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

语法分析II

《编译原理和技术(H)》、《编译原理(H)》

张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

语法分析的主要方法

- 自上而下（top-down）
  - 从开始符号(根结点)出发，自上而下、从左到右地为输入串寻找最左推导
  - 即便消除左递归、提取左公因子，仍然存在一些程序语言,它们对应的文法不是LL(1)
- 自下而上（bottom-up）
  - 针对输入串，尝试根据产生式规则归约(reduce，将与产生式右部匹配的串归约为左部符号)，直至归约到开始符号
  - 比top-down方法更一般化

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

3.3 自上而下解析

- 自上而下解析、左递归及消除、提左因子
- LL(1)文法、LL(k)文法
- 预测解析器：递归下降的、非递归的（预测分析表）
- 错误恢复

自上而下解析的一般方法

自上而下（top-down）解析

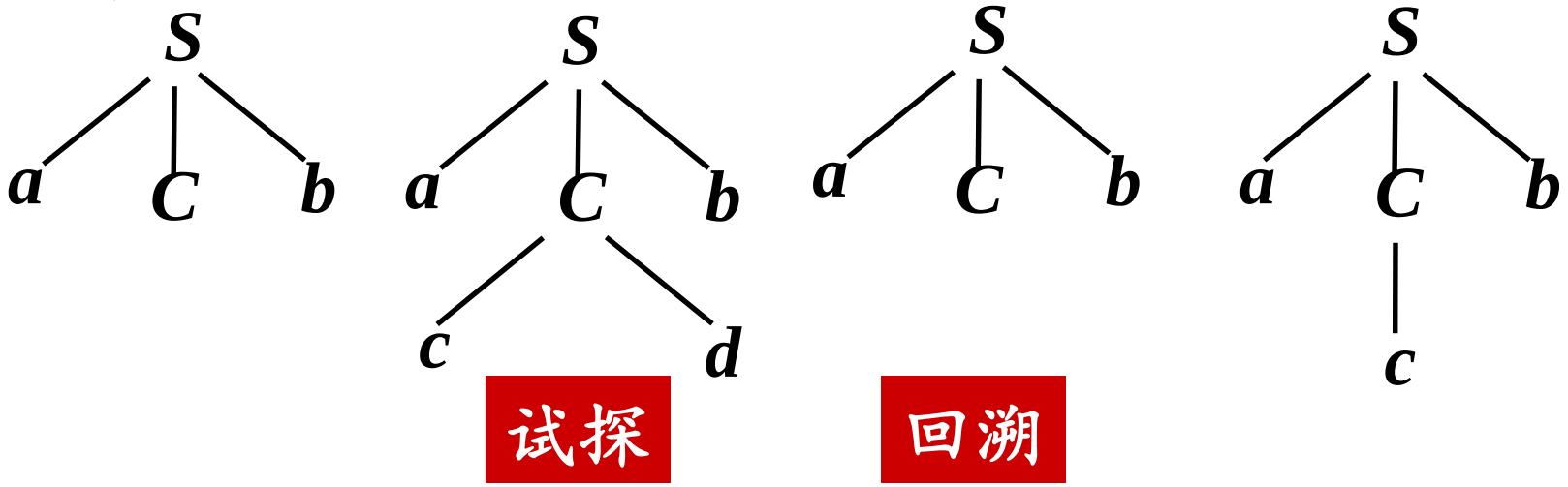
从开始符号出发，为输入串寻找最左推导

试探 产生式的选择 – 失败回溯(效率低，代价高)

