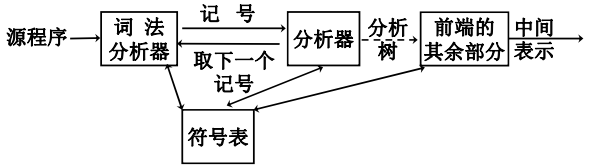
第3章 语法分析



- 本章内容
  - 上下文无关文法
  - 自上而下分析和自下而上分析
  - 围绕分析器的自动生成展开

3.1 上下文无关文法

3.1.1 上下文无关文法的定义

- 正规式能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：$a\,(ba)^5$, $a\,(ba)^*$

- 正规式不能用于描述配对或嵌套的结构

  例1：配对括号串的集合

  例2：{wcw | w是a和b的串}

3.1 上下文无关文法

- 上下文无关文法是四元组（V_T, V_N, S, P）

  V_T： 终结符集合

  V_N： 非终结符集合

  S： 开始符号

  P： 产生式集合， 产生式形式 ： $A \to \alpha$

- 例 ({id, +, ∗, –, (,)}, {*expr, op*}, *expr*, *P*)

  $expr \to expr\ op\ expr$　　$expr \to (expr)$

  $expr \to -\ expr$　　$expr \to$ id

  $op \to +$　　$op \to *$

3.1 上下文无关文法

- 简化表示

  $expr \to expr\ op\ expr \mid (expr) \mid -\ expr \mid$ id

  $op \to + \mid *$

- 简化表示

  $E \to E\ A\ E \mid (E) \mid -E \mid$ id

  $A \to + \mid *$

3.1 上下文无关文法

3.1.2 推导

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

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

- 概念
  - 上下文无关语言、等价的文法、句型
- 记号

  $S \Rightarrow^* \alpha$、 $S \Rightarrow^+ w$

3.1 上下文无关文法

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

- 最左推导

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

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

- 最右推导（规范推导）

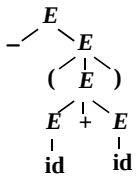
  $E \Rightarrow_{rm} -E \Rightarrow_{rm} -(E) \Rightarrow_{rm} -(E + E)$

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

3.1 上下文无关文法

3.1.3 分析树

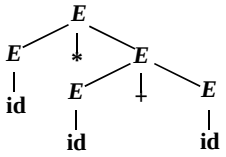
- 例 $E \to E + E \mid E * E \mid (E) \mid - E \mid \text{id}$



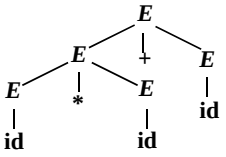
3.1 上下文无关文法

3.1.4 二义性

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



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



3.2 语言和文法

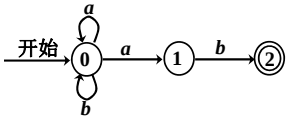
- 文法的优点
  - 文法为语言给出了精确的、易于理解的语法规范
  - 自动产生高效的分析器
  - 可以给语言定义出层次结构
  - 以文法为基础的语言的实现便于语言的修改
- 文法的问题
  - 文法只能描述编程语言的大部分语法，而不是所有的语法

3.2 语言和文法

3.2.1 正规式和上下文无关文法的比较

- 正规式

  $(a|b)^*ab$



- 文法

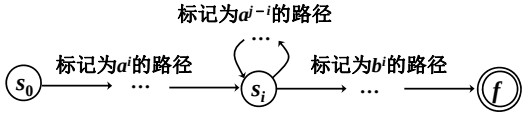
  $A_0 \to a\,A_0 \mid b\,A_0 \mid a\,A_1$
  $A_1 \to b\,A_2$
  $A_2 \to \varepsilon$

3.2 语言和文法

- 例 $L=\{a^n b^n \mid n \geq 1\}$

  $S \to aSb \mid ab$

  L是不能用正规式描述的语言的一个范例
  - 若存在接受L的DFA D，状态数为k个
  - 设D读完$\varepsilon, a, aa, \ldots, a^k$ 分别到达状态$s_0, s_1, \ldots, s_k$
  - 至少有两个状态相同，例如是s_i和s_j，则$a^i b^i$属于L



3.2 语言和文法

3.2.2 分离词法分析器理由

- 为什么要用正规式定义词法
  - 词法规则非常简单，不必用上下文无关文法
  - 对于词法记号，采用正规式描述简洁且易于理解
  - 从正规式构造出的词法分析器效率高

3.2 语言和文法

- 从软件工程角度看，词法分析和语法分析的分离有如下好处
  - 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

3.2 语言和文法

- 能否把词法分析并入到语法分析中，直接从字符流进行语法分析？
  - 若把词法分析和语法分析合在一起，则必须将语言的注释和空白的规则反映在文法中，文法将大大复杂
  - 注释和空白由自己来处理的分析器，比注释和空白已由词法分析器删除的分析器要复杂得多

3.2 语言和文法

3.2.3 验证文法产生的语言

$G: S \rightarrow (S)S \mid \varepsilon$　$L(G)$ = 配对的括号串的集合

3.2 语言和文法

3.2.3 验证文法产生的语言

$G: S \rightarrow (S)S \mid \varepsilon$　$L(G)$ = 配对的括号串的集合

- 按推导步数进行归纳：推出的是配对括号串
  - 归纳基础：$S \Rightarrow \varepsilon$
  - 归纳假设：少于n步的推导都产生配对的括号串
  - 归纳步骤：n步的最左推导如下：

$$S \Rightarrow (S)S \Rightarrow^* (x)S \Rightarrow^* (x)y$$

3.2 语言和文法

3.2.3 验证文法产生的语言

$G: S \rightarrow (S)S \mid \varepsilon$　$L(G)$ = 配对的括号串的集合

- 按串长进行归纳：配对括号串可由S推出
  - 归纳基础：$S \Rightarrow \varepsilon$
  - 归纳假设：长度小于$2n$的都可以从S推导出来
  - 归纳步骤：考虑长度为$2n(n \geq 1)$的$w = (x)y$

$$S \Rightarrow (S)S \Rightarrow^* (x)S \Rightarrow^* (x)y$$

3.2 语言和文法

3.2.4 适当的表达式文法

- 用一种层次观点看待表达式

  $\underline{\mathbf{id} * \mathbf{id} * (\mathbf{id}+\mathbf{id})} + \underline{\mathbf{id} * \mathbf{id}} + \underline{\mathbf{id}}$

  $\underline{\mathbf{id}} * \underline{\mathbf{id}} * \underline{(\mathbf{id}+\mathbf{id})}$
- 文法

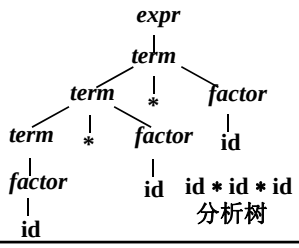
  $expr \rightarrow expr + term \mid term$

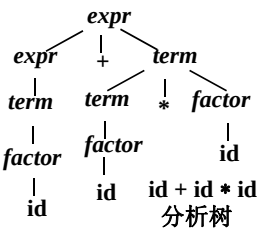
  $term \rightarrow term * factor \mid factor$

  $factor \rightarrow \mathbf{id} \mid (expr)$

3.2 语言和文法

expr → *expr* + *term* | *term*
term → *term* ∗ *factor* | *factor*
factor → id | (*expr*)





