$SELECT(A \to \alpha)$。
- 产生式$A \to \alpha$的可选集$SELECT$
  - 如果 $\varepsilon \notin FIRST(\alpha)$，那么

$$SELECT(A \to \alpha) = FIRST(\alpha)$$

  - 如果 $\varepsilon \in FIRST(\alpha)$，那么

$$SELECT(A \to \alpha) = (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A)$$

LL(1)文法

- 文法G是LL(1)的，当且仅当G的任意两个具有相同左部的产生式$A \to \alpha \mid \beta$ 满足条件：

$$SELECT(A \to \alpha) \cap SELECT(A \to \beta) = \Phi$$

等价于满足下面的条件：

① $FIRST(\alpha) \cap FIRST(\beta) = \Phi$ （α和β至多有一个能推导出ε）

② 如果 $\beta \overset{*}{\Rightarrow} \varepsilon$，则$FIRST(\alpha) \cap FOLLOW(A) = \Phi$；

③ 如果 $\alpha \overset{*}{\Rightarrow} \varepsilon$，则$FIRST(\beta) \cap FOLLOW(A) = \Phi$；

- 可以为LL(1)文法构造确定的预测分析器，进行确定的自顶向下分析。
  - 第一个“L”表示从左向右扫描输入，第二个“L”表示产生最左推导
  - “1”表示在每一步中只需要向前看一个输入符号来决定语法分析动作

判断文法是否是*LL*(1)文法

假设，文法不存在左递归、左公因子。

- 第一步，计算非终结符的***FIRST*集**
- 第二步，计算非终结符的***FOLLOW*集**
- 第三步，计算具有**相同左部的产生式**的***SELECT*集**
- 第四步，根据**具有相同左部的产生式的*SELECT*集是否相交**，做出判断，给出结论。

计算非终结符A的FOLLOW(A)

- FOLLOW(A)：可能在某个句型中紧跟在A后边的终结符a的集合
- 如果A是某个句型的最右符号，则将结束符”$”添加到FOLLOW(A)中

例 首先，将$添加到Follow(E)　非终结符的Follow集有依赖关系，需迭代计算

① $E \rightarrow TE'$　$FIRST(E) = \{ (, id \}$　$FOLLOW(E) = \{\$,) \}$

② $E' \rightarrow +TE' \mid \varepsilon$　$FIRST(E') = \{ +, \varepsilon \}$　$FOLLOW(E') = \{ \$,) \}$

③ $T \rightarrow FT'$　$FIRST(T) = \{ (, id \}$　$FOLLOW(T) = \{ +, \$,) \}$

④ $T' \rightarrow *FT' \mid \varepsilon$　$FIRST(T') = \{ *, \varepsilon \}$　$FOLLOW(T') = \{ +, \$,) \}$

⑤ $F \rightarrow (E) \mid id$　$FIRST(F) = \{ (, id \}$　$FOLLOW(F) = \{ *, +, \$,) \}$

算法——计算非终结符*A*的*FOLLOW*(*A*)

- 不断应用下列规则，直到没有新的终结符可以被加入到任何*FOLLOW*集合中为止
  - 将$放入*FOLLOW*(*S*)中，其中S是开始符号，$是输入右端的结束标记
  - 如果存在一个产生式$A \rightarrow \alpha B \beta$，那么*FIRST* (β)中除ε 之外的所有符号都加入*FOLLOW*(*B*)中
  - 如果存在一个产生式$A \rightarrow \alpha B$，或存在产生式$A \rightarrow \alpha B \beta$且*FIRST* (β) 包含ε，那么 *FOLLOW*(*A*)中的所有符号都加入*FOLLOW*(*B*)中

例：表达式文法各产生式的SELECT 集

X	FIRST (X)	FOLLOW (X)
E	(id	$)
E '	+ ε	$)
T	(id	+) $
T '	* ε	+) $
F	(id	* +) $

表达式文法是 LL(1) 文法

(1) E → T E '
(2) E' → + T E '
(3) E ' → ε
(4) T → F T '
(5) T ' → * F T '
(6) T ' → ε
(7) F → (E)
(8) F → id

SELECT (1)= FIRST(TE')
= { (, id }
SELECT (2) = FIRST(+TE ')
= { + }
SELECT (3)= FOLLOW(E ')
= { $,) }
SELECT (4) = FIRST(FT ')
= { (, id }
SELECT (5) = FIRST(*FT ')
= { * }
SELECT (6)= { + ,) , $ }
SELECT (7)= { (}
SELECT (8)= { id }