- ANTLR: 引入带谓词的DFA使得回溯后不重新分析输入串

例　文法　　$S \to aCb$　　$C \to cd \mid c$

为输入串$w = acb$ 建立解析树（从根结点开始）



自上而下解析：左递归

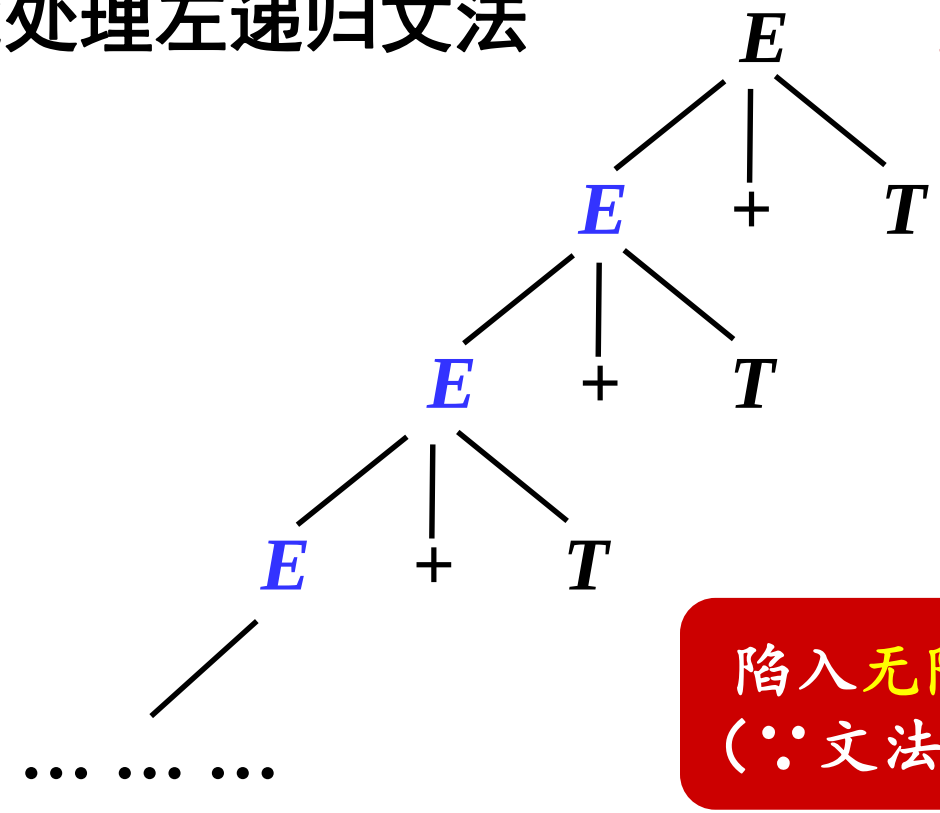
文法左递归

- 若文法中存在某非终结符A，使得$A \Rightarrow^{+} A\alpha$，其中$\alpha$是文法符号串

自上而下解析不能处理左递归文法

算术表达式文法

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



← 解析树

陷入无限循环
（∵文法左递归）

消除左递归(Eliminating left recursion)

文法左递归

- 若文法中存在某非终结符A，使得$A \Rightarrow^{+} A\alpha$，其中$\alpha$是文法符号串

直接左递归 (immediate left recursion)

若文法中存在 $A \rightarrow A\alpha \mid \beta$，其中文法符号串$\beta$不以$A$开头

- 由A推出的串的特点　　$A \Rightarrow^{+} \beta\alpha \ldots \alpha$

消除直接左递归

文法　$A \rightarrow A\alpha \mid \beta$，　可写成 $A \rightarrow \beta A'$

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

消除左递归

例 算术表达式文法

$$E \to E + T \mid T$$

$$T \to T * F \mid F$$

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

$$E \Rightarrow^{+} T + T \ldots + T$$

$$T \Rightarrow^{+} F * F \ldots * F$$

消除左递归后的文法

$$E \to TE'$$

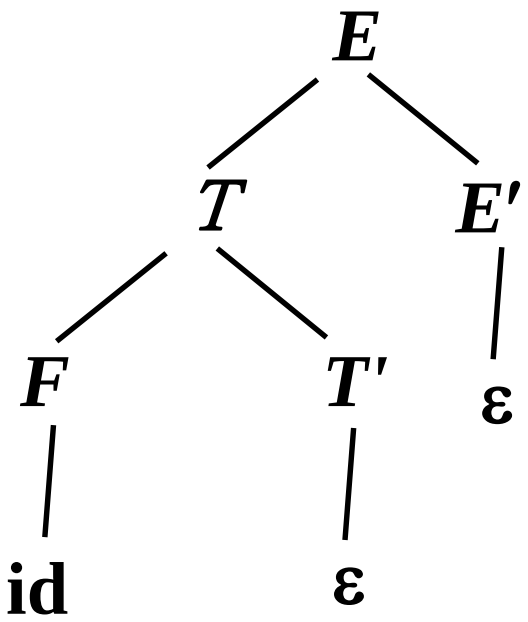
$$E' \to + TE' \mid \varepsilon$$

$$T \to FT'$$

$$T' \to * F T' \mid \varepsilon$$

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

自上而下识别id的解析树



消除非直接左递归

- 间接左递归

$$S \to Aa \mid b$$

$$A \to Sd \mid \varepsilon$$

- 先变换成直接左递归

$$S \to Aa \mid b$$

$$A \to Aad \mid bd \mid \varepsilon$$

- 再消除左递归

$$S \to Aa \mid b$$

$$A \to bdA' \mid A'$$

$$A' \to adA' \mid \varepsilon$$

- 隐藏左递归

$$S \to ABc$$

$$A \to \varepsilon$$

$$B \to ABe \mid Cd$$

$$C \to Sf$$

$B \Rightarrow ABe \Rightarrow Be$是隐藏直接左递归，而$S \Rightarrow ABc \Rightarrow Bc \Rightarrow Cdc \Rightarrow Sfdc$是隐藏间接左递归。