3.2 语言和文法

3.2.5 消除二义性

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*

3.2 语言和文法

- 无二义的文法

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*

3.2 语言和文法

3.2.6 消除左递归

- 文法左递归 $A \Rightarrow^{+} A\alpha$
- 直接左递归 $A \rightarrow A\alpha \mid \beta$
  – 串的特点 $\beta\alpha \ldots \alpha$
- 消除直接左递归
  $A \rightarrow \beta A'$
  $A' \rightarrow \alpha A' \mid \varepsilon$

3.2 语言和文法

- 例 算术表达文法
  $E \rightarrow E + T \mid T$ （ $T + T \ldots + T$ ）
  $T \rightarrow T * F \mid F$ （ $F * F \ldots * F$ ）
  $F \rightarrow (E) \mid \text{id}$

消除左递归后文法
  $E \rightarrow TE'$
  $E' \rightarrow +TE' \mid \varepsilon$
  $T \rightarrow FT'$
  $T' \rightarrow *FT' \mid \varepsilon$
  $F \rightarrow (E) \mid \text{id}$

3.2 语言和文法

- 非直接左递归
  $S \rightarrow Aa \mid b$
  $A \rightarrow Sd \mid \varepsilon$
- 先变换成直接左递归
  $S \rightarrow Aa \mid b$
  $A \rightarrow Aad \mid bd \mid \varepsilon$
- 再消除左递归
  $S \rightarrow Aa \mid b$
  $A \rightarrow bdA' \mid A'$
  $A' \rightarrow adA' \mid \varepsilon$

3.2 语言和文法

3.2.7 提左因子

- 有左因子的文法
  $A \to \alpha\beta_1 \mid \alpha\beta_2$
- 提左因子
  $A \to \alpha A'$
  $A' \to \beta_1 \mid \beta_2$

3.2 语言和文法

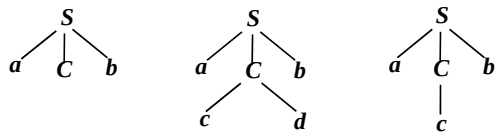
- 例 悬空*else*的文法
  stmt → **if *expr* then *stmt* else *stmt***
  | if *expr* then *stmt*
  | other

  提左因子
  stmt → **if *expr* then *stmt optional_else_part***
  | other
  optional_else_part → **else *stmt***
  | ε

3.3 自上而下分析

3.3.1 自上而下分析的一般方法

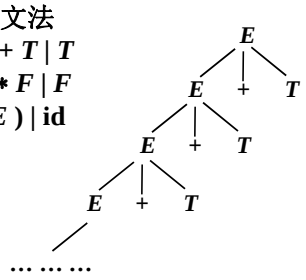
- 例 文法 $S \to aCb \quad C \to cd \mid c$
  为输入串$w = acb$建立分析树



不能处理左递归

3.3 自上而下分析

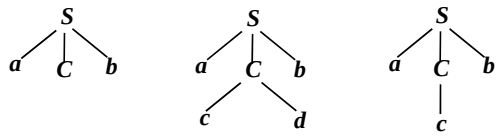
- 不能处理左递归的例子
  算术表达文法
  $E \to E + T \mid T$
  $T \to T * F \mid F$
  $F \to (E) \mid \mathbf{id}$



3.3 自上而下分析

3.3.1 自上而下分析的一般方法

- 例 文法 $S \to aCb \quad C \to cd \mid c$
  为输入串$w = acb$建立分析树

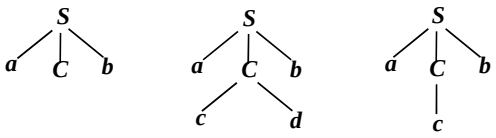


不能处理左递归，需要使用复杂的回溯技术

3.3 自上而下分析

3.3.1 自上而下分析的一般方法

- 例 文法 $S \to aCb \quad C \to cd \mid c$
  为输入串$w = acb$建立分析树

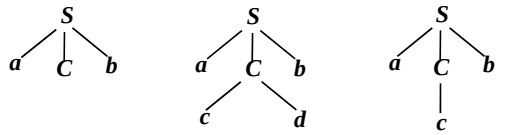


不能处理左递归，需要使用复杂的回溯技术，
回溯导致分析工作推倒重来

3.3 自上而下分析

3.3.1 自上而下分析的一般方法

- 例 文法 $S \to aCb \quad C \to cd \mid c$
  为输入串$w = acb$建立分析树

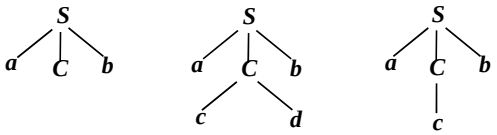


不能处理左递归，需要使用复杂的回溯技术，回溯导致分析工作推倒重来，难以报告出错的确切位置

3.3 自上而下分析

3.3.1 自上而下分析的一般方法

- 例 文法 $S \to aCb \quad C \to cd \mid c$
  为输入串$w = acb$建立分析树



不能处理左递归，需要使用复杂的回溯技术，回溯导致分析工作推倒重来，难以报告出错的确切位置，效率低

3.3 自上而下分析

3.3.2 LL(1)文法

- 对文法加什么样的限制可以保证没有回溯？

- 先定义两个和文法有关的函数
  - FIRST(α) = {$a \mid \alpha \Rightarrow^* a\ldots, a \in V_T$}
    特别是，$\alpha \Rightarrow^* \varepsilon$时，规定$\varepsilon \in$ FIRST(α)

  对A的任何两个不同的选择α_i和α_j，希望有
  FIRST(α_i) ∩ FIRST(α_j) = ∅
  若FIRST(α_i) 或 FIRST(α_j)含ε，还需增加条件

3.3 自上而下分析

3.3.2 LL(1)文法

- 对文法加什么样的限制可以保证没有回溯？

- 先定义两个和文法有关的函数
  - FIRST(α) = {$a \mid \alpha \Rightarrow^* a\ldots, a \in V_T$}
    特别是，$\alpha \Rightarrow^* \varepsilon$时，规定$\varepsilon \in$ FIRST(α)

  - FOLLOW(A) = {$a \mid S \Rightarrow^* \ldots Aa\ldots$，$a \in V_T$}
    如果A是某个句型的最右符号，那么结束标记$属于FOLLOW($A$)

3.3 自上而下分析

- LL(1)文法
  任何两个产生式$A \to \alpha \mid \beta$都满足下列条件：
  - FIRST(α)∩FIRST(β) = ∅
  - 若$\beta \Rightarrow^* \varepsilon$，那么FIRST($\alpha$)∩FOLLOW($A$) = ∅

  例如, 对于下面的文法，面临$a\ldots$时不知用什么规则
  $S \to A\ B$
  $A \to a\ b \mid \varepsilon \quad a \in$ FIRST(ab)∩FOLLOW(A)
  $B \to a\ C$
  $C \to \ldots$

3.3 自上而下分析

- LL(1)文法
  任何两个产生式$A \to \alpha \mid \beta$都满足下列条件：
  - FIRST(α)∩FIRST(β) = ∅
  - 若$\beta \Rightarrow^* \varepsilon$，那么FIRST($\alpha$)∩FOLLOW($A$) = ∅

