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

USTC, CHINA
ADSLAB
先进数据系统实验室

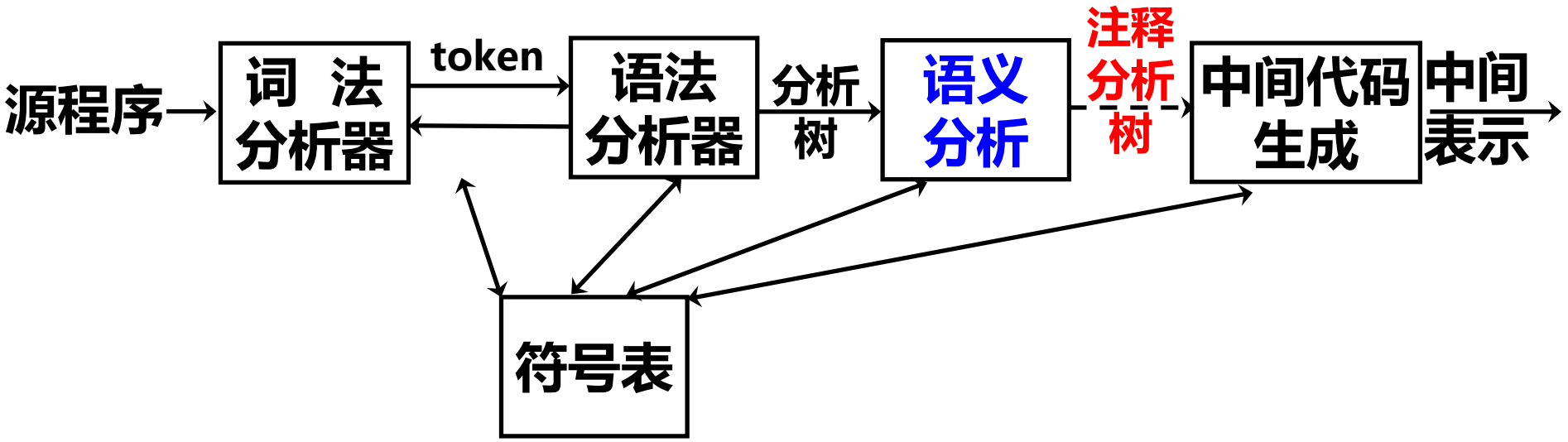
《编译原理与技术》
语法制导翻译 II

计算机科学与技术学院
李 诚
22/10/2018

Announcement

- **Tutorial on Thursday (25/10/2018)**
  - **3B201, Class time**
  - **Assignment review**
  - **Q & A**

主要内容

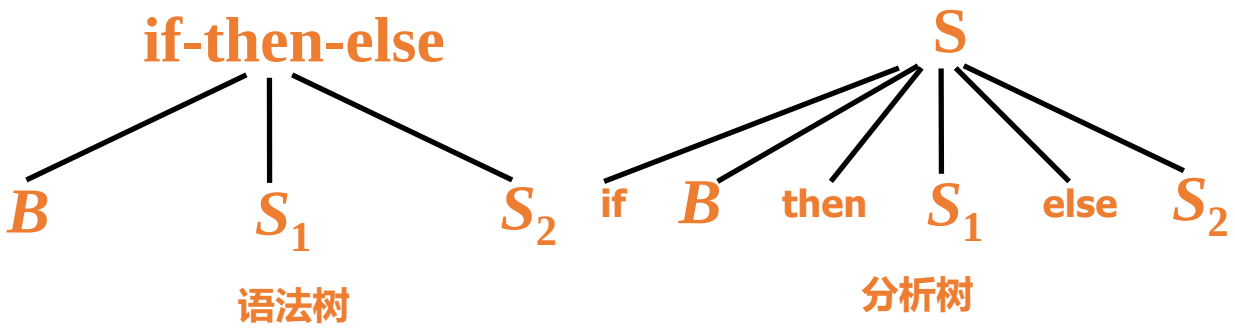


- 语法分析树→ 抽象语法树
- 从语法制导定义到翻译方案
  - S属性定义的SDT
  - L属性定义的SDT

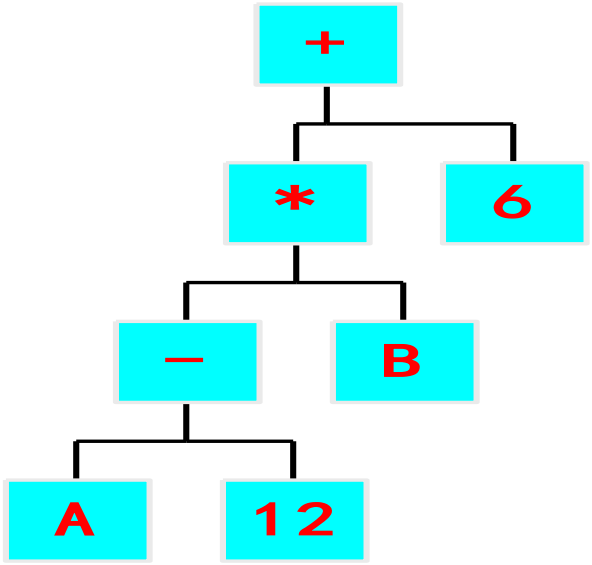
抽象语法树

- 语法树是分析树的浓缩表示：**算符**和**关键字**是作为内部结点。
- 语法制导翻译可以基于分析树，也可以基于语法树

S→ if B then S1 else S2



- 例：表达式 (A - 12) * B + 6 的语法结构树。



建立算符表达式的语法树

- mknode (op, left, right)
  - 建立一个运算符号结点，标号是op，两个域left和right分别指向左子树和右子树。
- mkleaf (id, entry)
  - 建立一个标识符结点，标号为id，一个域entry指向标识符在符号表中的入口。
- mkleaf (num, val)
  - 建立一个数结点，标号为num，一个域val用于存放数的值。