ANTLR4 接受的文法包含直接左递归，但不包含间接或隐藏的左递归

提左因子(left factoring)

- 有左因子的(left-factored)文法：$A \to \alpha\beta_1 \mid \alpha\beta_2$
  - 自上而下解析时，不清楚应该用A的哪个选择来代换
- 提左因子

$$A \to \alpha A' \qquad A' \to \beta_1 \mid \beta_2$$

例 悬空*else*的文法

$$stmt \to \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \mid \textbf{if } expr \textbf{ then } stmt \mid \textbf{other}$$

提左因子

$$stmt \to \textbf{if } expr \textbf{ then } stmt\ optional\_else\_part \mid \textbf{other}$$

$$optional\_else\_part \to \textbf{else } stmt \mid \varepsilon$$

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

3.3 自上而下解析

- 自上而下解析、左递归及消除、提左因子
- **LL(1)文法、LL(k)文法**
- 预测解析器：递归下降的、非递归的（预测分析表）
- 错误恢复

LL(1)文法

L-scanning from left to right; L- leftmost derivation

- 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
  - 符号串α的开始符号集合FIRST(α)

    $\text{FIRST}(\alpha) = \{a \mid \alpha \Rightarrow^* a\ldots, a \in V_T\} \cup \{\varepsilon \mid \alpha \Rightarrow^* \varepsilon\}$

    ↑ α 推出的非空串中首终结符和α 推出空串时的ε组成的集合

  - A的后继符号集合FOLLOW(A)

    $\text{FOLLOW}(A) = \{a \mid S \Rightarrow^* \ldots Aa\ldots,\ a \in V_T\} \cup \{\$ \mid S \Rightarrow^* \ldots A\}$

    ↑ A的后继符号集合：在句型中紧跟在A之后的终结符或$组成的集合

    这里约定$为输入记号串的结束标志，S是开始符号

LL(1)文法：FIRST(X)

- 计算FIRST(X), $X \in V_T \cup V_N$
  - $X \in V_T$, FIRST(X) = {X}
  - $X \in V_N$ 且 $X \to Y_1 Y_2 \ldots Y_k$

    如果 $a \in$ FIRST(Y_i)且ε 在FIRST(Y_1), …, FIRST(Y_{i-1})中， 则将 a加入到FIRST(X)

    如果 ε 在FIRST(Y_1), …, FIRST(Y_k)中， 则将ε 加入到FIRST(X)
  - $X \in V_N$ 且 $X \to \varepsilon$， 则将 ε 加入到 FIRST(X)

FIRST集合只可能包含终结符和ε

LL(1)文法：FIRST

- 计算FIRST($X_1 X_2 \dots X_n$)，$X_i \in V_T \cup V_N$，它包含
  - FIRST(X_1) 中所有的非 ε 符号
  - FIRST(X_i) 中所有的非 ε 符号，如果ε 在FIRST(X_1), …, FIRST(X_{i-1})中
  - ε，如果ε 在FIRST(X_1), …, FIRST(X_n)中

例

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

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

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }

FIRST(E') = {+, ε}

FRIST(T') = {*, ε}

LL(1)文法：FOLLOW

- 计算FOLLOW(A), $A \in V_N$
  - \$ 加入到FOLLOW(S) 中
  - 如果$A \to \alpha B \beta$, 则将 FIRST(β) 中所有终结符加入到 FOLLOW(B)
  - 如果$A \to \alpha B$ 或$A \to \alpha B \beta$ 且 $\varepsilon \in$ FIRST(β), 则FOLLOW(A)中的所有符号加入到FOLLOW(B)

$$\text{FOLLOW}(A) \subseteq V_T \cup \{\$\}$$

例

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to *FT' \mid \varepsilon$$
$$F \to (E) \mid \text{id}$$