SELECT(2) ∩ SELECT(3) = Φ
SELECT(5) ∩ SELECT(6) = Φ
SELECT(7) ∩ SELECT(8) = Φ

为文法构造预测分析表

可以证明：文法G是LL(1)文法，当且仅当文法G的预测分析表中，任何表项都不包含多重定义的冲突条目，即不包含两条以上的产生式。

	产生式	SELECT
E	E→TE'	(id
E'	E'→+TE'	+
	E'→ε	$)
T	T→FT'	(id
T'	T'→*FT'	*
	T'→ε	+) $
F	F→(E)	(
	F→id	id

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+T E'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε

预测分析表的构造算法

算法4.6 预测分析表(*LL*(1)分析表)的构造算法。

输入：文法*G*;

输出：分析表*M*;

步骤：

1. 对*G*中的任意一个产生式$A \to \alpha$, 执行第2步和第3步;
2. for　*a*　*FIRST*(α), 将$A \to \alpha$填入*M*[*A*, *a*];
3. if ε　*FIRST*(α) then　*a*　*FOLLOW*(*A*), 将$A \to \alpha$填入*M*[*A*, *a*];
4. 将所有无定义的*M*[*A*, *b*]标上出错标志。

LL(1)文法的分析方法

- 递归的预测分析法
- 非递归的预测分析法

4.2.2 递归的预测分析法

- 递归的预测分析法是指：在递归下降分析中，根据预测分析表进行产生式的选择
  - 根据每个非终结符的产生式和*LL*(1)文法的预测分析表，为每个非终结符编写对应的过程

```
    void A( TOKEN ) {
1)    选择一个A产生式，A →X1 X2 … Xk ;
2)    for ( i = 1 to k ) {
3)          if ( Xi是一个非终结符号)
4)                调用过程 Xi ( TOKEN ) ;
5)          else  if ( Xi 等于当前的输入符号a)
6)                读入下一个输入符号;
7)          else /* 发生了一个错误 */ ;
          }
    }
```

TOKEN：当前输入符号
产生式的选择：根据TOKEN，选择表项M(*A*, *TOKEN*)中对应的产生式进行推导，如表项为空，错误。

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
program DESCENT;
    begin
        GETNEXT(TOKEN);
        PROGRAM(TOKEN);
        if TOKEN≠'$'  then
    ERROR;
        write('success!');
    end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
    begin
      if TOKEN≠'program'  then ERROR;
      GETNEXT(TOKEN);

      DECLIST(TOKEN);

      if TOKEN≠':'  then ERROR;
      GETNEXT(TOKEN);

      TYPE(TOKEN)

      if TOKEN≠';'  then ERROR;
      GETNEXT(TOKEN);

      STLIST(TOKEN);

      if TOKEN≠'end'  then ERROR;
      GETNEXT(TOKEN);    //$结束符
    end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLIST(TOKEN);
    begin
        if TOKEN≠'id'  then ERROR;
        GETNEXT(TOKEN);

        DECLISTN(TOKEN);
    end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLISTN(TOKEN);
    begin
        if TOKEN =',' then
         //选择产生式 (3) 进行推导
         begin
          GETNEXT(TOKEN);
           if TOKEN≠'id'  then ERROR;
           GETNEXT(TOKEN);
           DECLISTN(TOKEN);
         end
        else  if TOKEN≠':'  then ERROR;
         // 否则TOKEN= ':' 选择产生式(4)
            进行推导，执行空操作
    end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLIST(TOKEN);
    begin
        if TOKEN≠'s'  then ERROR;
        GETNEXT(TOKEN);
        STLISTN(TOKEN);
    end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLISTN(TOKEN);
        begin
            if TOKEN =';' then
              begin //选择产生式 (6) 进行推导
               GETNEXT(TOKEN);

               if TOKEN≠'s'  then ERROR;
               GETNEXT(TOKEN);

               STLISTN(TOKEN);
              end
             //选择产生式 (7) 进行推导
            else  if TOKEN≠'end'  then ERROR;
        end
```

例：假设已构建了预测分析表

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

