



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



《编译原理与技术》

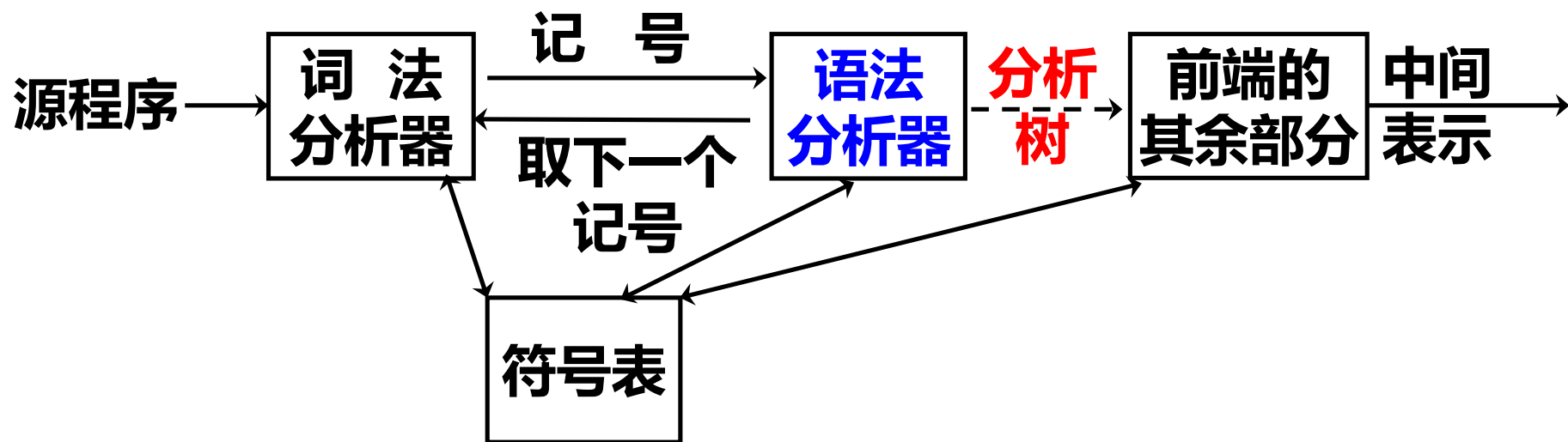
语法分析 II

计算机科学与技术学院

李 诚

18/09/2019

□第二节课以后，助教王一多和许冠斌同学将布置第一次实验，并作说明。



□ 语法分析方法概述

- ❖ 自顶向下与自底向上方法的区别
- ❖ 自顶向下分析方法
 - 递归下降预测分析方法
 - 消除左递归、提取左公因子
 - LL(1)文法及非递归预测分析方法



□ 自顶向下 (Top-down)

❖ 针对输入串，从文法的开始符号出发，尝试根据产生式规则**推导 (derive)** 出该输入串。

□ 自底向上 (Bottom-up)

❖ 针对输入串，尝试根据产生式规则**归约 (reduce)** 到文法的开始符号。



□ 自顶向下 (Top-down)

❖ 针对输入串，从文法的开始符号出发，尝试根据产生式规则**推导 (derive)** 出该输入串。

❖ 分析树的构造方法

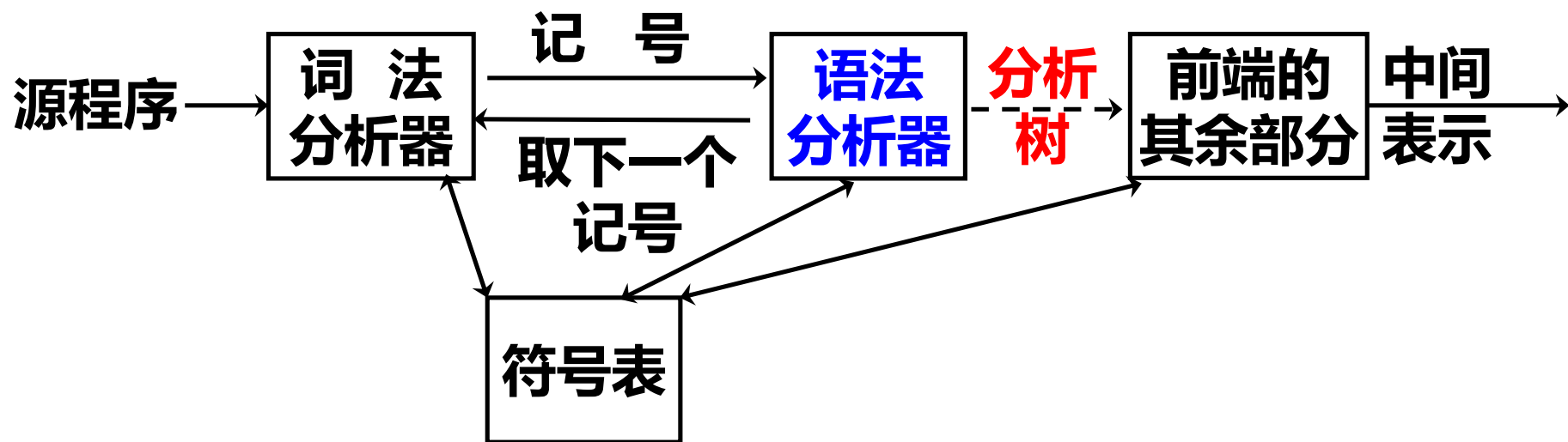
➤ 从根部开始

□ 自底向上 (Bottom-up)