❑以算数表达式为例

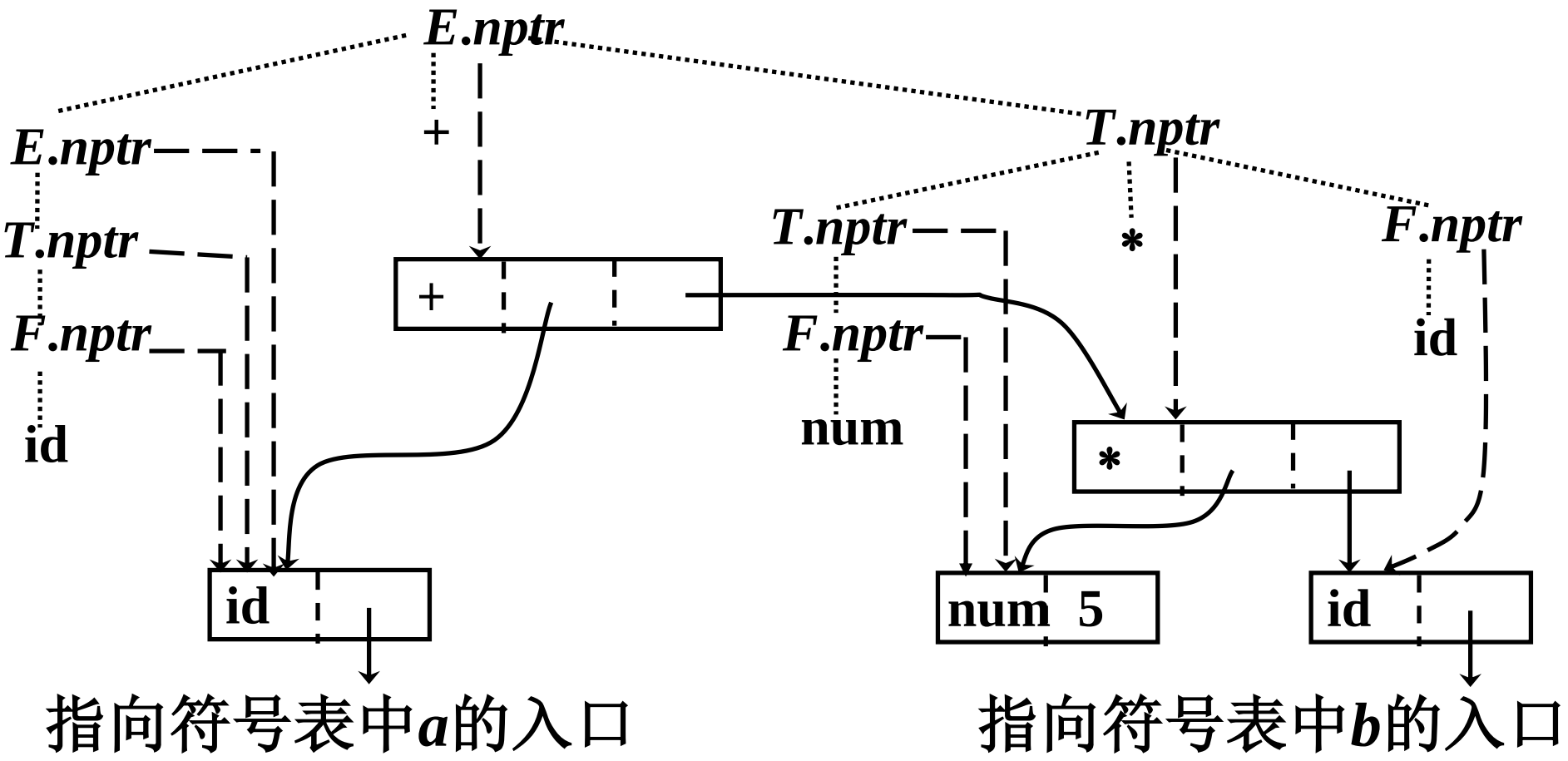
产　生　式	语 义 规 则
$E \rightarrow E_1 + T$	$E.nptr = mkNode(\text{‘+’}, E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode(\text{‘*’}, T_1.nptr, F.nptr)$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow \text{id}$	$F.nptr = mkLeaf(\text{id}, \text{id}.entry)$
$F \rightarrow \text{num}$	$F.nptr = mkLeaf(\text{num}, \text{num}.val)$

❑注意事项：

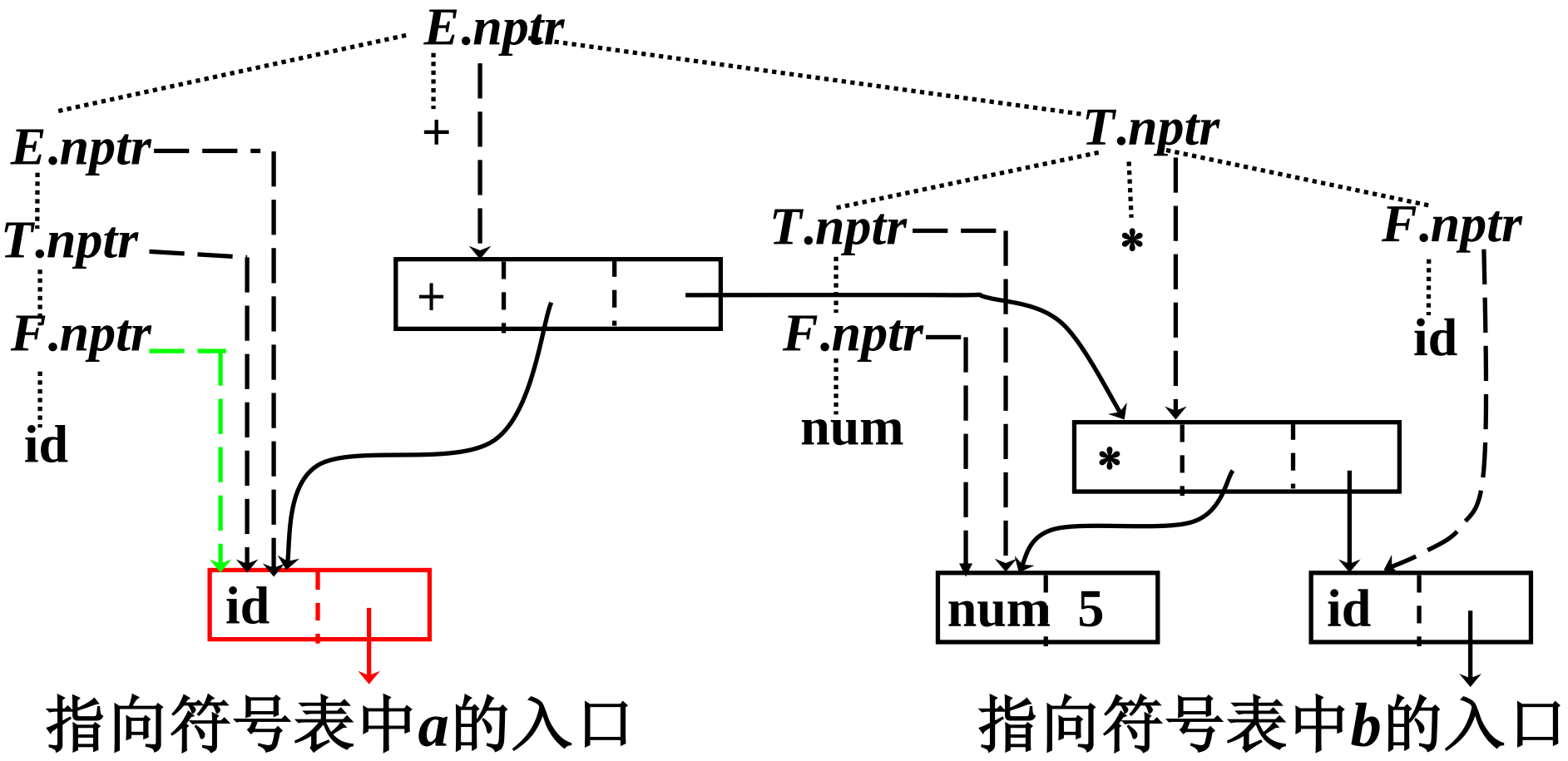
- ❖同样是产生式附带语义规则，不同的语义规则产生不同的作用。
- ❖对算符结点，一个域存放算符并作为该结点的标记，其余两个域存放指向运算对象的指针。
- ❖基本运算对象结点，一个域存放运算对象类别，另一个域存放其值。（也可用其他域保存其他属性或者指向该属性值的指针）