```
procedure TYPE(TOKEN);
    begin
        if TOKEN≠'real' and TOKEN≠'int'
            then ERROR;
        GETNEXT(TOKEN);
    end
```

4.2.3 非递归的预测分析法

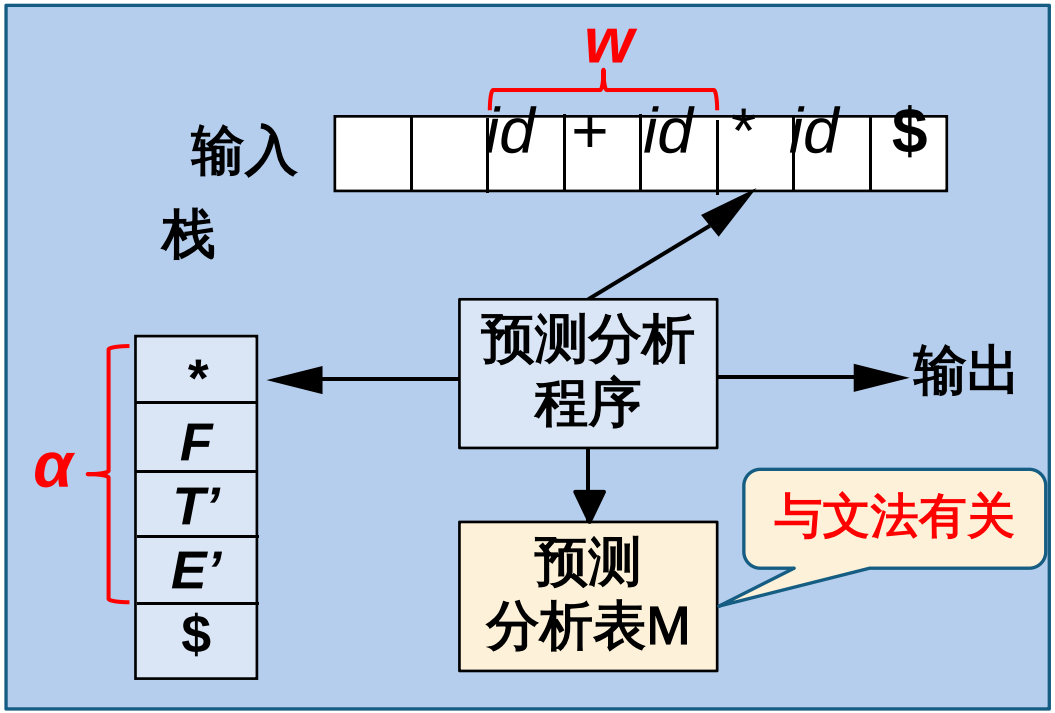
- 非递归的预测分析器，由预测分析程序、一个栈和一个预测分析表三部分组成，也叫表驱动的预测分析。

- 最左推导过程中的任意句型可以表示为：$w\alpha$ ，即：

$$S \overset{*}{\underset{lm}{\Rightarrow}} w\alpha$$

- *w*:已经被匹配的输入部分
- *α*:待分析栈中部分，左部在栈顶
- 若栈顶是终结符就进行匹配；
- 若栈顶是非终结符就查找预测分析表尝试替换（推导）

自动生成：预测分析表M的构造



例: 预测分析器的工作过程

非终结符	输入符号					
	id	+	*	(	)	$
E	$E\to TE'$			$E\to TE'$		
E'		$E'\to +TE'$			$E'\to\varepsilon$	
$E'\to\varepsilon$						
T	$T\to FT'$			$T\to FT'$		
T'		$T'\to\varepsilon$	$T'\to *FT'$		$T'\to\varepsilon$	
$T'\to\varepsilon$						
F	$F\to$ id			$F\to(E)$		

- 初始状态：栈中只有文法开始符号和栈底标志$.
- 若栈顶是非终结符A，根据输入符号a查找预测分析表项M[A,a]用产生式体替换栈顶符号，输出产生式。若表项为空，调用错误处理。
- 若栈顶是终结符，且与输入符号匹配，出栈，输入指针右移。不匹配，调用错误处理。
- 若栈顶为$, 输入符号也为$,则分析结束, 否则错误处理。

栈	剩余输入	输出
$E\ \$$	id+id*id $	
$TE'\ \$$	id+id*id $	$E\to TE'$
$FT'E'\ \$$	id+id*id $	$T\to FT'$
id$T'E'\ \$$	id+id*id $	$F\to$ id
$T'E'\ \$$	+id*id $	
$E'\ \$$	+id*id $	$T'\to\varepsilon$
$+TE'\ \$$	+id*id $	$E'\to +TE'$
$TE'\ \$$	id*id $	
$FT'E'\ \$$	id*id $	$T\to FT'$
id$T'E'\ \$$	id*id $	$F\to$ id
$T'E'\ \$$	*id $	
$*FT'E'\ \$$	*id $	$T'\to *FT'$
$FT'E'\ \$$	id $	
id$T'E'\ \$$	id $	$F\to$ id
$T'E'\ \$$	$	
$E'\ \$$	$	$T'\to\varepsilon$
$\$$	$	$E'\to\varepsilon$

表驱动的预测分析法

- 输入：一个串*w*和文法*G*的分析表 *M*
- 输出：如果*w*在*L*(*G*)中，输出*w*的最左推导；否则给出错误指示
- 方法：最初，语法分析器的格局如下：输入缓冲区中是*w*$，*G*的开始符号位于栈顶，其下面是$。下面的程序使用预测分析表*M*生成了处理这个输入的预测分析过程