❖ 针对输入串，尝试根据产生式规则**归约 (reduce)** 到文法的开始符号。

❖ 分析树的构造方法：

➤ 从叶子开始



□ 语法分析方法概述

❖ 自顶向下与自底向上方法的区别

❖ 自顶向下分析方法

➢ 递归下降分析方法

➢ 消除左递归、提取左公因子

➢ LL(1)文法及非递归预测分析方法



□ Recursive Descent Parsing

□ 考虑以下文法

$$E \rightarrow T / T + E$$

$$T \rightarrow \text{int} / \text{int} * T / (E)$$

□ 输入串: (int₅)

□ 分析步骤

❖ 从左到右扫描输入串

❖ 从开始非终结符 **E** 开始, 按顺序尝试 **E** 的产生式



递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$

E

(int_5)





递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$

E
|
 T

(int₅)
↑

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$

E
|
 T
|
 int

(int_5)
↑

不匹配
请回溯



递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$

E
|
 T

(int₅)
↑



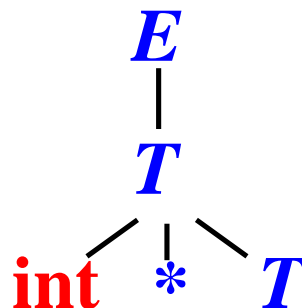
递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$



不匹配
请回溯

(int₅)





递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$

E
|
 T

(int₅)
↑



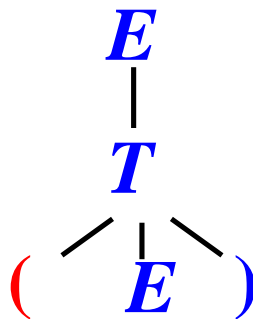
递归下降分析法



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

$$E \rightarrow T / T + E$$

$$T \rightarrow \text{int} / \text{int} * T / (E)$$



匹配
箭头前进

(int₅)





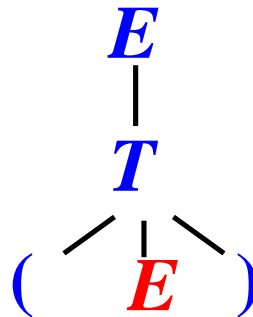
递归下降分析法



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

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$



(int₅)
↑



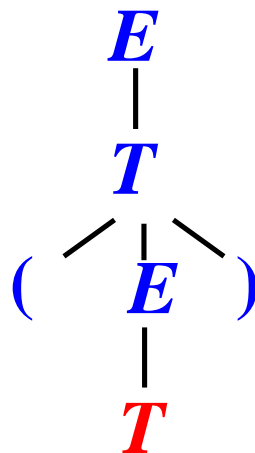
递归下降分析法



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

$E \rightarrow T / T + E$

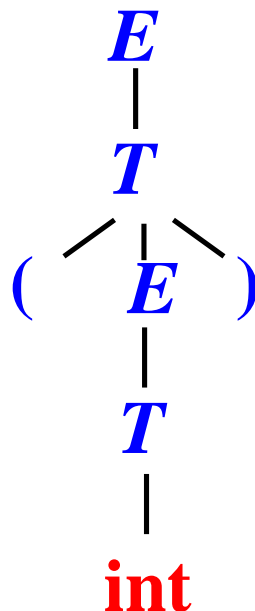
$T \rightarrow \text{int} / \text{int} * T / (E)$