S属性定义的语法树构造

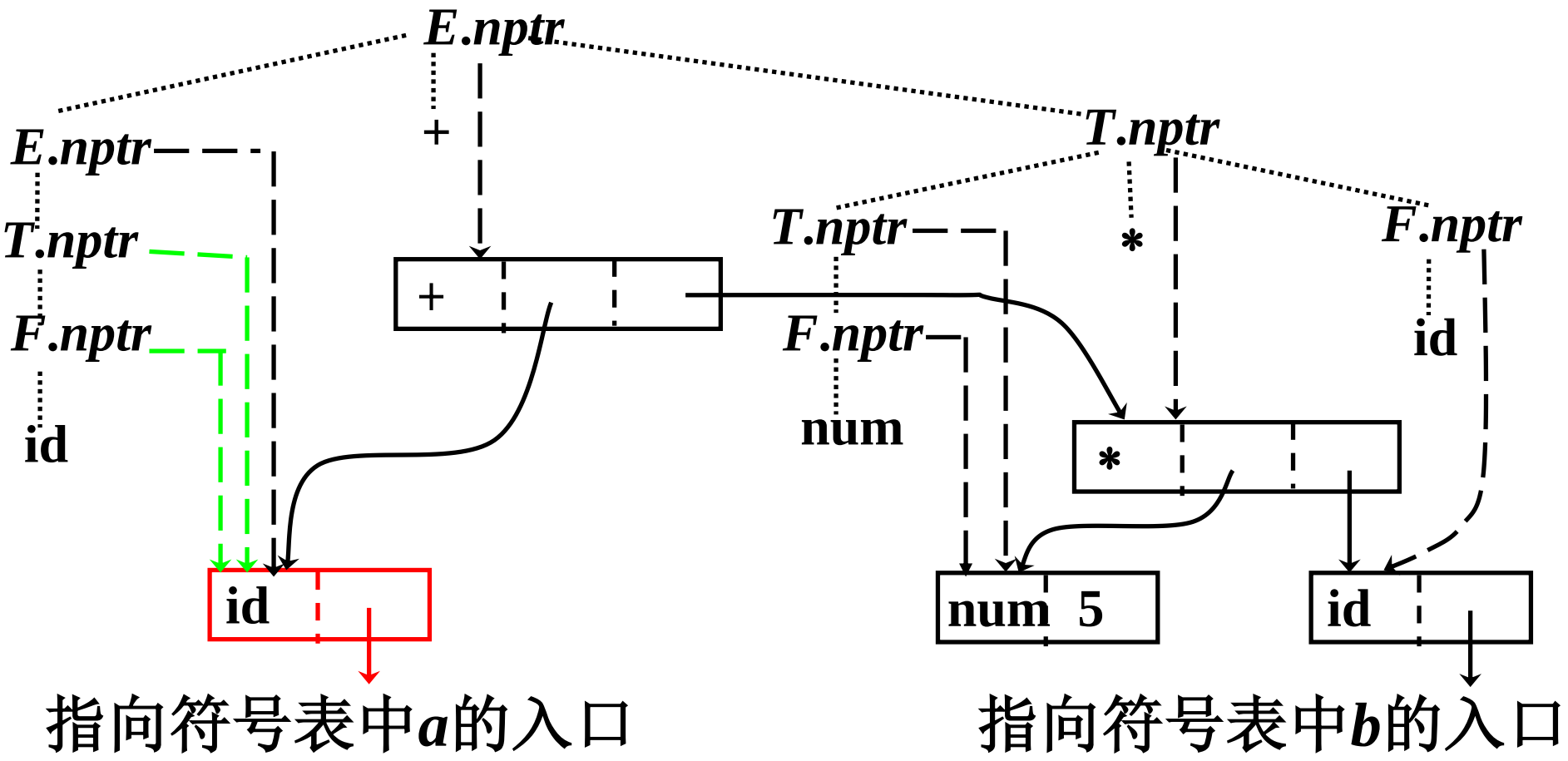
a+5**b*的语法树的构造



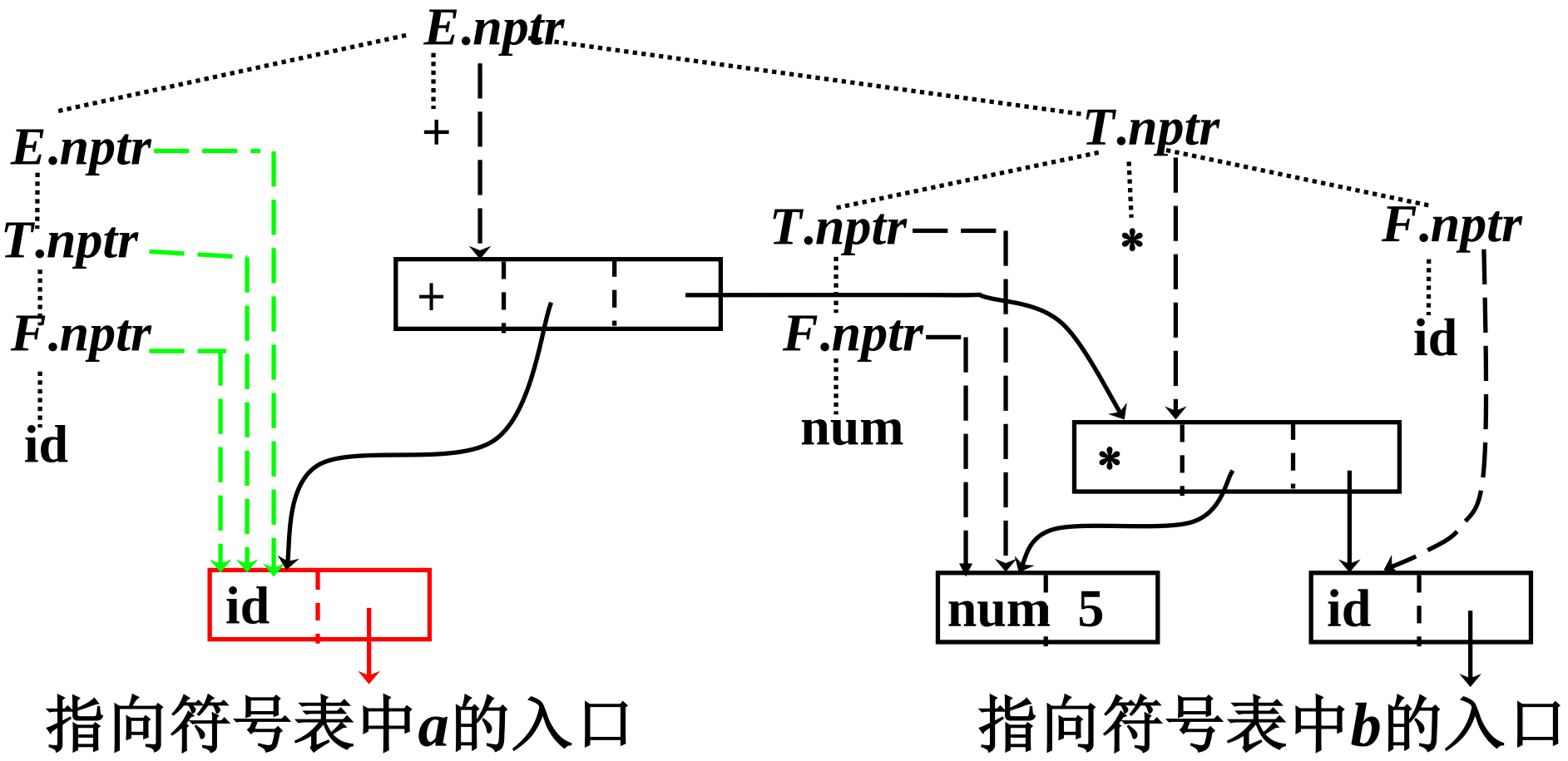
a+5**b*的语法树的构造


E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
num
*
id
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