FIRST(E) = FIRST(T) = FIRST(F) = { (, id }

FIRST(E') = {+, ε}

FRIST(T') = {*, ε}

FOLLOW(E) = FOLLOW(E') = {), \$}

FOLLOW(T) = FOLLOW(T') = {+,), \$}

FOLLOW(F) = {+, *,), \$}

LL(1)文法

LL(1)文法的定义

任何两个产生式 $A \to \alpha \mid \beta$ 都满足下列条件：

- $\mathbf{FIRST}(\alpha) \cap \mathbf{FIRST}(\beta) = \varnothing$
- 若 $\beta \Rightarrow^* \varepsilon$，那么 $\mathbf{FIRST}(\alpha) \cap \mathbf{FOLLOW}(A) = \varnothing$

例 对于下面文法，面临a...时，第2步推导不知用 A 的哪个产生式选择

$S \to A\ B$

$A \to a\ b \mid \varepsilon$ $\quad\quad\quad\quad\quad\quad$ $\because a \in \mathbf{FIRST}(ab) \cap \mathbf{FOLLOW}(A)$

$B \to a\ C$

$C \to \ldots$

LL(1)文法

LL(1)文法的定义

任何两个产生式 $A \to \alpha | \beta$ 都满足下列条件：

- $\mathbf{FIRST}(\alpha) \cap \mathbf{FIRST}(\beta) = \varnothing$
- 若 $\beta \Rightarrow^* \varepsilon$，那么 $\mathbf{FIRST}(\alpha) \cap \mathbf{FOLLOW}(A) = \varnothing$

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

LL(k)文法

LL(1)文法的定义可以扩展到LL(k)

任何两个产生式 $A \to \alpha | \beta$ 都满足下列条件：

- **FIRST(k,α) ∩ FIRST(k, β) = ∅**
- 若 $\beta \Rightarrow^* \varepsilon$，那么**FIRST(k, α) ∩ FOLLOW(k, A) = ∅**

FIRST (k, α)表示文法符号串α的前k个终结符串集合
FOLLOW(k, A)表示跟在A后的前k个终结符串集合

L- scanning from left to right 从左到右扫描输入

L- leftmost derivation 产生最左推导

k - 决定解析器的每步动作时需要向前查看k个输入符号

（即输入指针所指向的符号）

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

3.3 自上而下解析

- 自上而下解析、左递归及消除、提左因子
- LL(1)文法、LL(k)文法
- 预测解析器：递归下降、非递归（预测分析表）
- 错误恢复

预测解析器

预测解析

- 根据面临的输入符号，预测用非终结符的哪个选择进行展开(推导)