(int₅)
↑

$E \rightarrow T / T + E$

$T \rightarrow \text{int} / \text{int} * T / (E)$



(int₅)
↑

匹配
箭头前进



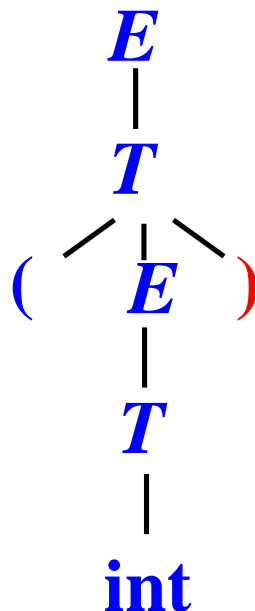
递归下降分析法



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

$$E \rightarrow T / T + E$$

$$T \rightarrow \text{int} / \text{int} * T / (E)$$



(int₅)
↑

匹配
箭头前进



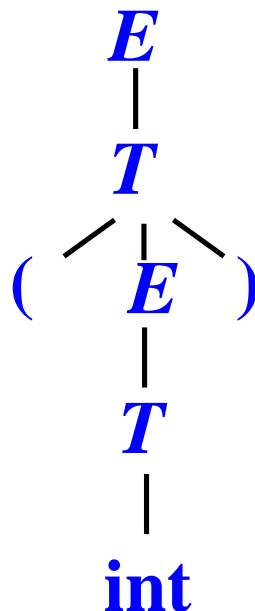
递归下降分析法



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

$$E \rightarrow T / T + E$$

$$T \rightarrow \text{int} / \text{int} * T / (E)$$



(int₅)



分析完毕
接受该串



□ 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：考虑下列文法

$type \rightarrow simple$
 | $\uparrow id$
 | $array [simple] \text{ of } type$
 $simple \rightarrow integer$
 | $char$
 | $num \text{ dotdot } num$



一个辅助过程

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



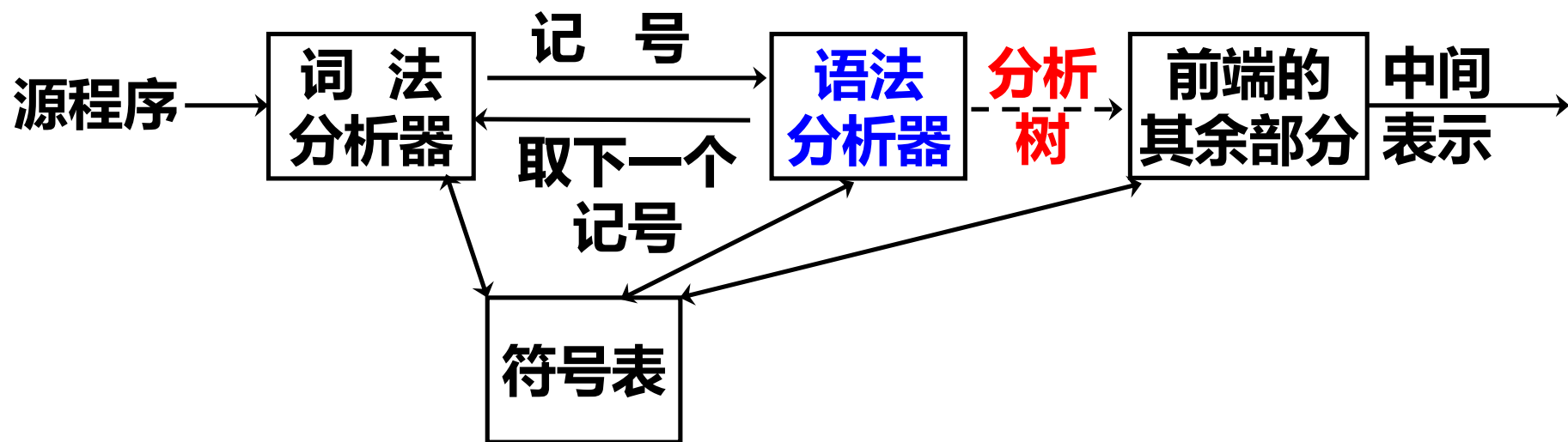
```
void simple( ) {  
    if ( lookahead == integer) match(integer);  
    else if (lookahead == char) match(char);  
    else if (lookahead == num) {  
        match(num); match(dotdot); match(num);  
    }  
    else error( );  
}
```