- LL(1)文法有一些明显的性质
  - 没有公共左因子
  - 不是二义的
  - 不含左递归

3.3 自上而下分析

- 例 $E \rightarrow TE'$
  $E' \rightarrow +TE' \mid \varepsilon$
  $T \rightarrow FT'$
  $T' \rightarrow *FT' \mid \varepsilon$
  $F \rightarrow (E) \mid id$

FIRST(*E*) = FIRST(*T*) = FIRST(*F*) = { (, id }
FIRST(*E* ′) = {+, ε}
FRIST(*T* ′) = {*, ε}
FOLLOW(*E*) = FOLLOW(*E* ′) = {), $}
FOLLOW(*T*) = FOLLOW (*T* ′) = {+,), $}
FOLLOW(*F*) = {+, *,), $}

3.3 自上而下分析

3.3.3 递归下降的预测分析

- 为每一个非终结符写一个分析过程
- 这些过程可能是递归的

- 例

type → *simple*
| ↑ id
| array [*simple*] of *type*
simple → integer
| char
| num dotdot num

3.3 自上而下分析

一个辅助过程

```
void match (terminal t) {
    if (lookahead == t) lookahead = nextToken( );
    else error( );
}
```

3.3 自上而下分析

```
void type( ) {
  if ( (lookahead == integer) || (lookahead == char) ||
                                 (lookahead == num) )
      simple( );
  else if ( lookahead == '↑' ) { match('↑'); match(id);}
  else if (lookahead == array) {
      match(array); match( '[' ); simple( );
      match( ']' ); match(of ); type( );
  }
  else error( );
}
```

type → *simple*
| ↑ id
| array [*simple*] of *type*

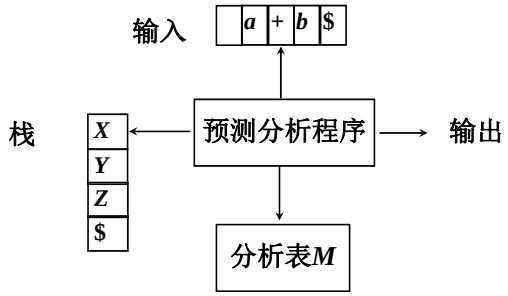
3.3 自上而下分析

```
void simple( ) {
  if ( lookahead == integer) match(integer);
  else if (lookahead == char) match(char);
  else if (lookahead == num) {
      match(num); match(dotdot); match(num);
  }
  else error( );
}
```

simple → integer
| char
| num dotdot num

3.3 自上而下分析

3.3.4 非递归的预测分析



3.3 自上而下分析

分析表的一部分

非终结符	输 入 符 号			
	id	+	*	. . .
E	$E \to TE'$			
E'		$E' \to +TE'$		
T	$T \to FT'$			
T'		$T' \to \varepsilon$	$T' \to *FT'$	
F	$F \to$ id			

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输 入	输 出
$E	id * id + id$	

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输 入	输 出
$E	id * id + id$	
$E'T	id * id + id$	$E \to TE'$

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输 入	输 出
$E	id * id + id$	
$E'T	id * id + id$	$E \to TE'$
$E'T'F	id * id + id$	$T \to FT'$

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输 入	输 出
$E	id * id + id$	
$E'T	id * id + id$	$E \to TE'$
$E'T'F	id * id + id$	$T \to FT'$
$E'T'id	id * id + id$	$F \to$ id

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输 入	输 出
$E	id * id + id$	
$E'T	id * id + id$	$E \to TE'$
$E'T'F	id * id + id$	$T \to FT'$
$E'T'id	id * id + id$	$F \to$ id
$E'T'	* id + id$	匹配id

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输　入	输　出
$*E*	id * id + id$	
$*E*′*T*	id * id + id$	*E* → *TE*′
$*E*′*T*′*F*	id * id + id$	*T* → *FT*′
$*E*′*T*′ id	id * id + id$	*F* → id
$*E*′*T*′	* id + id$	匹配id
$*E*′*T*′*F* *	* id + id$	*T*′ → **FT*′

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输　入	输　出
$*E*	id * id + id$	
$*E*′*T*	id * id + id$	*E* → *TE*′
$*E*′*T*′*F*	id * id + id$	*T* → *FT*′
$*E*′*T*′ id	id * id + id$	*F* → id
$*E*′*T*′	* id + id$	匹配id
$*E*′*T*′*F* *	* id + id$	*T*′ → **FT*′
$*E*′*T*′*F*	id + id$	匹配*

3.3 自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输　入	输　出
$*E*	id * id + id$	
$*E*′*T*	id * id + id$	*E* → *TE*′
$*E*′*T*′*F*	id * id + id$	*T* → *FT*′
$*E*′*T*′ id	id * id + id$	*F* → id
$*E*′*T*′	* id + id$	匹配id
$*E*′*T*′*F* *	* id + id$	*T*′ → **FT*′
$*E*′*T*′*F*	id + id$	匹配*
$*E*′*T*′ id	id + id$	*F* → id

3.3 自上而下分析

3.3.5 构造预测分析表

（1）对文法的每个产生式*A* → *α*，执行(2)和(3)

（2）对FIRST(*α*)的每个终结符*a*，

把*A* → *α* 加入*M*[*A*, *a*]

（3）如果ε在FIRST(*α*)中，对FOLLOW(*A*)的每个终结符*b*（包括$），把*A* → *α*加入*M*[*A*, *b*]

（4）*M*中其他没有定义的条目都是error

3.3 自上而下分析

多重定义的条目

非终结符	输　入　符　号			
	other	*b*	else	. . .
stmt	*stmt* → other			
e_part			*e_part*→ else *stmt* *e_part* → ε	
expr		*expr* → *b*		

3.3 自上而下分析

消去多重定义

非终结符	输　入　符　号			
	other	*b*	else	. . .
stmt	*stmt* → other			
e_part			*e_part*→ else *stmt*	
expr		*expr* → *b*		

3.3 自上而下分析

3.3.6 预测分析的错误恢复

1、先对编译器的错误处理作一个概述

- 词法错误，如标识符、关键字或算符的拼写错
- 语法错误，如算术表达式的括号不配对
- 语义错误，如算符和运算对象之间的类型不匹配
- 逻辑错误，如无穷的递归调用

3.3 自上而下分析

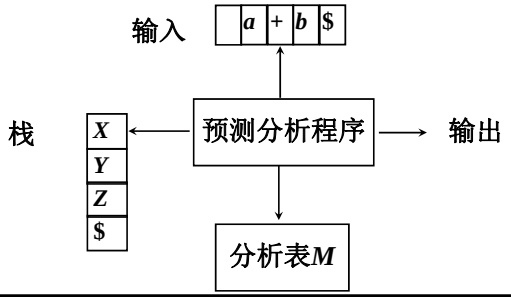
2、分析器对错误处理的基本目标

- 清楚而准确地报告错误的出现，并尽量少出现伪错误
- 迅速地从每个错误中恢复过来，以便诊断后面的错误
- 它不应该使处理正确程序的速度降低太多

3.3 自上而下分析

3、非递归预测分析在什么场合下发现错误？

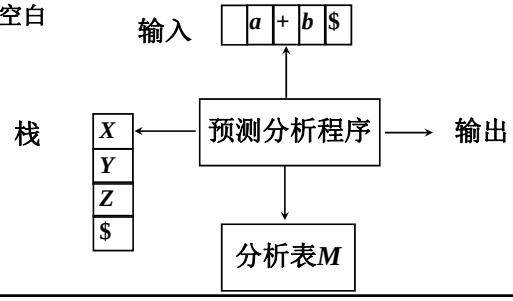
- 栈顶的终结符和下一个输入符号不匹配



3.3 自上而下分析

3、非递归预测分析在什么场合下发现错误？

- 栈顶是非终结符*A*，输入符号是*a*，而*M*[*A* , *a*]是空白



3.3 自上而下分析

4、非递归预测分析：采用紧急方式的错误恢复

发现错误时，分析器每次抛弃一个输入记号，直到输入记号属于某个指定的同步记号集合为止

5、同步

词法分析器当前提供的记号流能构成的语法构造，正是语法分析器所期望的

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合

if *expr* **then** *stmt*

（then是*expr*的一个同步记号）

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合
- 把高层构造的开始符号加到低层构造的同步记号集合中