a+5**b*的语法树的构造

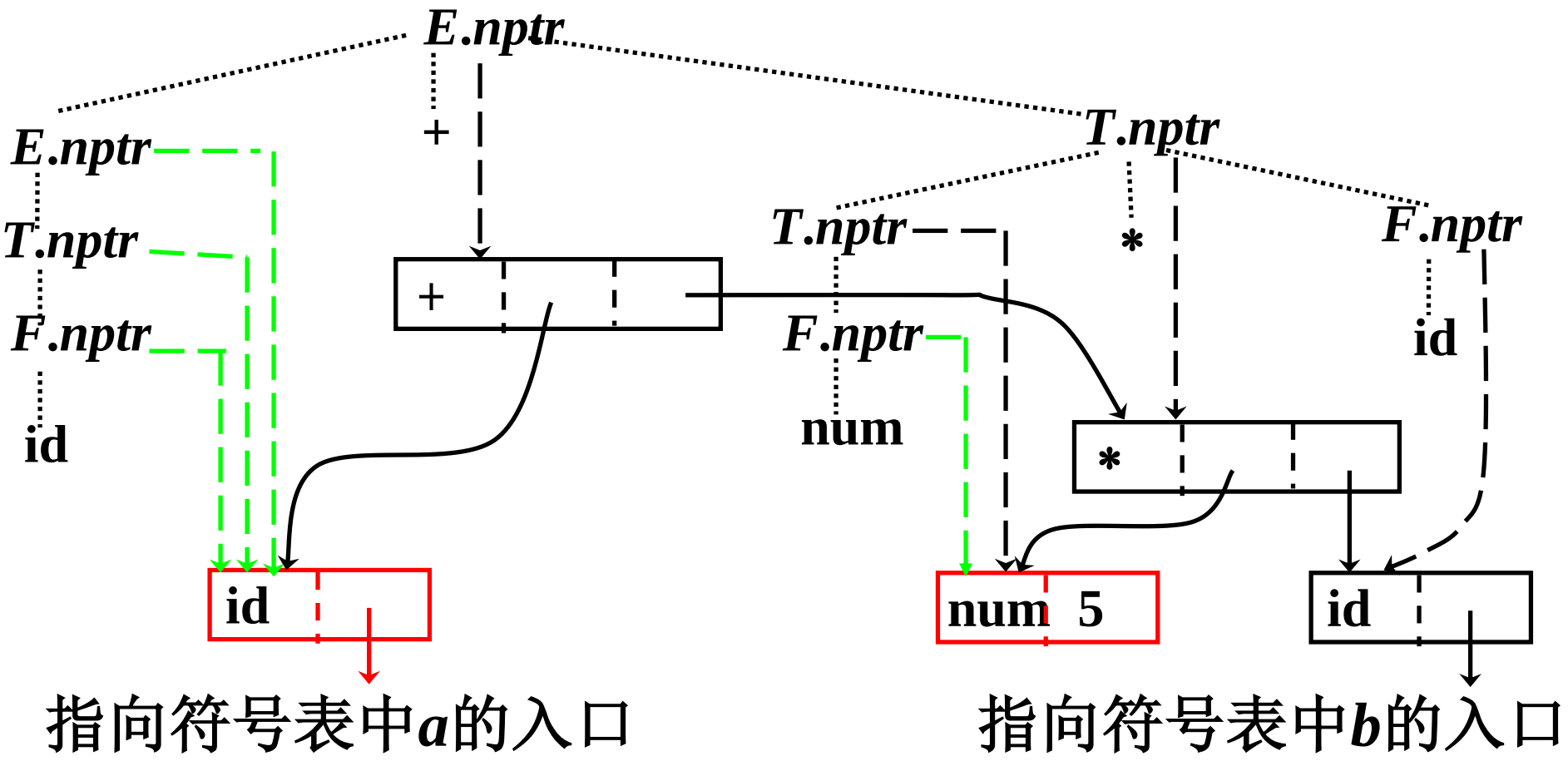

E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
id
num
*
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

a+5**b*的语法树的构造

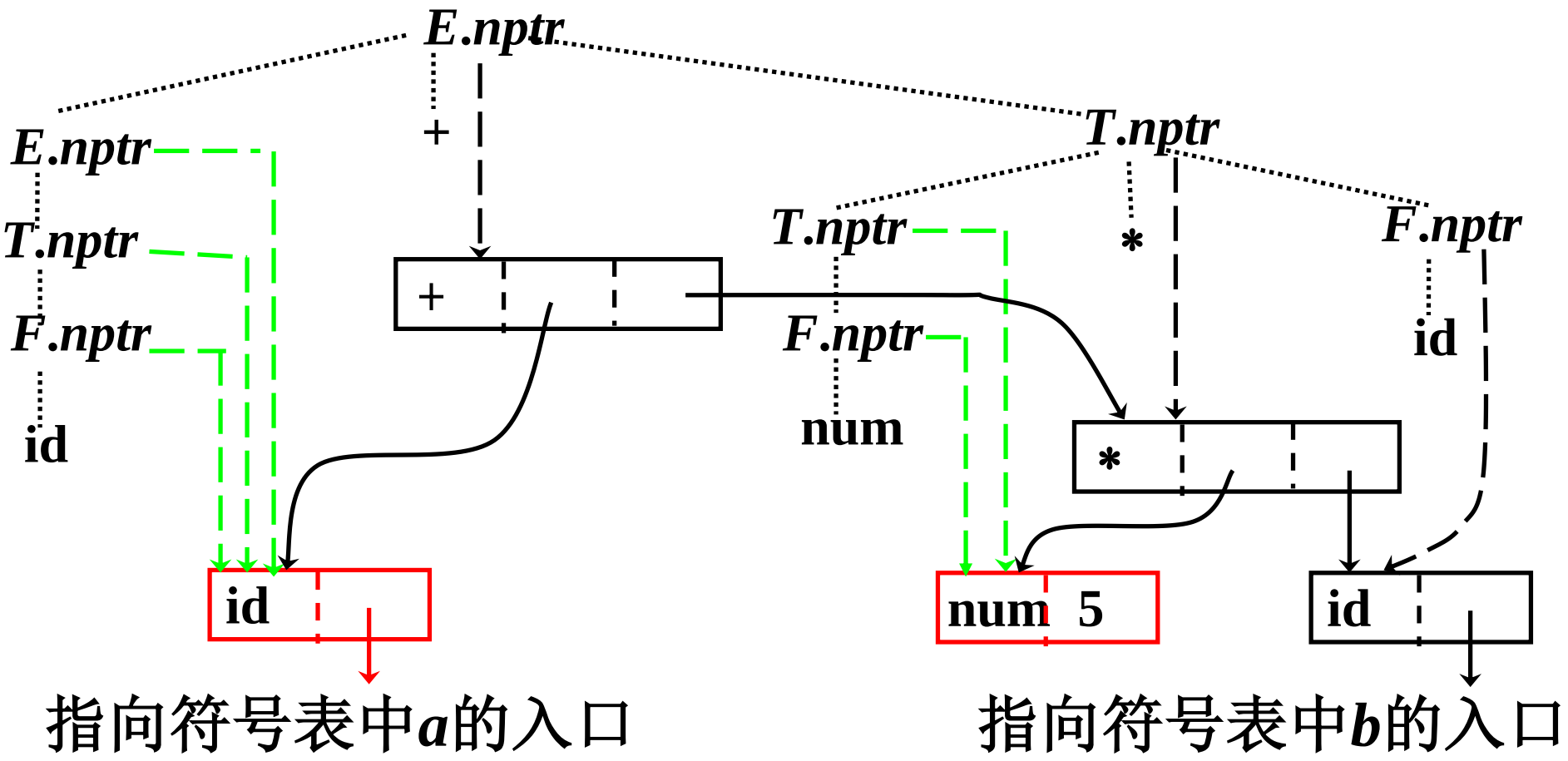

E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
num
*
id
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