递归下降(recursive-descent)的预测解析

- 为每一个非终结符写一个解析函数
- 这些函数可能是递归的

递归：分析函数可能是递归的
下降：自上而下

例　下面的文法产生Pascal语言的类型子集

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

递归下降的预测解析器

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

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

递归下降的预测解析器

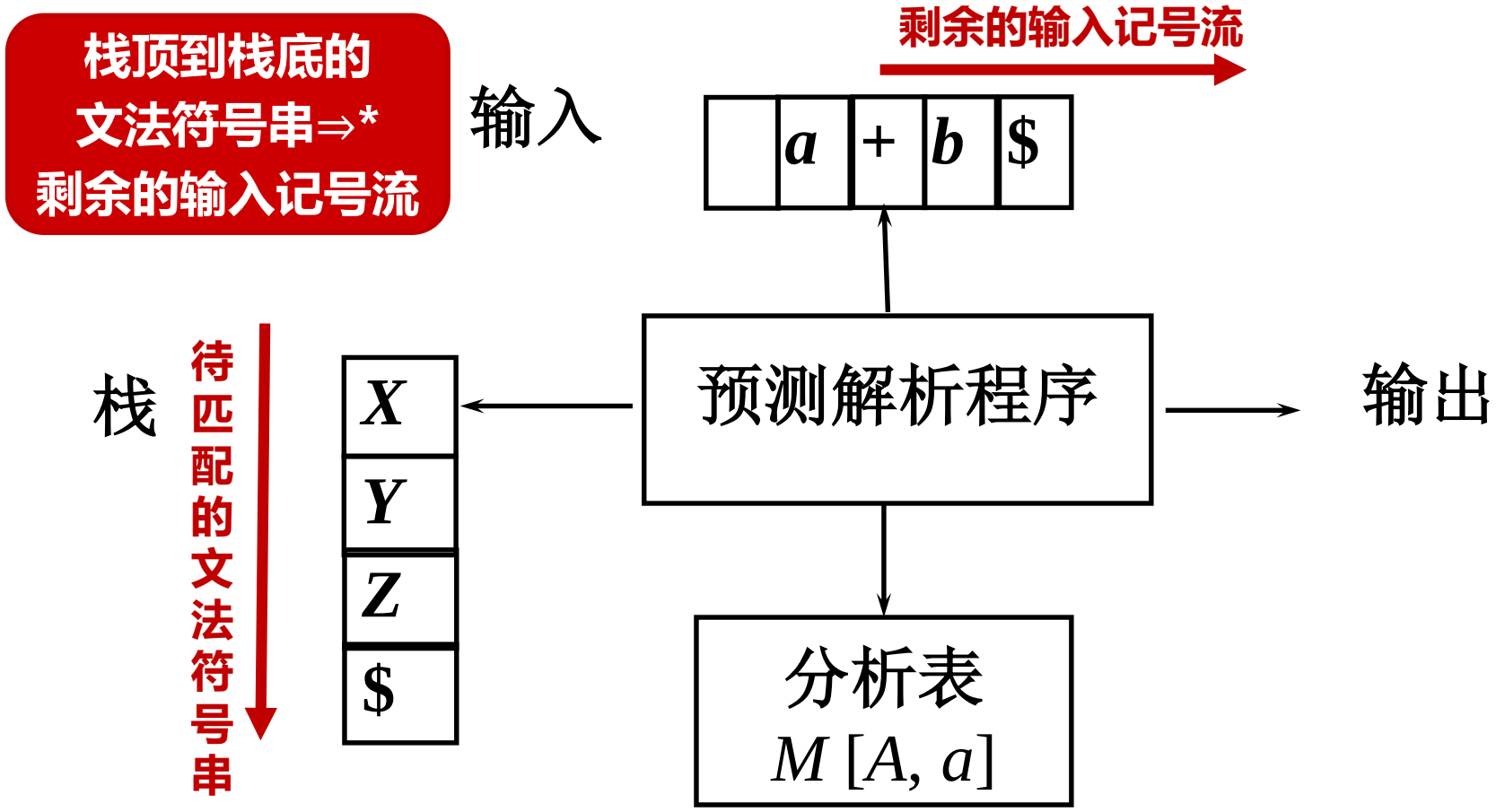
type → *simple* | ↑ id | array [*simple*] of *type*
simple → integer | char | num dotdot num

```
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( );
}
```

递归下降的预测解析程序的构造 与 文法结构直接对应
➔编程简单

非递归的预测解析

Nonrecursive Predictive Parsing



非递归的预测解析算法

让输入指针*ip*指向剩余输入记号流w$的第一个符号*a*;

令*X*等于栈顶符号;

while (*X* != $) { /* 栈非空 */

if (*X*是*a*) 把*X*从栈顶弹出并把*ip*推进到指向下一个符号;

else if (X是终结符) error ( );

else if (M[*X*, *a*]是出错入口) error ( );

else if (M[*X*, *a*] = $X \rightarrow Y_1 Y_2 \ldots Y_k$) {

输出产生式$X \rightarrow Y_1 Y_2 \ldots Y_k$;