simple → integer
 | char
 | num dotdot num



```
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*



□ 语法分析方法概述

❖ 自顶向下与自底向上方法的区别

❖ 自顶向下分析方法

➤ 递归下降分析方法

➤ 消除左递归、提取左公因子

➤ LL(1)文法及非递归预测分析方法



□可能进入无限循环

□考虑以下文法

$$S \rightarrow Sa$$

□该文法是左递归的(left-recursive)



□可能进入无限循环

□考虑以下文法

$$S \rightarrow Sa$$

□该文法是左递归的(left-recursive)

□自顶向下分析方法无法处理左递归

❖ Why?



□文法左递归

$$A \Rightarrow^+ A \alpha$$

□直接左递归

$A \rightarrow A \alpha \mid \beta$, 其中 α, β 不以 A 开头

❖ 串的特点 $\beta \alpha \dots \alpha$

□消除直接左递归

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \epsilon$$



□例 算术表达文法

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

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

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



□例 算术表达文法

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \text{id}$$
$$(T + T \dots + T)$$
$$(F * F \dots * F)$$



□例 算术表达文法

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

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

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

$$(T + T \dots + T)$$

$$(F * F \dots * F)$$

□消除左递归后文法

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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



□非直接左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Sd \mid \varepsilon$$



□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

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bd A' \mid A'$$

$$A' \rightarrow adA' \mid \varepsilon$$



□ 复杂的回溯 → 代价太高

- ❖ 非终结符有可能有多个产生式
- ❖ 由于信息缺失，无法准确预测选择哪一个

□ 有左因子的(left -factored)文法:

$$\text{❖ } A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$



□ 推后选择产生式的时机，以便获取更多信息

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