```
设置ip使它指向w的第一个符号，其中ip 是输入指针；
令X=栈顶符号；
while ( X ≠ $ ) {     /* 栈非空 */
        if ( X 等于ip所指向的符号a) 执行栈的弹出操作，将ip向前移动一个位
    置；
        else  if ( X是一个终结符号)  error ( ) ;
        else  if ( M[X，a]是一个报错条目)  error ( ) ;
        else  if ( M[X，a] = X → Y1Y2 … Yk  ) {
                    输出产生式 X → Y1Y2 … Yk  ；
                  弹出栈顶符号；
                   将Yk，Yk-1 …，Y1 压入栈中，其中Y1位于栈顶。
        }
        令X=栈顶符号
}
```

递归的预测分析法vs.非递归的预测分析法

	递归的预测分析法	非递归的预测分析法
程序规模	程序规模较大， 不需载入分析表	主控程序规模较小， 需载入分析表（表较小）
直观性	较好	较差
效率	较低(递归调用)	分析时间大约正比于待分析程序的长度
自动生成	较难	较易

二义性文法的预测分析表冲突处理

- if 语句的文法G(S):

$$S \quad iEtS \mid iEtSeS \mid a$$
$$E \quad b$$

提取左公因子之后，改写成：

$$S \quad iEtSS' \mid a$$
$$S' \quad eS \mid$$
$$E \quad b$$

二义性文法的预测分析表冲突处理

- 改造后if 语句的文法G(S):

  $S \quad iEtSS' \mid a$

  $S' \quad eS \mid$

  $E \quad b$

SELECT(S' $\rightarrow$) = FOLLOW(S') = { e, $\$$ }

SELECT(S' $\rightarrow$ eS) = FIRST(eS) = { e }

非终结符	输入符号					
	a	b	e	i	t	$\$$
S	$S \rightarrow a$			$S \rightarrow iEtSS$		
S'			$S' \rightarrow eS$ $S' \rightarrow$			$S' \rightarrow$
E		$E \rightarrow b$				

- 通过消歧规则消解冲突，仍然可以使用预测分析法对该文法进行分析。
- 但不存在普遍适用的规则来消除任意文法预测分析表中的冲突条目。

预测分析法的实现步骤

1. 改造文法：消除二义性、消除左递归、提取左公因子
2. 判断是否LL(1)文法：具有相同左部的产生式的SELECT集互不相交。
   （1）求每个非终结符的FIRST集和FOLLOW集
   （2）求每个候选式的FIRST集
   （3）求具有相同左部产生式的SELECT集
3. 若是LL(1)文法，构造预测析表，实现预测分析器。
4. 若不是 LL(1)文法，说明文法的复杂性超过预测分析法的分析能力。
   - 附加新的“信息”，处理冲突表项，依然可以采用预测分析法。
   - 无法处理冲突，建议采用自底向上分析方法。

判定LL(1)文法

练习：判断文法是不是LL(1)文法