S属性定义的语法树构造

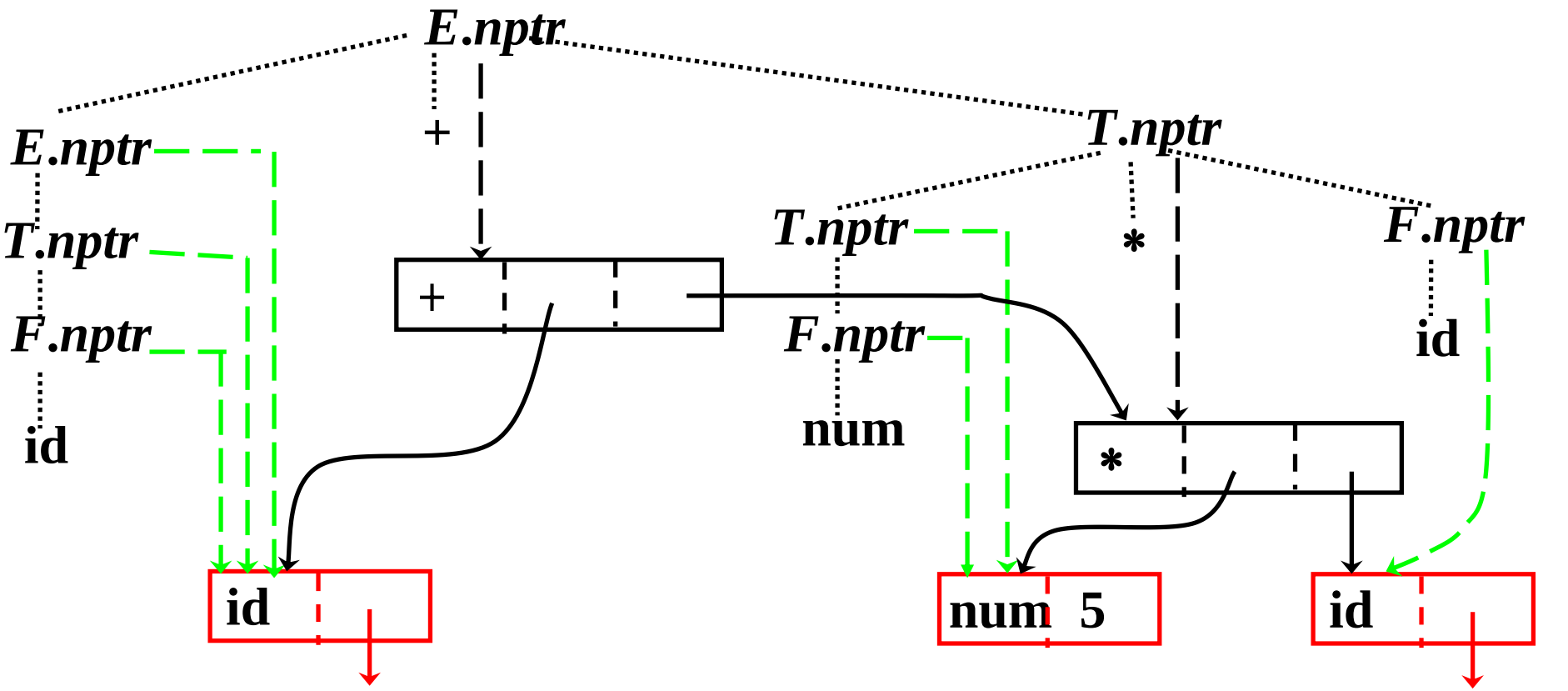
a+5**b*的语法树的构造



a+5**b*的语法树的构造


E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
num
*
id
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