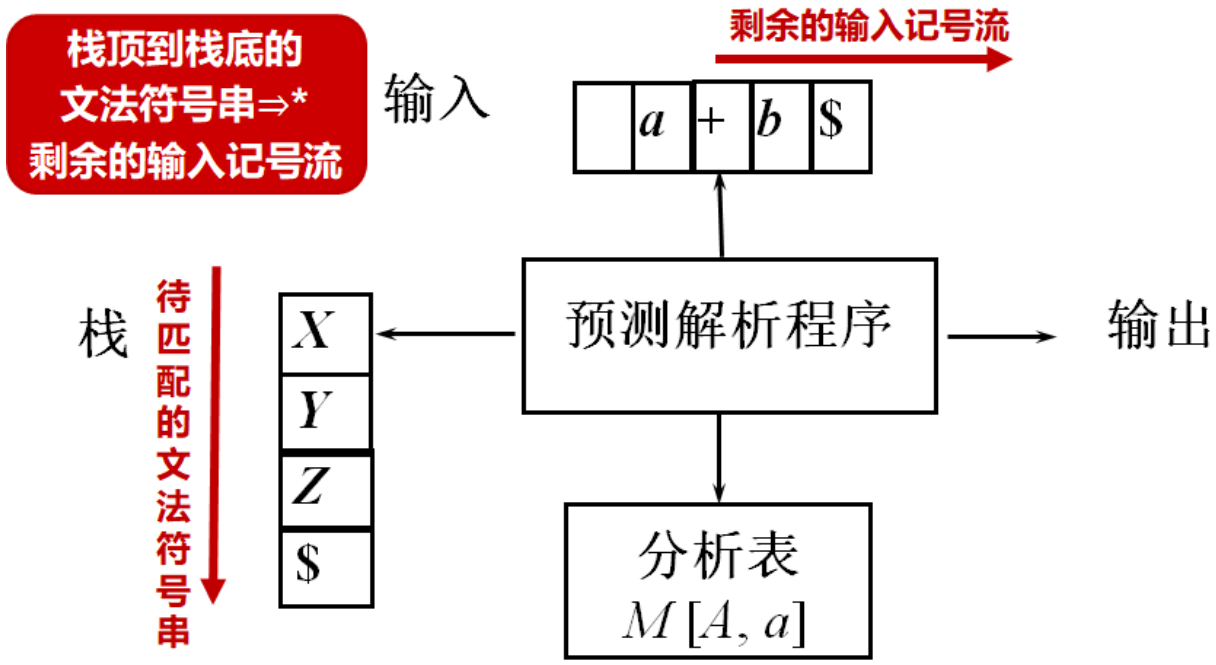
从栈中弹出*X*;

把$Y_k, Y_{k-1}, \ldots, Y_1$依次压入栈，$Y_1$在栈顶;

}

令*X*等于栈顶符号;

}



预测分析表

- 行：非终结符；列：终结符或$；单元：产生式
- 教材 表3.1

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

栈	输　入	输　出
$*E*	id * id + id$	

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测解析举例

预测解析器接受输入id * id + id的前一部分动作

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

预测分析表的构造

预测分析表(predictive parsing table)

行：非终结符；列：终结符 或$；单元：产生式

$M[A, a]$产生式$A \to \alpha$表示在面临a时，将栈顶符号A替换为α

构造方法

（1）对文法的每个产生式$A \to \alpha$，执行(2)和(3)

（2）对FIRST(α)的每个终结符a，把$A \to \alpha$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 \to \alpha$加入$M[A, b]$

（4）M中其它没有定义的条目都是error

多重定义

例 *stmt* → **if** *expr* **then** *stmt e_part* | *other*

e_part → **else** *stmt* | ε　　*expr* → *b*

非终结符	输入符号					
	other	*b*	**else**	**if**	**then**	$
stmt	*stmt* → **other**			*stmt* → **if…**		
e_part			*e_part* → **else** *stmt* *e_part* → ε			*e_part* → ε
expr		*expr* → *b*				

多重定义条目意味着文法左递归或者是二义的

多重定义的消除

例 删去*e_part* → ε，这正好满足 else和最近的then配对

LL(1)文法 ⇔ 预测分析表无多重定义的条目

非终结符	输入符号					
	other	*b*	**else**	**if**	**then**	$
stmt	*stmt* → **other**			*stmt* → **if…**		
e_part			*e_part* → **else** *stmt* ~~*e_part* → ε~~			*e_part* → ε
expr		*expr* → *b*				