$A \rightarrow \alpha A'$

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



□例 悬空 $else$ 的文法

$$\begin{aligned} stmt &\rightarrow \text{if } expr \text{ then } stmt \text{ else } stmt \\ &| \text{if } expr \text{ then } stmt \\ &| \text{other} \end{aligned}$$

提左因子

$$\begin{aligned} stmt &\rightarrow \text{if } expr \text{ then } stmt \text{ optional_else_part} \\ &| \text{other} \\ \text{optional_else_part} &\rightarrow \text{else } stmt \\ &| \epsilon \end{aligned}$$



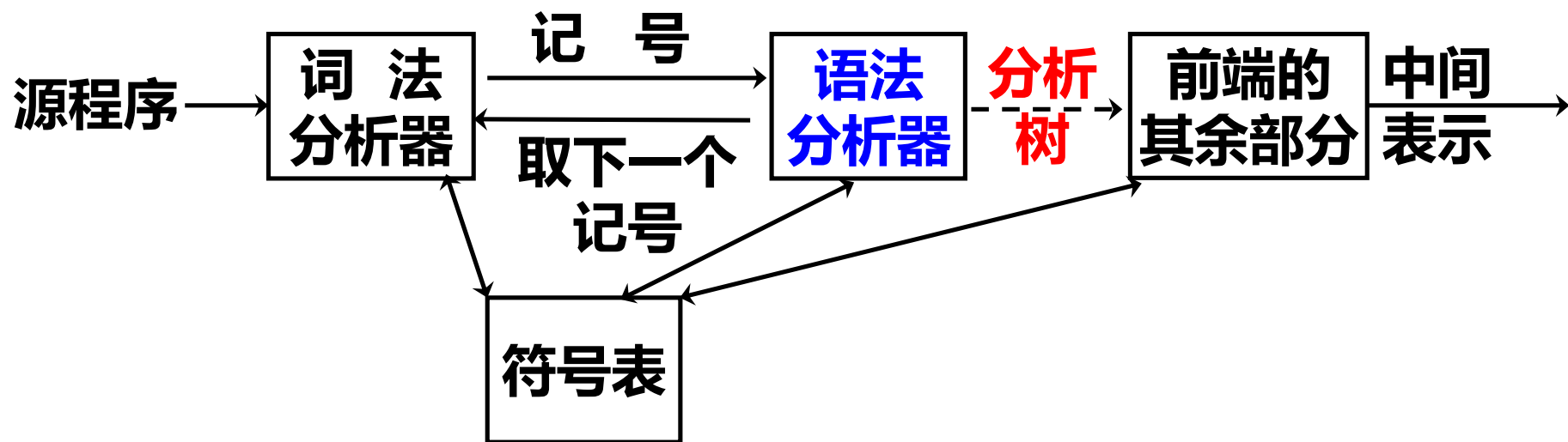
□ 简单、一般化的语法分析方法

- ❖ 但左递归必须先消除
- ❖ 可自动化地消除左递归

□ 因为回溯而不受欢迎

- ❖ 效率太低

□ 因此，在构造编译器时，可通过对文法加以限制来避免回溯



□ 语法分析方法概述

❖ 自顶向下与自底向上方法的区别

❖ 自顶向下分析方法

➢ 递归下降分析方法

➢ 消除左递归、提取左公因子

➢ LL(1)文法及非递归预测分析方法



□ Predictive parsing

□ 与递归下降法相似，但

- ❖ 不会对若干产生式进行尝试
- ❖ 没有回溯
- ❖ 通过向前看一些记号来预测需要用到的产生式

□ 此方法接受 $LL(k)$ 文法

- ❖ L -means “left-to-right” scan of input
- ❖ L -means “leftmost derivation”
- ❖ k -means “predict based on k tokens of lookahead”
- ❖ In practice, $LL(1)$ is used



□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

❖ $\text{FIRST}(\alpha) = \{a \mid \alpha \Rightarrow^* a..., a \in V_T\}$

特别是, $\alpha \Rightarrow^* \varepsilon$ 时, 规定 $\varepsilon \in \text{FIRST}(\alpha)$

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

如果非终结符 A 是某个句型的最右符号, 那么 $\$$ 属于 $\text{FOLLOW}(A)$

$\$$ 是输入串的结束符号



□ 计算FIRST(X), $X \in V_T \cup V_N$

❖ $X \in V_T$, $\text{FIRST}(X) = \{X\}$

❖ $X \in V_N$ 且 $X \rightarrow \epsilon$

则将 ϵ 加入到FIRST(X)

❖ $X \in V_N$ 且 $X \rightarrow Y_1 Y_2 \dots Y_k$

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

➤ 如果 ϵ 在 $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_k)$ 中, 则将 ϵ 加入到FIRST(X)



□ 计算FOLLOW(A), $A \in V_N$