a+5**b*的语法树的构造

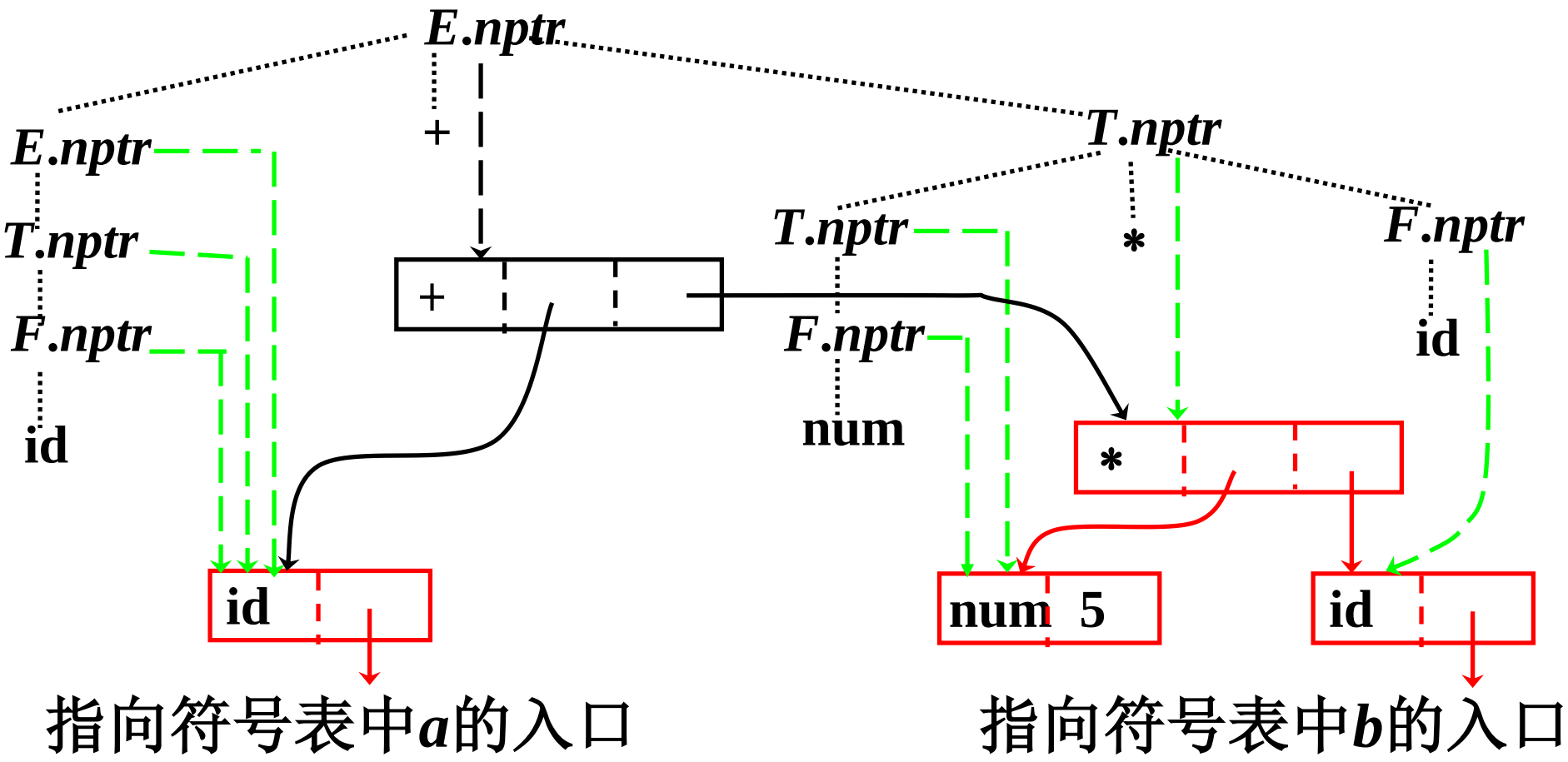


指向符号表中*a*的入口

指向符号表中*b*的入口

S属性定义的语法树构造

a+5**b*的语法树的构造

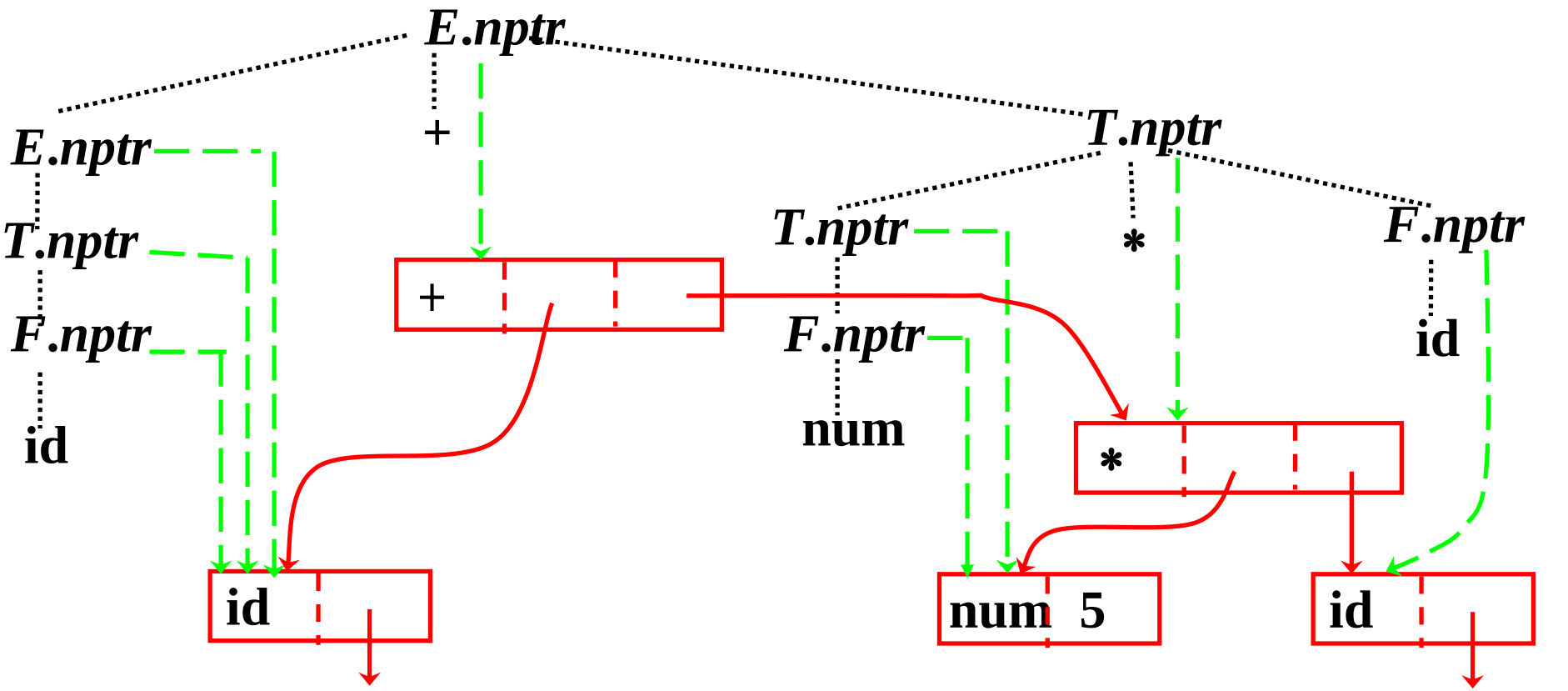


指向符号表中*a*的入口

指向符号表中*b*的入口

S属性定义的语法树构造

a+5**b*的语法树的构造



指向符号表中*a*的入口

指向符号表中*b*的入口

构造 a-4+c语法树的步骤

(1) p1:=mkleaf(id,entry a);

(2) p2:=mkleaf(num, 4);

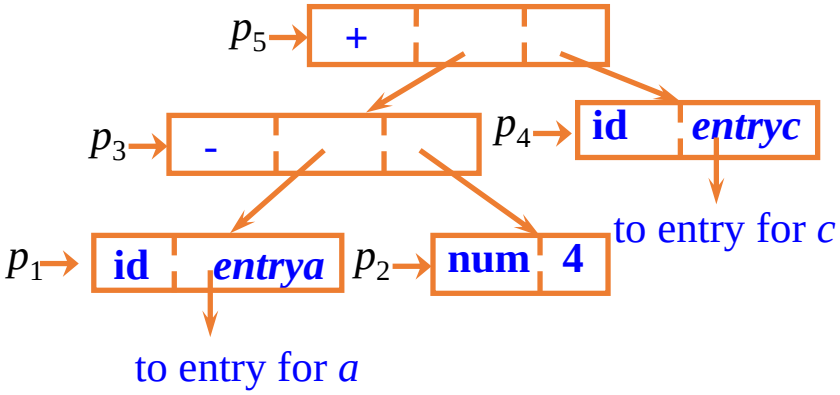
(3) p3:=mknode('-', p1,p2)

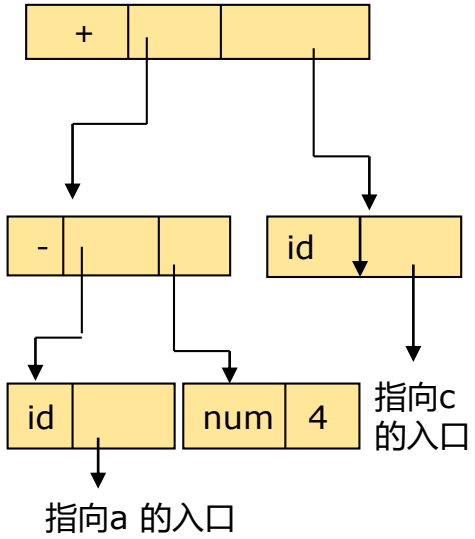
(4) p4:=mkleaf(id, entry c)

(5) p5:=mknode('+',p3,p4)

p_1, p_2, ..., p_5是指向结点的指针，

entry a 和 *entry c* 分别指向符号表中标识符*a*和*c*的指针。





❑考虑以下左递归文法

产 生 式	语 义 规 则
$E \rightarrow E_1 + T$	$E.nptr = mkNode(\text{'+'}, E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode(\text{'*'}, T_1.nptr, F.nptr)$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow \text{id}$	$F.nptr = mkLeaf(\text{id}, \text{id}.entry)$
$F \rightarrow \text{num}$	$F.nptr = mkLeaf(\text{num}, \text{num}.val)$

左递归的消除引起继承属性

❑首先消除左递归

$E \rightarrow E_1 + T$
$E \rightarrow T$
$T \rightarrow T_1 * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \text{id}$
$F \rightarrow \text{num}$

$T + T + T + \ldots$

$E \rightarrow TR$
$R \rightarrow +TR_1$
$R \rightarrow \varepsilon$
$T \rightarrow FW$
$W \rightarrow *FW_1$
$W \rightarrow \varepsilon$