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合
- 把高层构造的开始符号加到低层构造的同步记号集合中

***a* = *expr*; if …**

(语句的开始符号作为表达式的同步记号，以免表达式出错又遗漏分号时忽略if语句等一大段程序)

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合
- 把高层构造的开始符号加到低层构造的同步记号集合中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合

***a* = *expr*; , if …**

（语句的开始符号作为语句的同步符号，以免多出一个逗号时会把if语句忽略了）

3.3 自上而下分析

6、同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合
- 把高层构造的开始符号加到低层构造的同步记号集合中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合
- 如果非终结符可以产生空串，若出错时栈顶是这样的非终结符，则可以使用推出空串的产生式

3.3 自上而下分析

• 例 栈顶为*T′*，面临id时出错

非终结符	输入符号			
	id	+	*	…
E	*E*→*TE′*			
E′		*E′*→+*TE′*		
T	*T*→*FT′*			
T′	出错	*T′*→ε	*T′*→**FT′*	
…				

3.3 自上而下分析

T'可以产生空串，报错并用$T' \to \varepsilon$

非终结符	输入符号			
	id	+	*	...
E	$E \to TE'$			
E'		$E' \to +TE'$		
T	$T \to FT'$			
T'	出错，用$T' \to \varepsilon$	$T' \to \varepsilon$	$T' \to *FT'$	
...				

3.3 自上而下分析

同步记号集合的选择

- 把FOLLOW(*A*)的所有终结符放入非终结符*A*的同步记号集合
- 把高层结构的开始符号加到低层结构的同步记号集合中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合
- 如果非终结符可以产生空串，若出错时栈顶是这样的非终结符，则可以使用推出空串的产生式
- 如果终结符在栈顶而不能匹配，弹出此终结符

3.4 自下而上分析

3.4.1 归约

- 例 $S \to aABe$
  $A \to Abc \mid b$
  $B \to d$

3.4 自下而上分析

3.4.1 归约

- 例 $S \to aABe$
  $A \to Abc \mid b$
  $B \to d$

abbcde（读入*ab*）

a *b*

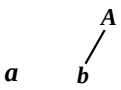
3.4 自下而上分析

3.4.1 归约

- 例 $S \to aABe$
  $A \to Abc \mid b$
  $B \to d$

abbcde

aAbcde（归约）



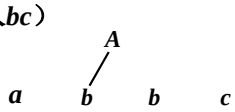
3.4 自下而上分析

3.4.1 归约

- 例 $S \to aABe$
  $A \to Abc \mid b$
  $B \to d$

abbcde

aAbcde（再读入*bc*）



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

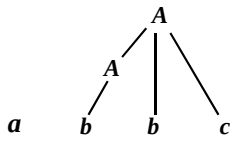
$A \rightarrow Abc \mid b$

$B \rightarrow d$

abbcde

aAbcde

aAde（归约）



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

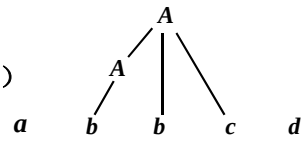
$A \rightarrow Abc \mid b$

$B \rightarrow d$

abbcde

aAbcde

aAde（再读入*d*）



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

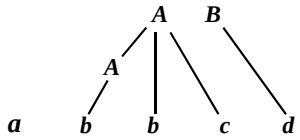
$B \rightarrow d$

abbcde

aAbcde

aAde

aABe（归约）



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

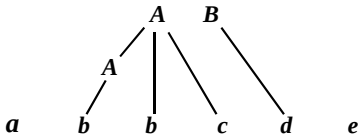
$B \rightarrow d$

abbcde

aAbcde

aAde

aABe(再读入*e*)



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

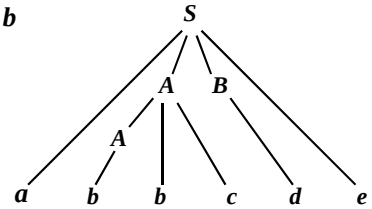
abbcde

aAbcde

aAde

aABe

S（归约）



3.4 自下而上分析

3.4.1 归约

• 例 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

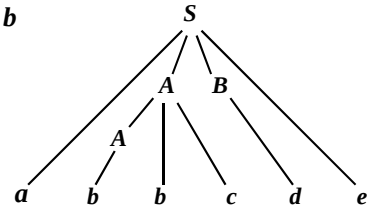
abbcde

aAbcde

aAde

aABe

S



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$

3.4 自下而上分析

3.4.2 句柄

句型的句柄是和某产生式右部匹配的子串，并且，把它归约成该产生式左部的非终结符代表了最右推导过程的逆过程的一步

$S \to aABe$

$A \to Abc \mid b$

$B \to d$

$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$

句柄的右边仅含终结符

如果文法二义，那么句柄可能不唯一

3.4 自下而上分析

- 例 句柄不唯一

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

3.4 自下而上分析

- 例 句柄不唯一

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

$E \Rightarrow_{rm} E * E$

$\Rightarrow_{rm} E * E + E$

$\Rightarrow_{rm} E * E + \mathbf{id}_3$

$\Rightarrow_{rm} E * \mathbf{id}_2 + \mathbf{id}_3$

$\Rightarrow_{rm} \mathbf{id}_1 * \mathbf{id}_2 + \mathbf{id}_3$

3.4 自下而上分析

- 例 句柄不唯一

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

$E \Rightarrow_{rm} E * E$

$\Rightarrow_{rm} E * E + E$

$\Rightarrow_{rm} E * E + \mathbf{id}_3$

$\Rightarrow_{rm} E * \mathbf{id}_2 + \mathbf{id}_3$

$\Rightarrow_{rm} \mathbf{id}_1 * \mathbf{id}_2 + \mathbf{id}_3$

$E \Rightarrow_{rm} E + E$

$\Rightarrow_{rm} E + \mathbf{id}_3$

$\Rightarrow_{rm} E * E + \mathbf{id}_3$

$\Rightarrow_{rm} E * \mathbf{id}_2 + \mathbf{id}_3$

$\Rightarrow_{rm} \mathbf{id}_1 * \mathbf{id}_2 + \mathbf{id}_3$

在句型$E * E + \mathbf{id}_3$中，句柄不唯一

3.4 自下而上分析

3.4.3 用栈实现移进–归约分析

先通过观察

移进–归约分析器在分析输入串$\mathbf{id}_1 * \mathbf{id}_2 + \mathbf{id}_3$时的动作序列

来了解移进–归约分析的工作方式

3.4 自下而上分析

栈	输　入	动　作
$	$\mathbf{id}_1 * \mathbf{id}_2 + \mathbf{id}_3$$	

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \rightarrow$ id归约

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \rightarrow$ id归约
$$E$	$* id_2 + id_3$$	