S->aA | BC　　A->bA|c　　B->Ac|　　C->d |

参考答案：

FIRST(S) = {*a, b, c, d,* } ; 因 *FIRST*(*B*) 且 *FIRST*(C)

FIRST(A) = {b,c}; FIRST(B) = { *b*,*c*, } ; FIRST(C)={*d*, };

FOLLOW(S)={$}; FOLLOW(A)={*c*,$}; FOLLOW(B) = {*d*, $};

FOLLOW(C)={$};

SELECT(S->*a*A)={*a*}; SELECT(S->BC) = { *b, c, d,* $);

SELECT(*A*->*bA*)={*b*}; SELECT(*A*->*c*)={*c*};

SELECT(*B*->*Ac*)={*b*,*c*}; SELECT(*B*->)={*d*,$};

SELECT(*C*->*d*)={*d*}; SELECT(*C*->)={$};

课后练习

- 练习: 请先改造文法，消除左递归和左公因子。然后再求所有非终结符的FIRST集，FOLLOW集，求所有产生式的SELECT集并画出预测分析表。
  - ***S　SAB|ab　A　Ba|　B　Db|D　D　d|***

参考答案：

FIRST(S) = {a}; FIRST(S')=FIRST(A)= {a, b, d, };
FIRST(B) = {b, d, };
FIRST(B') = {b, }; FIRST(D)={ d, } ;
FOLLOW(A) = FOLLOW(B) = FOLLOW(B') = FOLLOW(D)= { a, b, d, $ };
FOLLOW(S) = FOLLOW(S') = { $ };
SELECT(S'->ABS') = {a, b, d, $}; SELECT(A->Ba) = {a, b, d};
SELECT(B->DB') = {a, b, d, $}

4.2.4 预测分析中的错误处理

- 错误处理两个任务
  - 检测到错误并报错——位置和类型
  - 错误恢复—— 快速
- 两种情况下可以检测到语法错误
  - 栈顶的终结符和当前输入符号不匹配
  - 栈顶非终结符*A*与当前输入符号*a*在预测分析表对应项中的信息为空

预测分析中的错误恢复

- 恐慌模式恢复(紧急恢复)
  - 情况1：如果终结符在栈顶而不能匹配，弹出此终结符（当做已经匹配成功了），继续分析栈中符号；
  - 情况2：如果M[*A*, *a*]为空，则忽略输入中的一些符号，直至遇到同步符号(*synchronizing token*)，弹出*A*；或M[*A*, *a*]包含产生式，继续分析。
  - 同步符号的设置：
    - 可以把 *FOLLOW(A)*中符号设置为同步符号，遇到*FOLLOW(A)*中的符号，将*A*弹出(当做A已经推导成功了)，继续分析栈中符号。
    - 可以把较高层构造的开始符号设置为较低层构造对应非终结符的同步符号，遇到这些符号时，将*A*弹出，继续分析栈中符号。
  - 如果*A*可以生成空串，可以把推导出 的产生式当作错误处理的默认选择，好处是无需设计同步符号，实现简单，代价是推迟错误的发现时机

例

非终结符	输入符号					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$	synch	synch