❖ \$ 加入到FOLLOW(A), 当A是开始符号

❖ 如果 $A \rightarrow \alpha B \beta$, 则 $\text{FIRST}(\beta) - \{\epsilon\}$ 加入到 FOLLOW(B)

❖ 如果 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B \beta$ 且 $\epsilon \in \text{FIRST}(\beta)$, 则 FOLLOW(A) 加入到 FOLLOW(B)



□为什么需要First和Follow集合?



□ 为什么需要First和Follow集合?

□ 考虑以下文法:

❖ $S \rightarrow aAS / d$

❖ $A \rightarrow bAS / \varepsilon$

□ 同时考虑串**abdd**的推导过程



□ 为什么需要First和Follow集合?

□ 考虑以下文法:

❖ $S \rightarrow aAS / d$

❖ $A \rightarrow bAS / \varepsilon$

□ 同时考虑串**abdd**的推导过程

❖ $S \Rightarrow aAS \Rightarrow abASS$, 推导无法进行下去

❖ 输入串需要考虑的记号是d

❖ 按照FIRST集合, 应该选择 $S \rightarrow d$ 这个产生式

❖ 但是, 当前需要推导的非终结符是A

❖ 因此, 我们需要利用d属于FOLLOW(A)的事实,
通过 $A \rightarrow \varepsilon$ 将A消掉



□LL(1)文法的定义

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

❖ $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$



□LL(1)文法的定义

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

❖ **FIRST(α) \cap FIRST(β) = \emptyset**

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么**FIRST(α) \cap FOLLOW(A) = \emptyset**

□该条件存在的必要性

❖ 容易理解

❖ 每次通过输入词法单元记号和FIRST集合匹配产生式的时候, 需要有唯一的选择



□LL(1)文法的定义

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

❖ $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

□假设 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \{a\}$

$a \in \text{FIRST}(\alpha): A \Rightarrow^* a\alpha'$

$a \in \text{FOLLOW}(A): B \Rightarrow^* \dots A a \dots$



□LL(1)文法的定义

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

❖ $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

□假设 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \{a\}$

$a \in \text{FIRST}(\alpha): A \Rightarrow^* a\alpha'$

$a \in \text{FOLLOW}(A): B \Rightarrow^* \dots A a \dots$