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \rightarrow$ id归约
$$E$	$* id_2 + id_3$$	移进

3.4 自下而上分析

栈	输　入	动　作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \rightarrow$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进
$$E$*$id_2$	$+ id_3$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进
$$E$*$id_2$	$+ id_3$$	按$E \to$ id归约

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进
$$E$*$id_2$	$+ id_3$$	按$E \to$ id归约
$$E$*$E$	$+ id_3$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进
$$E$*$id_2$	$+ id_3$$	按$E \to$ id归约
$$E$*$E$	$+ id_3$$	移进

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E$*	$id_2 + id_3$$	移进
$$E$*$id_2$	$+ id_3$$	按$E \to$ id归约
$$E$*$E$	$+ id_3$$	移进
$$E$*$E$+	$id_3$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$$	按$E \to$ id归约

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$$	按$E \to$ id归约
$$E*E+E$	$$	按$E \to E+E$归约

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$$	按$E \to$ id归约
$$E*E+E$	$$	按$E \to E+E$归约
$$E*E$	$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$$	按$E \to$ id归约
$$E*E+E$	$$	按$E \to E+E$归约
$$E*E$	$$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$	按$E \to$ id归约
$$E*E+E$	$	按$E \to E+E$归约
$$E*E$	$	按$E \to E*E$归约

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$	按$E \to$ id归约
$$E*E+E$	$	按$E \to E+E$归约
$$E*E$	$	按$E \to E*E$归约
$$E$	$	

3.4 自下而上分析

栈	输 入	动 作
$	$id_1 * id_2 + id_3$$	移进
$ id_1	$* id_2 + id_3$$	按$E \to$ id归约
$$E$	$* id_2 + id_3$$	移进
$$E*$	$id_2 + id_3$$	移进
$$E*id_2$	$+ id_3$$	按$E \to$ id归约
$$E*E$	$+ id_3$$	移进
$$E*E+$	$id_3$$	移进
$$E*E+id_3$	$	按$E \to$ id归约
$$E*E+E$	$	按$E \to E+E$归约
$$E*E$	$	按$E \to E*E$归约
$$E$	$	接受

3.4 自下而上分析

- 要想使用移进–归约方式分析句子，尚需解决一些问题
  - 如何决策选择移进还是归约
  - 进行归约时，确定右句型中将要归约的子串
  - 进行归约时，如何确定选择哪一个产生式

3.4 自下而上分析

3.4.4 移进–归约分析的冲突

1、移进–归约冲突

- 例

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

如果移进–归约分析器处于格局

栈	输入
… if *expr* then *stmt*	else … $

3.4 自下而上分析

2、归约–归约冲突

stmt* → id (*parameter_list*) | *expr* = *expr
parameter_list* → *parameter_list*, *parameter* | *parameter
***parameter* → id**
***expr* → id (*expr_list*) | id**
expr_list* → *expr_list*, *expr* | *expr

由*A*(*I*, *J*)开始的语句

归约成*expr*还是*parameter*？

栈	输入
… id (id	, id)…

3.4 自下而上分析

2、归约–归约冲突

stmt → id (*parameter_list*) | *expr* = *expr*
parameter_list → *parameter_list*, *parameter* | *parameter*
parameter → id
expr → id (*expr_list*) | id
expr_list → *expr_list*, *expr* | *expr*

由*A*(*I*, *J*)开始的语句(词法分析查符号表, 区分第一个id)

栈	输入
… procid (id	, id)…

需要修改上面的文法

3.5 LR分析器

本节介绍LR(*k*)分析技术

- 特点
  - 适用于一大类上下文无关文法
  - 效率高
- 介绍构造LR分析表的三种技术
  - 详细介绍简单的LR方法（简称SLR）
  - 概要介绍规范的LR方法
  - 概要介绍向前看的LR方法（简称LALR）

3.5 LR分析器

3.5.1 构造SLR分析表

- 例1

(1) *E* → *E* + *T*　　(4) *T* → *F*
(2) *E* → *T*　　(5) *F* → (*E*)
(3) *T* → *T* * *F*　　(6) *F* → id

若分析器的格局是

栈	输入
…*E* + *T*	+…

则肯定句柄已经在栈顶。但仅从栈顶符号*T* 不能判断句柄是*T* 还是*E* + *T*

3.5 LR分析器

- 例2

若分析器的格局是

栈	输入
… procid (id	, id) …

则知道栈顶的id就是句柄。但不看到栈顶向下的第3个符号，就不知把栈顶的id归约成哪个非终结符

- 思考

栈中文法符号包含的信息偏少；
需要重新设计栈，让栈中每个符号能概括栈中它下面部分所含的信息

3.5 LR分析器

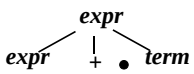
术语：LR(0)项目（简称项目）

- 在右部的某个地方加点的产生式
- 加点的目的是用来表示分析过程中的状态

3.5 LR分析器

术语：LR(0)项目（简称项目）

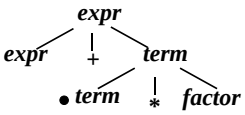
- 在右部的某个地方加点的产生式
- 加点的目的是用来表示分析过程中的状态



3.5 LR分析器

术语：LR(0)项目（简称项目）

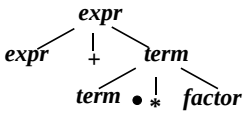
- 在右部的某个地方加点的产生式
- 加点的目的是用来表示分析过程中的状态



3.5 LR分析器

术语：LR(0)项目（简称项目）

- 在右部的某个地方加点的产生式
- 加点的目的是用来表示分析过程中的状态



3.5 LR分析器

术语：LR(0)项目（简称项目）

- 在右部的某个地方加点的产生式
- 加点的目的是用来表示分析过程中的状态

• 例 $A \to XYZ$对应有四个项目

$A \to \cdot XYZ$

$A \to X \cdot YZ$

$A \to XY \cdot Z$

$A \to XYZ \cdot$

• 例 $A \to \varepsilon$只有一个项目和它对应

$A \to \cdot$

3.5 LR分析器

构造SLR分析表的两大步骤

1、从文法构造识别活前缀的DFA

2、从上述DFA构造分析表

3.5 LR分析器

1、从文法构造识别活前缀的DFA

1）拓广文法

$E \to E + T \mid T$

$T \to T * F \mid F$

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

3.5 LR分析器

1、从文法构造识别活前缀的DFA

1）拓广文法

$E' \to E$

$E \to E + T \mid T$

$T \to T * F \mid F$

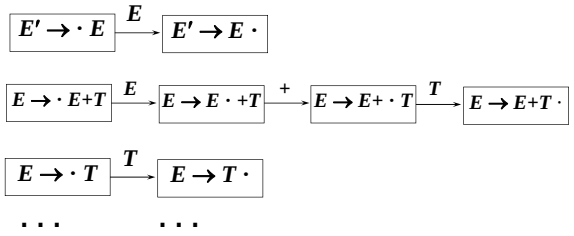
$F \to (E) \mid \mathbf{id}$

3.5 LR分析器

1、从文法构造识别活前缀的DFA

2）构造每个产生式的状态转换图

$E' \to E$　$E \to E + T$　$E \to T$　. . . . . .