没有通用的方法来指导多重定义条目的删除

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

3.3 自上而下解析

- 自上而下解析、左递归及消除、提左因子
- LL(1)文法、LL(k)文法
- 预测解析器：递归下降的、非递归的（预测分析表）
- 错误恢复

预测解析的错误恢复

编译器的错误处理

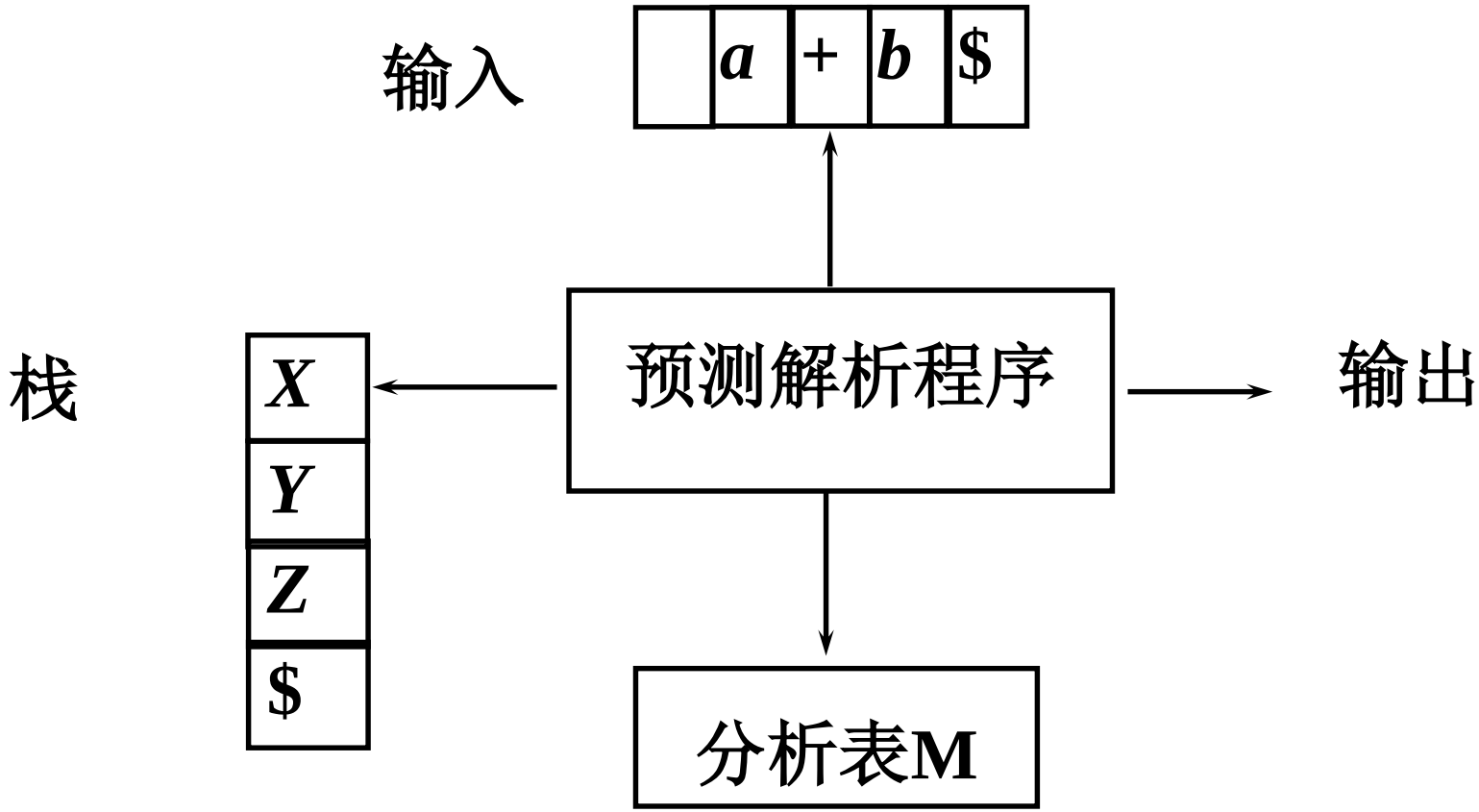
- 词法错误，如标识符、关键字或算符的拼写错
- 语法错误，如算术表达式的括号不配对
- 语义错误，如算符作用于不相容的运算对象
- 逻辑错误，如无穷的递归调用