由于 $\beta \Rightarrow^* \varepsilon$, 所以遇到 a 时, 无法判断用哪一个产生式

❖ 可以用 $A \rightarrow \alpha$ 来对 A 进行展开

❖ 亦可以用 $A \rightarrow \beta$ 和 $\beta \Rightarrow^* \varepsilon$ 最后把 A 消掉



□LL(1)文法的定义

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

❖ $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

□例如, 考虑下面文法

面临 $a\dots$ 时, 第2步推导不知用哪个产生式

$$S \rightarrow A B$$

$$A \rightarrow a b \mid \varepsilon \quad a \in \text{FIRST}(ab) \cap \text{FOLLOW}(A)$$

$$B \rightarrow a C$$

$$C \rightarrow \dots$$



□LL(1)文法的定义

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

❖ $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$

❖ 若 $\beta \Rightarrow^* \varepsilon$, 那么 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

□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



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

$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, \text{id} \}$

$\text{FIRST}(E') = \{ +, \varepsilon \}$

$\text{FIRST}(T') = \{ *, \varepsilon \}$

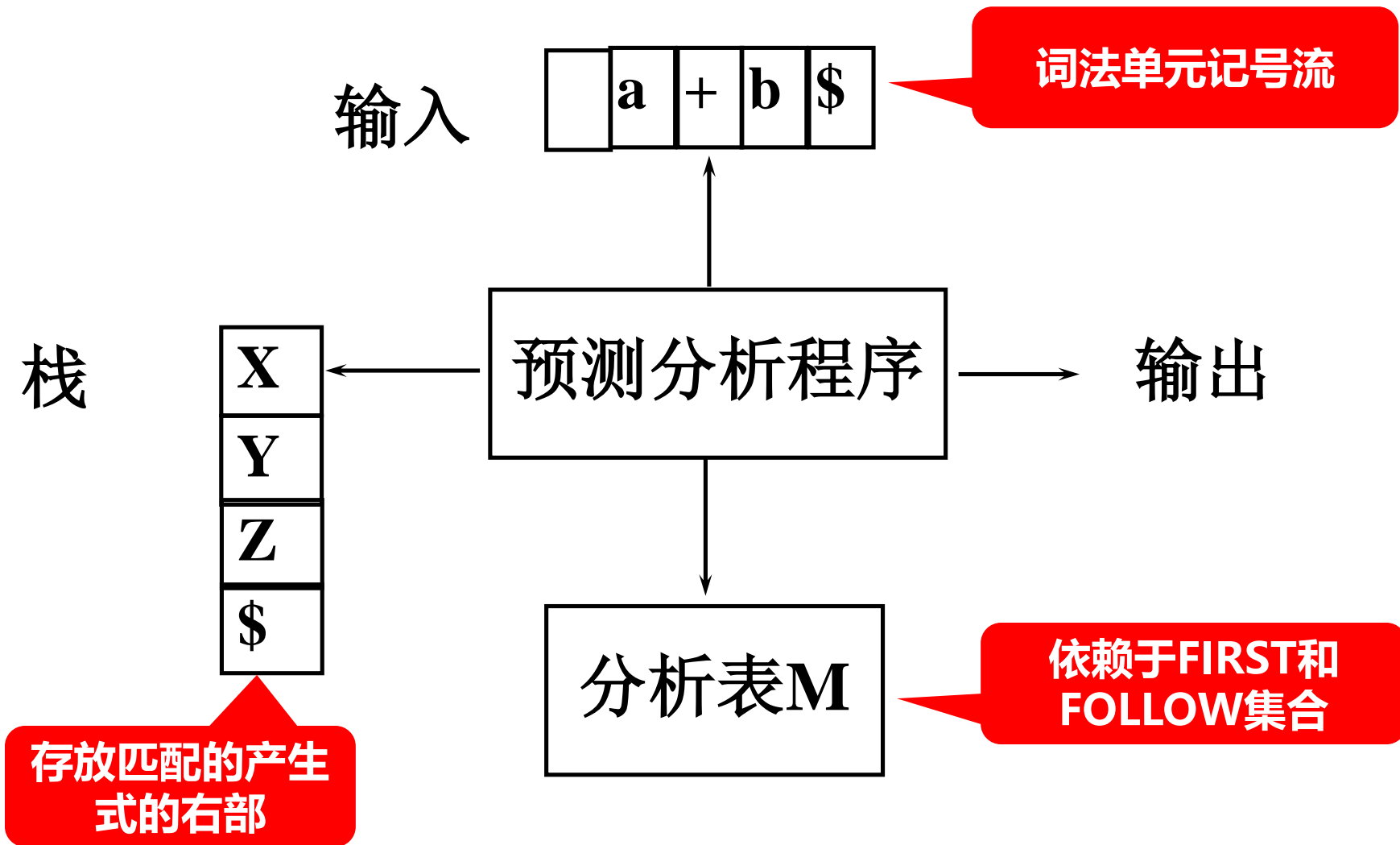
$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{), \$ \}$

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

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



非递归的预测分析





□对文法的每个产生式 $A \rightarrow \alpha$ ，执行(1)和(2)

- ❖ (1) 对 $\text{FIRST}(\alpha)$ 的每个终结符 a ，把 $A \rightarrow \alpha$ 加入 $M[A, a]$
- ❖ (2) 如果 ϵ 在 $\text{FIRST}(\alpha)$ 中，对 $\text{FOLLOW}(A)$ 的每个终结符 b （包括 $\$$ ），把 $A \rightarrow \alpha$ 加入 $M[A, b]$

M 中其它没有定义的条目都是error



预测分析表M的构造



□行：非终结符；列：终结符 或\$；单元：产生式

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

栈	输 入	输 出
$\$E$	$id * id + id\$$	



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