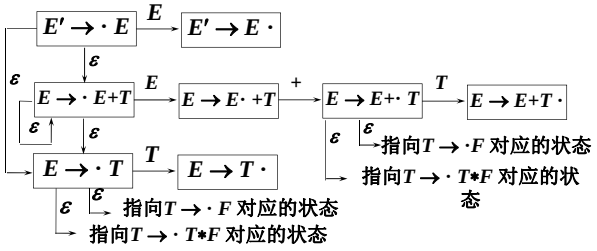


3.5 LR分析器

1、从文法构造识别活前缀的DFA

3）增加ε转换，得到完整NFA的状态转换图

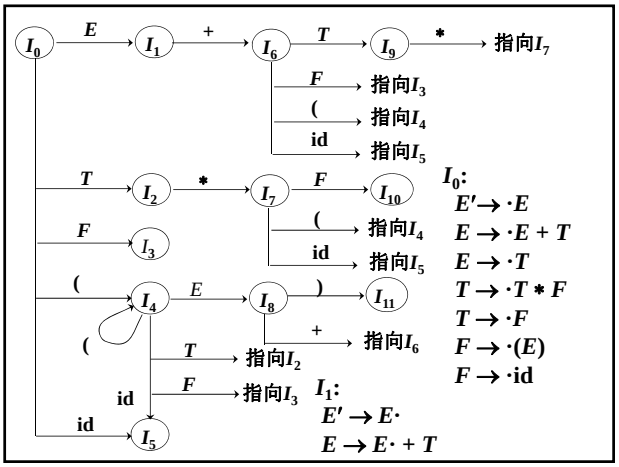
$E' \to E$　$E \to E + T$　$E \to T$　. . . . . .

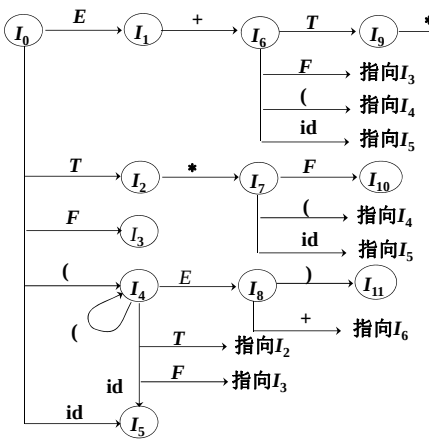


3.5 LR分析器

1、从文法构造识别活前缀的DFA

4）将NFA用子集构造法确定化成DFA





把每个状态都作为接受状态，则这是接受文法活前缀的DFA

3.5 LR分析器

1、从文法构造识别活前缀的DFA

- 活前缀
  - 指文法G的某个右句型的一个前缀，该前缀不超过该右句型的最右句柄的右端
  - 在活前缀的右端添若干终结符可使它成为右句型
  - 由一个拓广文法构造的这种DFA是接受该文法活前缀的DFA

3.5 LR分析器

构造SLR分析表的两大步骤

1、从文法构造识别活前缀的DFA

2、从DFA构造分析表

$E' \rightarrow E$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid \mathbf{id}$

2、从DFA构造分析表 DFA的状态转换表

状态	id	+	*	(	)	$	E	T	F
0	5			4			1	2	3
1		6							
2			7						
3									
4	5			4			8	2	3
5									
6	5			4				9	3
7	5			4					10
8		6			11				
9			7						
10									
11									

2、从DFA构造分析表 DFA的状态转换表

(1) 将状态转换表分成终结符和非终结符两部分

状态	id	+	*	(	)	$	E	T	F
0	5			4			1	2	3
1		6							
2			7						
3									
4	5			4			8	2	3
5									
6	5			4				9	3
7	5			4					10
8		6			11				
9			7						
10									
11									

2、从DFA构造分析表 DFA的状态转换表

(2)在终结符部分中的数字前加上s，表示状态

状态	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6							
2			s7						
3									
4	s5			s4			8	2	3
5									
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9			s7						
10									
11									

2、从DFA构造分析表 DFA的状态转换表

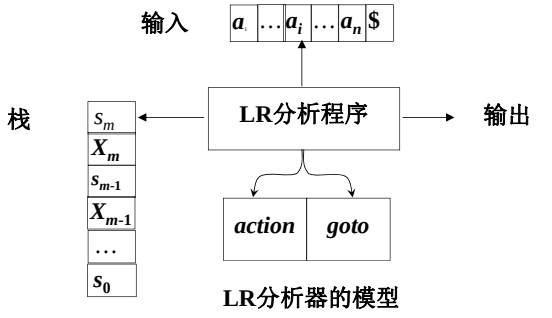
(3)在终结符表中增加归约信息

(4) 空白条目都表示出错

状态	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

3.5 LR分析器

3.5.2 LR分析算法



LR分析器的模型

3.5 LR分析器

• 例 $E \to E + T \mid E \to T$
$T \to T * F \mid T \to E$
$F \to (E) \mid F \to \mathbf{id}$

状态	动 作						转 移		
	id	+	*	(	)	$	*E*	*T*	*F*
0	*s*5			*s*4			1	2	3
1		*s*6				*acc*			
2		*r*2	*s*7		*r*2	*r*2			
3		*r*4	*r*4		*r*4	*r*4			
4	*s*5			*s*4			8	2	3

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约
0 *T* 2	* id + id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约
0 *T* 2	* id + id $	移进

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 *F* 3	* id + id $	按$T \rightarrow F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约
0 *T* 2 * 7 *F* 10	+ id $	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约
0 *T* 2 * 7 *F* 10	+ id $	按$T \to T*F$归约

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约
0 *T* 2 * 7 *F* 10	+ id $	按$T \to T*F$归约
. . .	. . .	. . .

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约
0 *T* 2 * 7 *F* 10	+ id $	按$T \to T*F$归约
. . .	. . .	. . .
0 *E* 1	$	

3.5 LR分析器

栈	输 入	动 作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \to$ id归约
0 *F* 3	* id + id $	按$T \to F$归约
0 *T* 2	* id + id $	移进
0 *T* 2 * 7	id + id $	移进
0 *T* 2 * 7 id 5	+ id $	按$F \to$ id归约
0 *T* 2 * 7 *F* 10	+ id $	按$T \to T*F$归约
. . .	. . .	. . .
0 *E* 1	$	接受

3.5 LR分析器

- LR分析方法和LL分析方法的比较

在下面的推导中，最后一步用的是$A \to l\beta$

LR(1)决定用该产生式的位置

$$S \Rightarrow_{rm} \cdots \Rightarrow_{rm} \gamma A b w \Rightarrow_{rm} \gamma l \beta b w$$

LL(1)决定用该产生式的位置

3.5 LR分析器

3.5.3 其他LR分析表构造概述

- SLR技术能力有限

$S \to V = E$
$S \to E$
$V \to * E$
$V \to \text{id}$
$E \to V$

I_0:
$S' \to \cdot S$
$S \to \cdot V = E$
$S \to \cdot E$
$V \to \cdot * E$
$V \to \cdot \text{id}$
$E \to \cdot V$