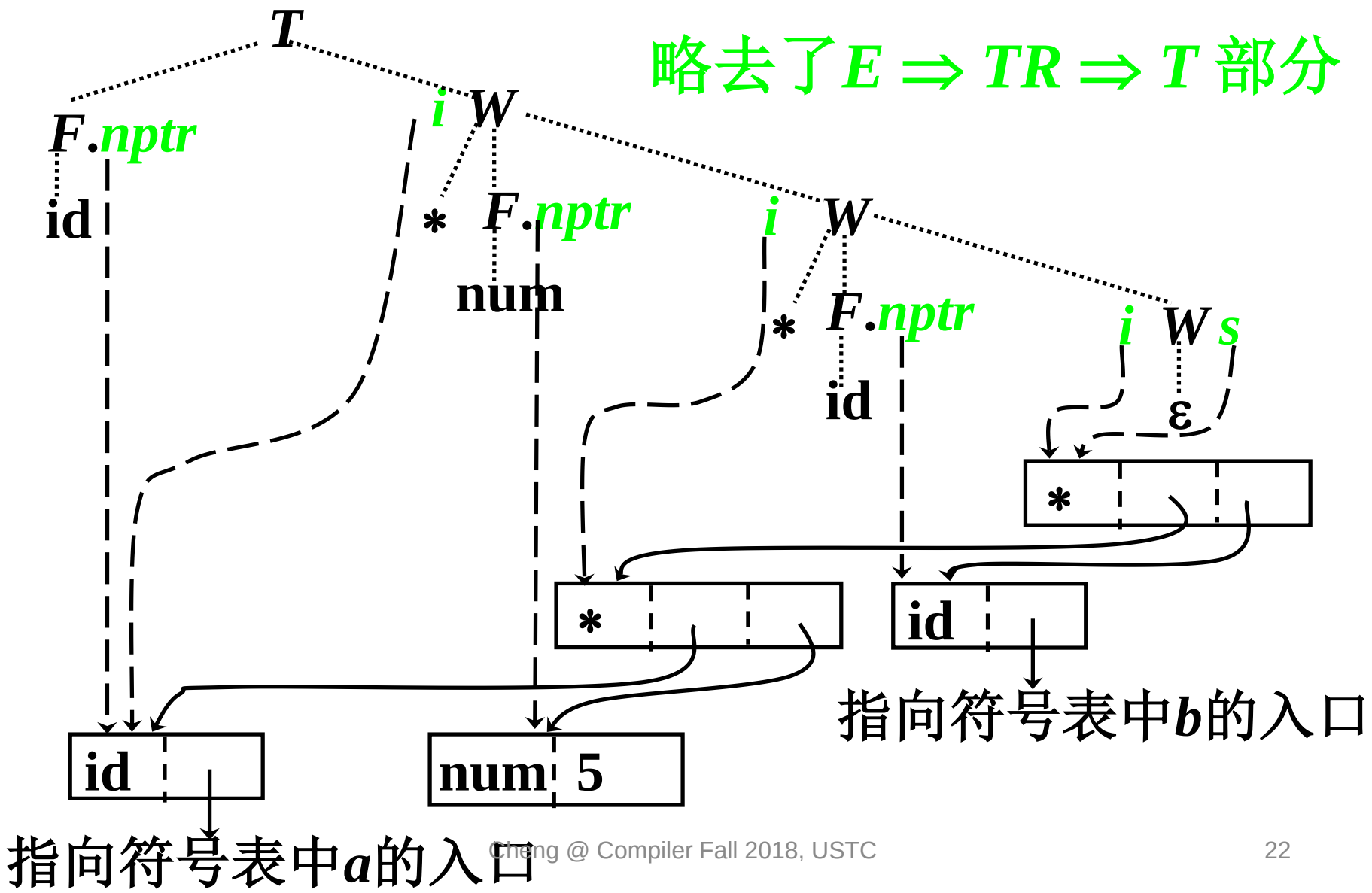
F产生式部分不再给出

左递归的消除引起继承属性

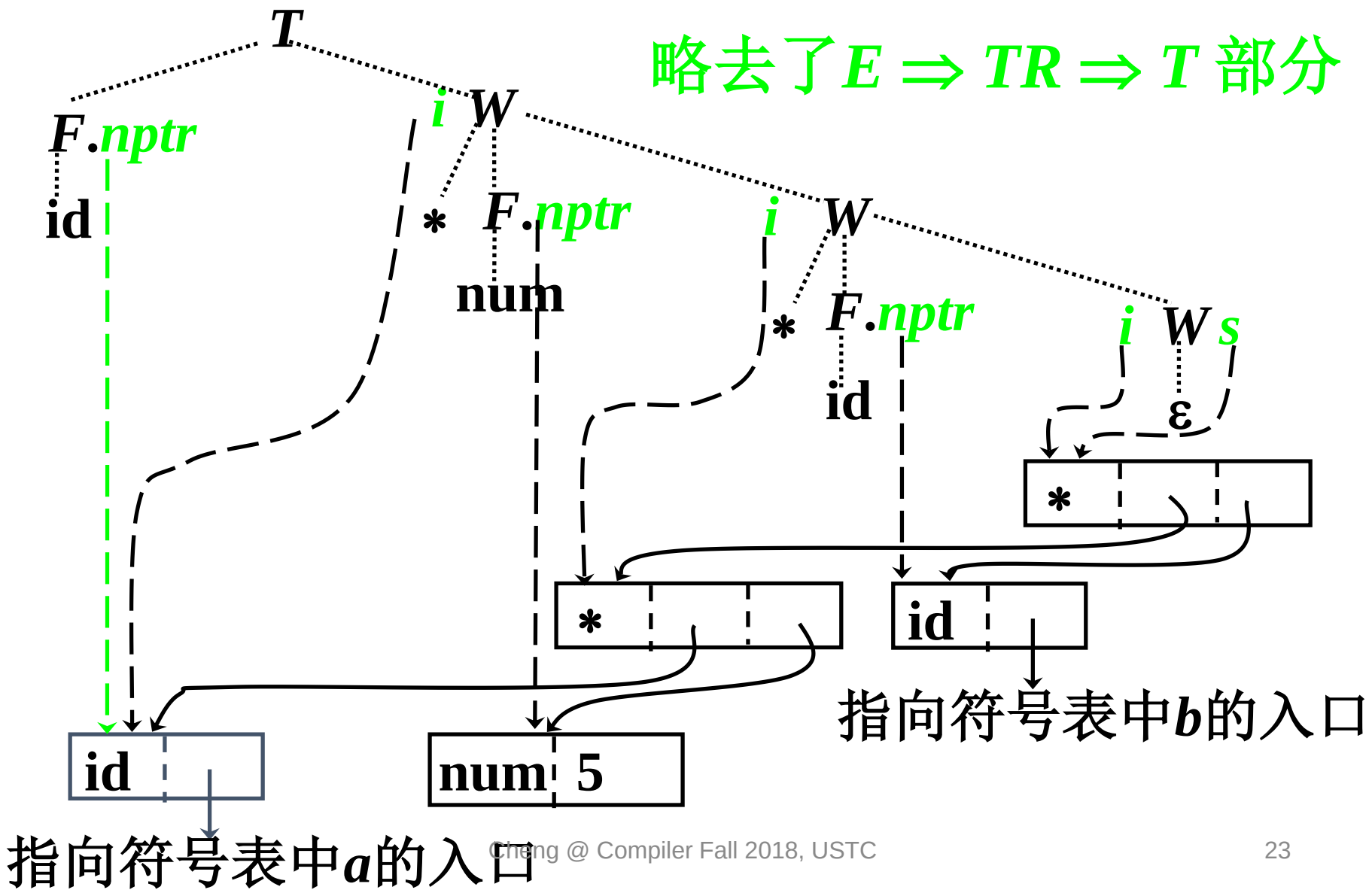
$E \rightarrow T$	$\{R.i = T.nptr\}$	$T + T + T + \ldots$
$\quad R$	$\{E.nptr = R.s\}$	
$R \rightarrow +$		
$\quad T$	$\{R_1.i = mkNode(\text{'+'}, R.i, T.nptr)\}$	
$\quad R_1$	$\{R.s = R_1.s\}$	
$R \rightarrow \varepsilon$	$\{R.s = R.i\}$	
$T \rightarrow F$	$\{W.i = F.nptr\}$	
$\quad W$	$\{T.nptr = W.s\}$	
$W \rightarrow *$		
$\quad F$	$\{W_1.i = mkNode(\text{'*'}, W.i, F.nptr)\}$	
$\quad W_1$	$\{W.s = W_1.s\}$	
$W \rightarrow \varepsilon$	$\{W.s = W.i\}$	