预测分析器接受输入 $id * id + id$ 的前一部分动作

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



例: $stmt \rightarrow \text{if } expr \text{ then } stmt \text{ } e_part \mid \text{other}$
 $e_part \rightarrow \text{else } stmt \mid \varepsilon$ $expr \rightarrow b$

非终结符	输入符号			
	other	b	else	...
$stmt$	$stmt \rightarrow \text{other}$			
e_part			$e_part \rightarrow$ $\text{else } stmt$ $e_part \rightarrow \varepsilon$	
$expr$		$expr \rightarrow b$		

多重定义条目意味着文法左递归或者是二义的



例：删去 $e_part \rightarrow \varepsilon$ ，这正好满足else和近的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	...
$stmt$	$stmt \rightarrow other$			
e_part			$e_part \rightarrow$ else $stmt$ $e_part \rightarrow \varepsilon$	
$expr$		$expr \rightarrow b$		



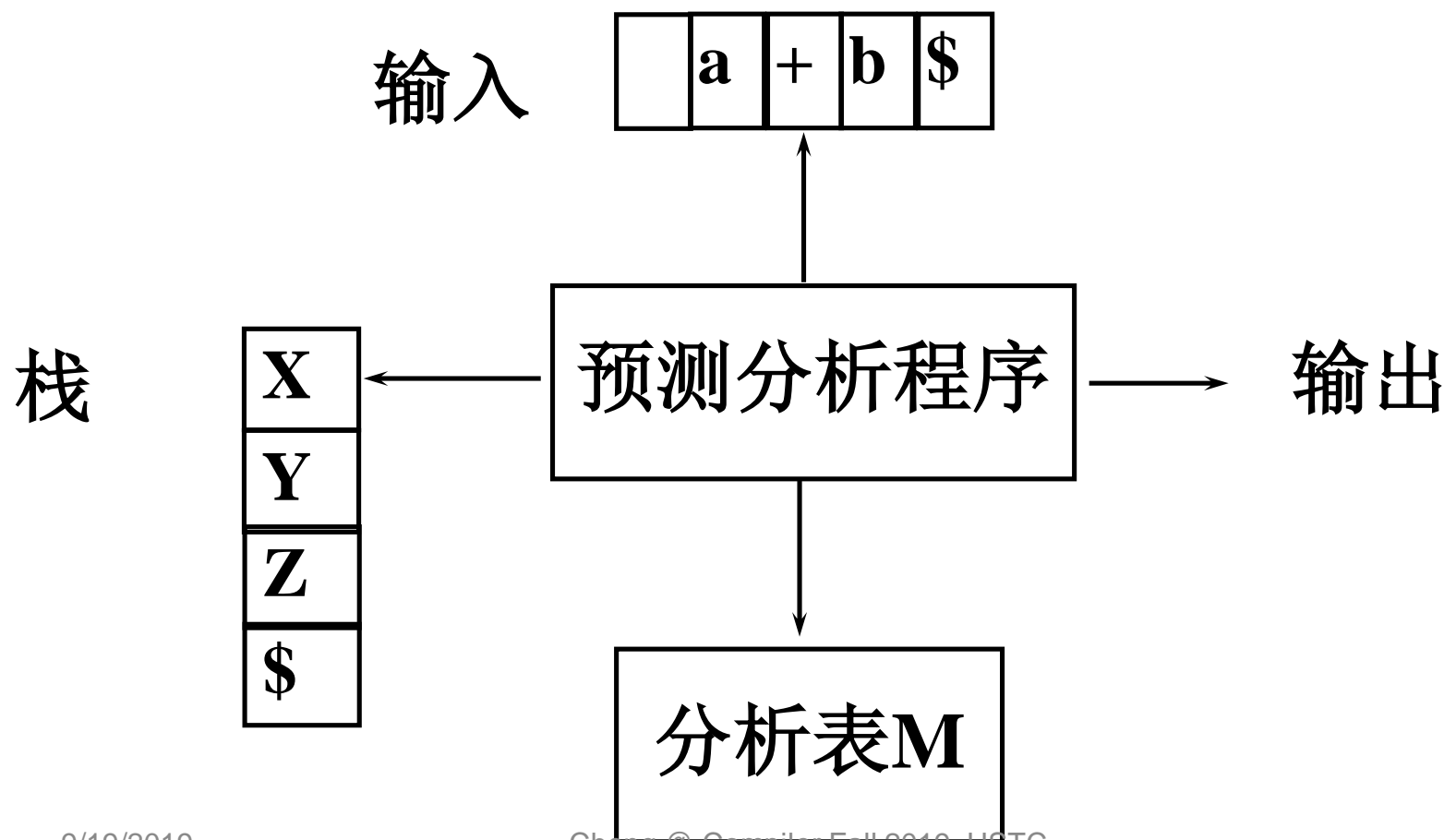
□编译器的错误

- ❖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



□非递归预测分析在什么场合下发现错误

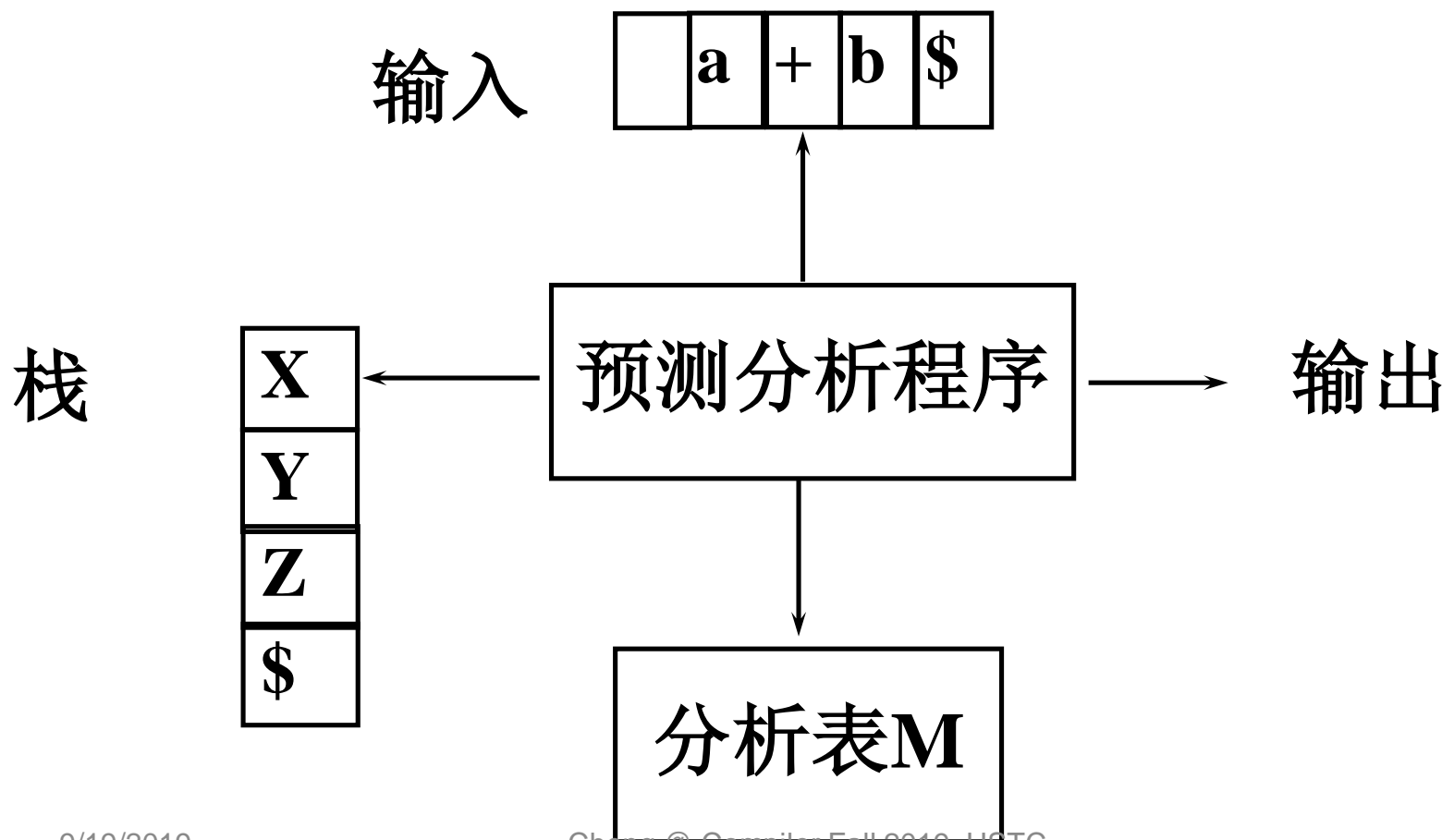
❖ 栈顶的终结符和下一个输入符号不匹配





□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





□**颁奖词：** For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

❖ https://amturing.acm.org/award_winners/knuth_1013846.cfm

□**演讲：**

❖ Computer Programming as an Art

❖ https://dl.acm.org/ft_gateway.cfm?id=1283929&type=pdf

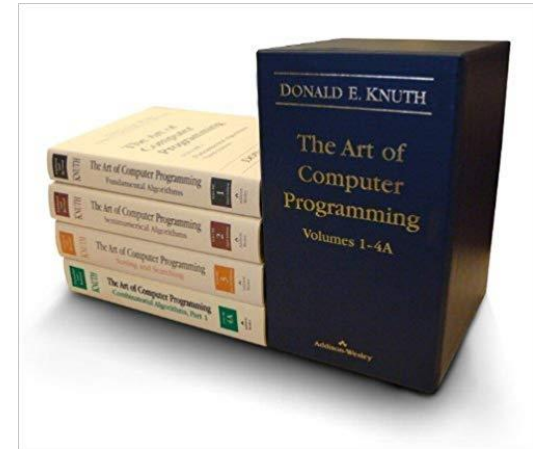


加州理工学院博士
斯坦福大学教授



□ 鸿篇巨著: *The Art of Computer Programming* (“TAOCP”)

- ❖ 始于1963年
- ❖ 规划7卷，分别涉及：算法基础、高级算法、排列组合、上下文无关语言理论基础、编译原理等
- ❖ 自1968年第一卷问世起，已出版4卷



□ *If you think you're a really good programmer... read [Knuth's] Art of Computer Programming... You should definitely send me a resume if you can read the whole thing.*

❖ —Bill Gates



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



《编译原理与技术》

语法分析 II

任何伟大的科学发现都是不断平凡的积累！
—— 钱学森