V

I_2:
$S \to V \cdot = E$
$E \to V \cdot$

I_2中第一个项目使得 $action[2, =] = s6$

I_2中第二个项目使得 $action[2, =]$为按$E \to V$归约，因为=是E的一个后继符

=是E的一个后继符：
$S\$ \Rightarrow V = E\$ \Rightarrow * E = E\$$

3.5 LR分析器

3.5.3 其他LR分析表构造概述

- SLR技术能力有限

$S \to V = E$
$S \to E$
$V \to * E$
$V \to \text{id}$
$E \to V$

I_0:
$S' \to \cdot S$
$S \to \cdot V = E$
$S \to \cdot E$
$V \to \cdot * E$
$V \to \cdot \text{id}$
$E \to \cdot V$

V

I_2:
$S \to V \cdot = E$
$E \to V \cdot$

I_2中第一个项目使得 $action[2, =] = s6$

I_2中第二个项目使得 $action[2, =]$为按$E \to V$归约，因为=是E的一个后继符

在所关注场合E 的后继是$：
$S\$ \Rightarrow V = E\$ \Rightarrow * E = E\$$
$S\$ \Rightarrow E\$ \Rightarrow V\$$

3.5 LR分析器

- 改进SLR(1)分析技术的办法
  - 把LR(0)项目拓展为LR(1)项目
    让LR(0)项目带上搜索符，成为如下形式
    $[A \to \alpha \cdot \beta, a]$
  - 形式为$[A \to \alpha \cdot, a]$的项目告知在面临a时按产生式$A \to \alpha$进行归约
- 按该思路构造的分析表称为规范的LR分析表
  - 一个文法的LR(1)项目比它的LR(0)项目多得多，相应的状态转换图也大得多
- LALR分析技术是一种折中的技术

3.5 LR分析器

- LR分析技术富有吸引力的原因
  - LR分析技术能够用来识别所有能用上下文无关文法写出的编程语言构造
  - LR分析技术是最一般的无回溯移进-归约技术，能和其他移进-归约技术一样有效地实现
  - LR分析技术能分析的文法类是预测分析或者说LL方法能分析的文法类的真超集
  - 在自左向右扫描输入的前提下，LR分析器能尽可能快地发现语法错误

3.5 LR分析器

3.5.4 非二义且非LR的上下文无关文法

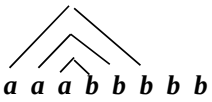
若自左向右扫描的移进–归约分析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR的

语言$L = \{ww^R \mid w \in (a \mid b)^*\}$的文法
$S \to aSa \mid bSb \mid \varepsilon$
不是LR的 $abababbbaba$

语言$L = \{wcw^R \mid w \in (a \mid b)^*\}$的文法
$S \to aSa \mid bSb \mid c$
是LR的 $ababbcbbaba$

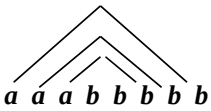
3.5 LR分析器

- 例 为语言$L = \{ a^m b^n \mid n > m \geq 0 \}$写三个文法，它们分别是LR(1)的、二义的和非二义且非LR(1)的
- LR(1)文法：$S \to AB \quad A \to aAb \mid \varepsilon \quad B \to Bb \mid b$



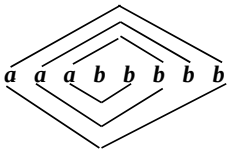
3.5 LR分析器

- 例 为语言$L = \{ a^m b^n \mid n > m \geq 0 \}$写三个文法，它们分别是LR(1)的、二义的和非二义且非LR(1)的
- LR(1)文法：$S \to AB \quad A \to aAb \mid \varepsilon \quad B \to Bb \mid b$
- 非二义且非LR(1)的文法：$S \to aSb \mid B \quad B \to Bb \mid b$



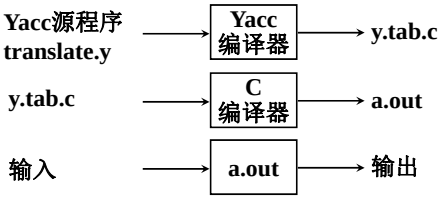
3.5 LR分析器

- 例 为语言$L = \{ a^m b^n \mid n > m \geq 0 \}$写三个文法，它们分别是LR(1)的、二义的和非二义且非LR(1)的
- LR(1)文法：$S \to AB \quad A \to aAb \mid \varepsilon \quad B \to Bb \mid b$
- 非二义且非LR(1)的文法：$S \to aSb \mid B \quad B \to Bb \mid b$
- 二义的文法：$S \to aSb \mid Sb \mid b$



3.6 语法分析器的生成器

3.6.1 分析器的生成器Yacc



3.6 语法分析器的生成器

3.6.2 用Yacc处理二义文法

- 例 台式计算器
  - 输入一个表达式并回车，显示计算结果
  - 也可以输入一个空白行

3.6 语法分析器的生成器

```
%{
# include  <ctype .h>
# include  <stdio.h >
# define YYSTYPE double  /* 将栈定义为double类型
  */
%}

%token  NUMBER
%left  '+'  '–'
%left  '*'  '/ '
%right  UMINUS
%%
```

3.6 语法分析器的生成器

```
lines       : lines expr '\n'    {printf ( "%g \n", $2 ) }
            | lines '\n'
            | /* ε */
            ;
expr        : expr '+' expr      {$$ = $1 + $3; }
            | expr '–' expr      {$$ = $1 – $3; }
            | expr '*' expr      {$$ = $1 * $3; }
            | expr '/ ' expr     {$$ = $1 / $3; }
            | '(' expr ')'       {$$ = $2; }
            | '–' expr  %prec UMINUS    {$$ = –$2; }
            | NUMBER
            ;
%%
```

3.6 语法分析器的生成器

```
lines       : lines expr '\n'    {printf ( "%g \n", $2 ) }
            | lines '\n'
            | /* ε */
            ;
expr        : expr '+' expr      {$$ = $1 + $3; }
            | expr '–' expr      {$$ = $1 – $3; }
            | expr '*' expr      {$$ = $1 * $3; }
            | expr '/ ' expr     {$$ = $1 / $3; }
            | '(' expr ')'       {$$ = $2; }
            | '–' expr  %prec UMINUS    {$$ = –$2; }
            | NUMBER
            ;
%%
```

-5+10看成是-(5+10), 还是(-5)+10? 取后者

3.6 语法分析器的生成器

```
yylex ( ) {
        int c;
        while ( ( c = getchar ( ) ) == ' ' );
        if ( ( c == '.' ) | | (isdigit (c) ) ) {
                ungetc (c, stdin);
                scanf ( "% lf ",  &yylval);
                return  NUMBER;
        }
        return c;
}
```

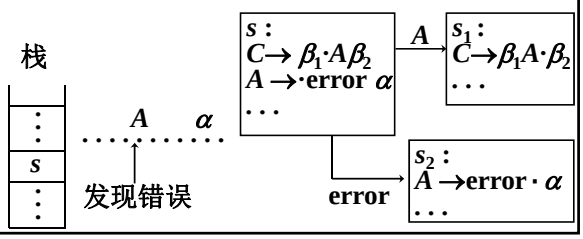
3.6 语法分析器的生成器

3.6.3 Yacc的错误恢复

- 编译器设计者的工作
  - 决定哪些“主要的”非终结符将有错误恢复与它们相关联
  - 为各主要非终结符A加入形式为$A \rightarrow \text{error } \alpha$的错误产生式，其中$\alpha$是文法符号串
  - 为这样的产生式配上语义动作
- Yacc把错误产生式当作普通产生式处理