解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
- 它不应该使正确程序的处理速度降低太多

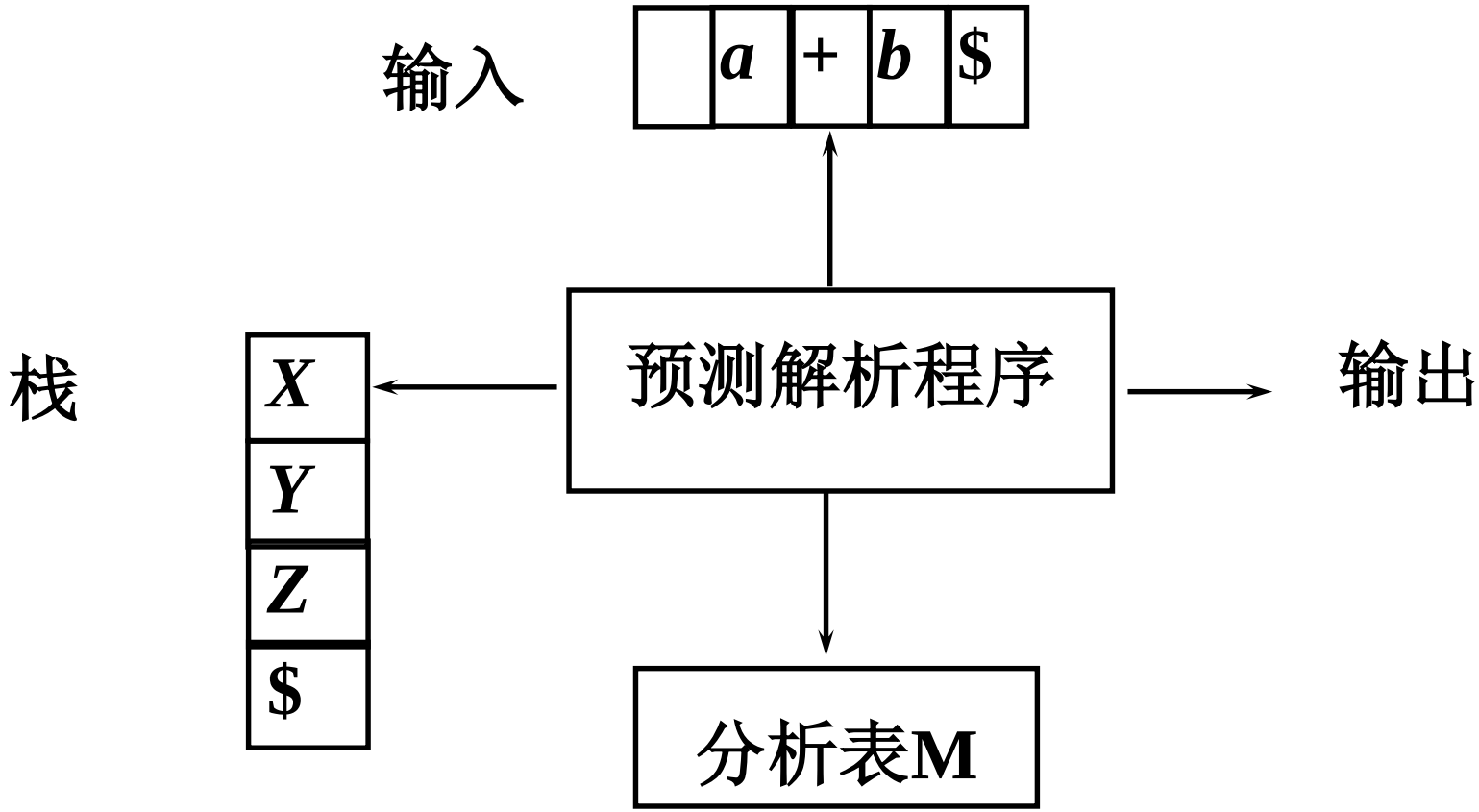
预测解析的错误恢复

- 非递归预测解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



预测解析的错误恢复

- 非递归预测解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



预测解析的错误恢复

非递归预测解析

采用紧急方式(panic mode)的错误恢复

- 发现错误时，抛弃输入记号直到其属于某个指定的同步记号(synchronizing tokens)集合为止

同步(synchronizing)

- 同步：词法分析器当前提供的记号流能够构成的语法构造，正是语法分析器所期望的
- 不同步的例子

  语法分析器期望剩余的前缀构成过程调用语句，而实际剩余的前缀形成的是赋值语句

预测解析的错误恢复

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*　　同步记号

  出错　（then和分号等记号是*expr*的同步记号）

- 把高层构造的首终结符加到低层构造的同步记号集中

  a = *expr*; if …

  出错　同步记号

  (语句的首终结符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)

预测解析的错误恢复

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
- 把高层构造的首终结符加到低层构造的同步记号集中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合

a = *expr*; , **if** …

出错　　同步记号

(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)

预测解析的错误恢复

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
- 把高层构造的首终结符加到低层构造的同步记号集中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合
- 如果出错时栈顶是存在有产生空串选择的非终结符，则可以使用其推出空串的产生式选择

错误恢复举例

例 栈顶为T'，面临id时出错

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

错误恢复举例

例 栈顶为T'，面临id时出错

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

预测解析的错误恢复

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
- 把高层构造的首终结符加到低层构造的同步记号集中
- 把FIRST(*A*)的终结符加入*A*的同步记号集合
- 如果出错时栈顶是存在有产生空串选择的非终结符，则可以使用其推出空串的产生式选择
- 如果终结符在栈顶而不能匹配，弹出此终结符

中国科学技术大学

University of Science and Technology of China

下期预告：ANTLR、自下而上的分析