E'		$E' \to +TE'$			$E' \to \varepsilon$	
$E' \to \varepsilon$		synch			synch	synch
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	synch
$T' \to \varepsilon$						

X	FOLLOW(X)
E	$)
E'	$)
T	+) $
T'	+) $
F	* +) $

*Synch*表示根据相应非终结符的*FOLLOW*集得到的同步词法单元

- 紧急错误恢复
  - 如果M[*A*, *a*]是空，检测到错误，忽略输入符号*a*
  - 如果M[*A*, *a*]是*synch*，则弹出栈顶的非终结符*A*，继续分析后面的语法成分
  - 如果M[*A*, *a*]包含产生式，恢复对*A*的分析(*a* 在FIRST(A)的情况)
  - 如果栈顶的终结符和输入符号不匹配，则弹出栈顶的终结符

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE'			E→TE'	synch	synch
E'		E'→+TE'			E'→ε	
E'→ε						
T	T→FT'	synch		T→FT'	synch	synch
T'		T'→ε	T'→*FT'		T'→ε	
T'→ε		synch	synch		synch	synch
F	F→id			F→(E)		

栈	剩余输入	
E $	+id*+id $	
ignore +		
E $	id*+id $	id在FIRST(A)中
TE' $	id*+id $	
FT'E' $	id*+id $	
idT'E' $	id*+id $	
T'E' $	*+id $	
*FT'E' $	*+id $	
FT'E' $	+id $	synch, F出栈
T'E' $	+id $	
E' $	+id $	
+TE' $	+id $	
TE' $	id $	

小结

1.自顶向下分析法和自底向上分析法分别寻找输入串的最左推导和最左归约

2.自顶向下分析会遇到二义性问题、回溯问题、左递归引起的无穷推导问题，需对文法进行改造：消除二义性、消除左递归、提取左公因子

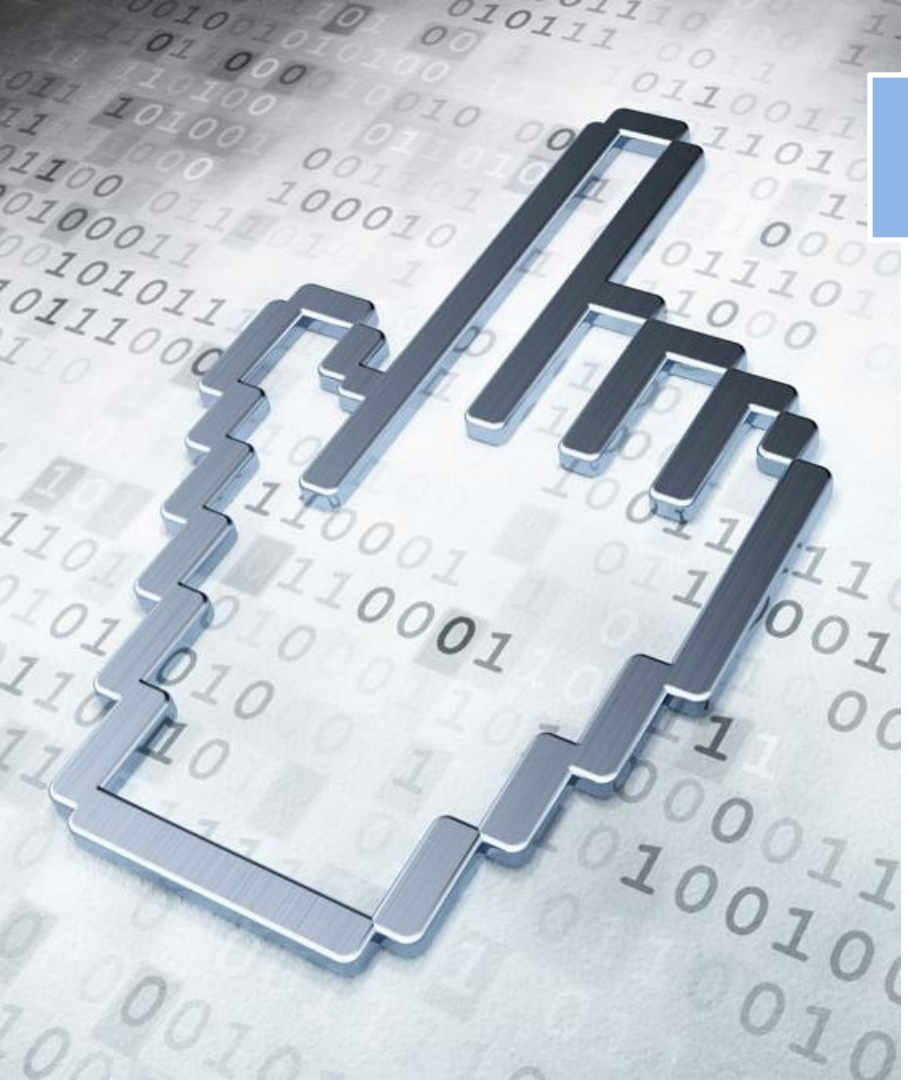
3. *LL*(1)文法是一类可以进行确定分析的文法，利用SELECT集或FIRST集和FOLLOW集可以判定某个上下文无关文法是否为*LL*(1)文法

小结

4. *LL*(1)文法可以用预测分析法也称为*LL*(1)分析法进行分析，对应的分析器称为预测分析器。此方法的核心是构造文法的预测分析表。
5. 递归下降分析法根据各个候选式的结构为每个非终结符编写一个子程序。
6. 表驱动的预测分析维护一个栈，保存尚未分析的句型子串，通过栈顶符号和当前输入符号决定分析器的下一步动作。
7. 非*LL*(1)文法不能使用预测分析进行语法分析，可以考虑采用自底向上的语法分析。

课程主要内容

1．绪论	（2学时）
2. 语言及其文法	（2学时）
3．词法分析	（3学时）
4．语法分析	（9学时）
5．语法制导翻译	（6学时）
6．中间代码生成	（7学时）
7．运行时的存贮组织	（3学时）
8．代码优化	（6学时）
9．代码生成	（2学时）



提纲