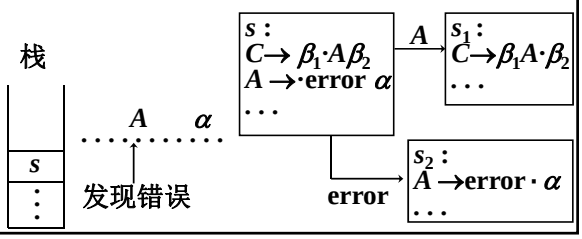
3.6 语法分析器的生成器

- 遇到语法错误时



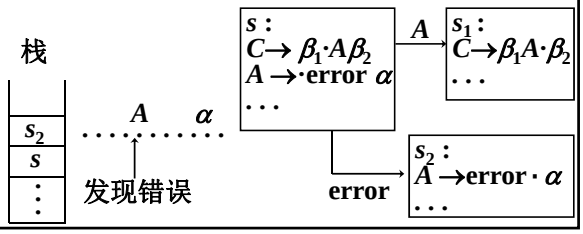
3.6 语法分析器的生成器

- 遇到语法错误时
  - 从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow \cdot \text{error } \alpha$的项目为止



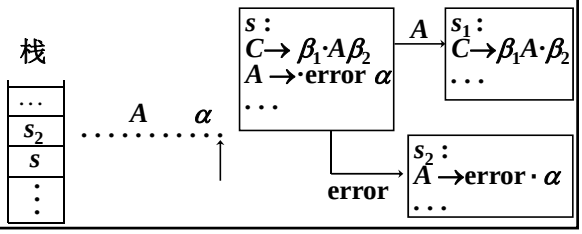
3.6 语法分析器的生成器

- 遇到语法错误时
  - 从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow \cdot error\ \alpha$的项目为止
  - 把虚构的终结符error“移进”栈



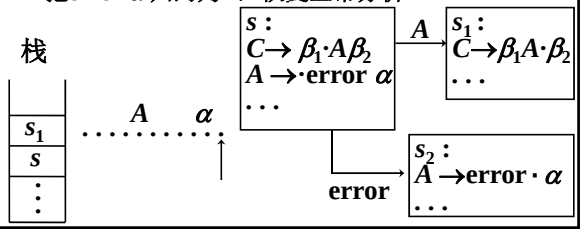
3.6 语法分析器的生成器

- 遇到语法错误时
  - 从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow \cdot error\ \alpha$的项目为止
  - 把虚构的终结符error“移进”栈
  - 抛弃若干输入符号，直至找到α，把α移进栈



3.6 语法分析器的生成器

- 遇到语法错误时
  - 从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow \cdot error\ \alpha$的项目为止
  - 把虚构的终结符error“移进”栈
  - 抛弃若干输入符号，直至找到α，把α移进栈
  - 把error α归约为A，恢复正常分析



3.6 语法分析器的生成器

- 增加错误恢复的台式计算器

```
lines  : lines expr '\n'    {printf ( "%g \n", $2 ) }
       | lines '\n'
       | /* ε */
       | error '\n'   {yyerror ("重新输入上一行");
                       yyerrok;}
       ;
```

本 章 要 点

- 文法和语言的基本知识
- 自上而下的分析方法：
  预测分析，非递归的预测分析，LL(1)文法
- 自下而上的分析方法：
  SLR(1)方法
- 语法分析器的生成器Yacc的基本原理
- 语法分析的错误恢复方法

例 题 1

下面的二义文法描述命题演算公式的语法，为它写一个等价的非二义文法

S → S and S | S or S | not S | p | q | (S)

非二义文法的产生式如下：

E → E or T | T

T → T and F | F

F → not F | (E) | p | q

例 题 1

下面的二义文法描述命题演算公式的语法，为它写一个等价的非二义文法

S → S and S | S or S | not S | p | q | (S)

非二义文法的产生式如下：

E → E or T | T

T → T and F | F

F → not E | (E) | p | q ?

not p and q有两种不同的最左推导

例 题 1

下面的二义文法描述命题演算公式的语法，为它写一个等价的非二义文法

S → S and S | S or S | not S | p | q | (S)

非二义文法的产生式如下：

E → E or T | T **not p and q**

T → T and F | F **not p and q**

F → not E | (E) | p | q ?

not p and q有两种不同的最左推导

例 题 2

设计一个文法:字母表{*a, b*}上*a*和*b*的个数相等的所有串的集合

- 二义文法： $S \to a\ S\ b\ S \mid b\ S\ a\ S \mid \varepsilon$

aabbabab *aabbabab*

S S *S S*

例 题 2

设计一个文法:字母表{*a, b*}上*a*和*b*的个数相等的所有串的集合

- 二义文法： $S \to a\ S\ b\ S \mid b\ S\ a\ S \mid \varepsilon$

aabbabab *aabbabab*

- 二义文法： $S \to a\ B \mid b\ A \mid \varepsilon$

$A \to a\ S \mid b\ A\ A$

$B \to b\ S \mid a\ B\ B$

aabbabab *aabbabab* *aabbabab*

B B *B B* *B B*

例 题 2

设计一个文法:字母表{*a, b*}上*a*和*b*的个数相等的所有串的集合

- 二义文法： $S \to a\ S\ b\ S \mid b\ S\ a\ S \mid \varepsilon$

aabbabab *aabbabab*

- 二义文法： $S \to a\ B \mid b\ A \mid \varepsilon$

$A \to a\ S \mid b\ A\ A$

$B \to b\ S \mid a\ B\ B$

aabbabab *aabbabab* *aabbabab*

- 非二义文法： $S \to a\ B\ S \mid b\ A\ S \mid \varepsilon$

$A \to a \mid b\ A\ A$

$B \to b \mid a\ B\ B$

a abb abab

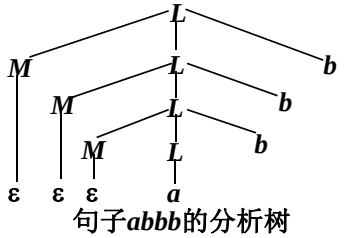
a B S

例 题 3

试说明下面文法不是LR(1)的：

$L \to M\ L\ b \mid a$

$M \to \varepsilon$



句子*abbb*的分析树

例　题　4

下面的文法不是LR(1)的，对它略做修改，使之成为一个等价的SLR(1)文法

program → **begin** ***declist*** **;** ***statement*** **end**
declist → **d ;** ***declist*** **| d**
statement → **s ;** ***statement*** **| s**

该文法产生的句子的形式是

begin d ; d ; … ; d ; s ; s ; … ; s end

修改后的文法如下：

program → **begin** ***declist statement*** **end**
declist → **d ;** ***declist*** **| d ;**
statement → **s ;** ***statement*** **| s**

习　　题

- 第一次：**3.2, 3.4(b)(c), 3.6(a)(b)**
- 第二次：**3.8, 3.13**
- 第三次：**3.14, 3.17**
- 第四次：**3.19, 3.22**