F 产生式部分不再给出

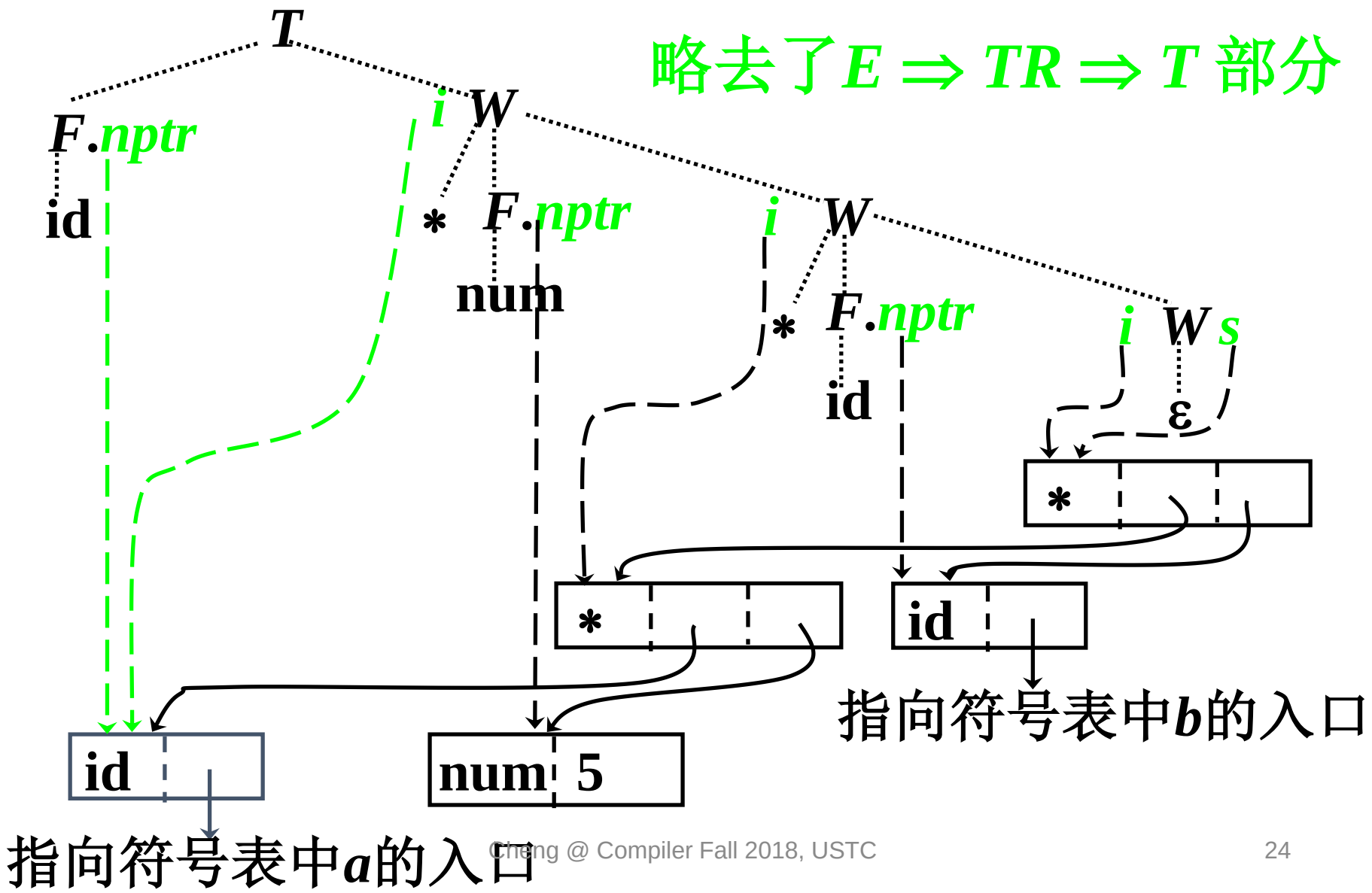
左递归的消除引起继承属性



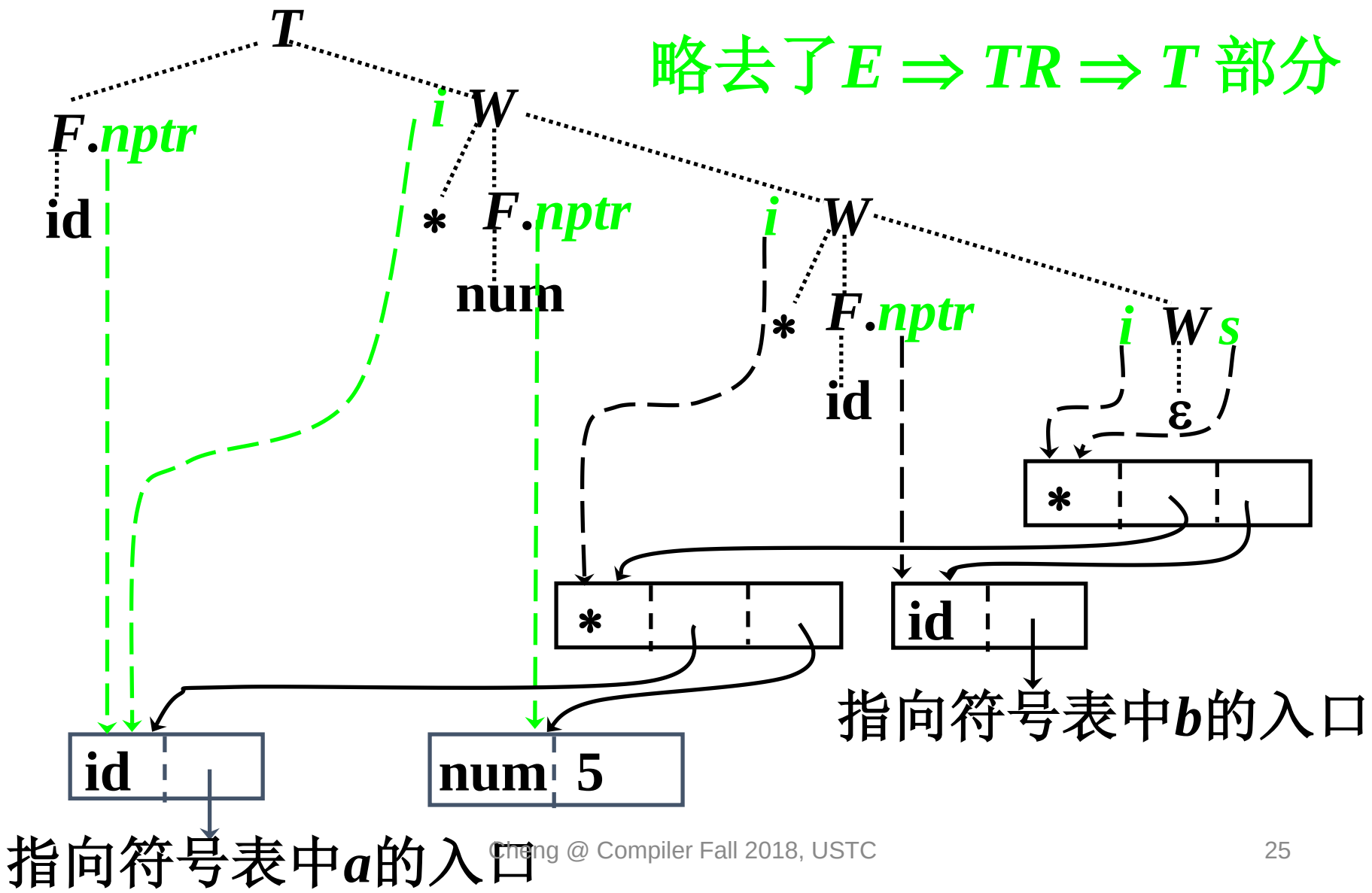
左递归的消除引起继承属性



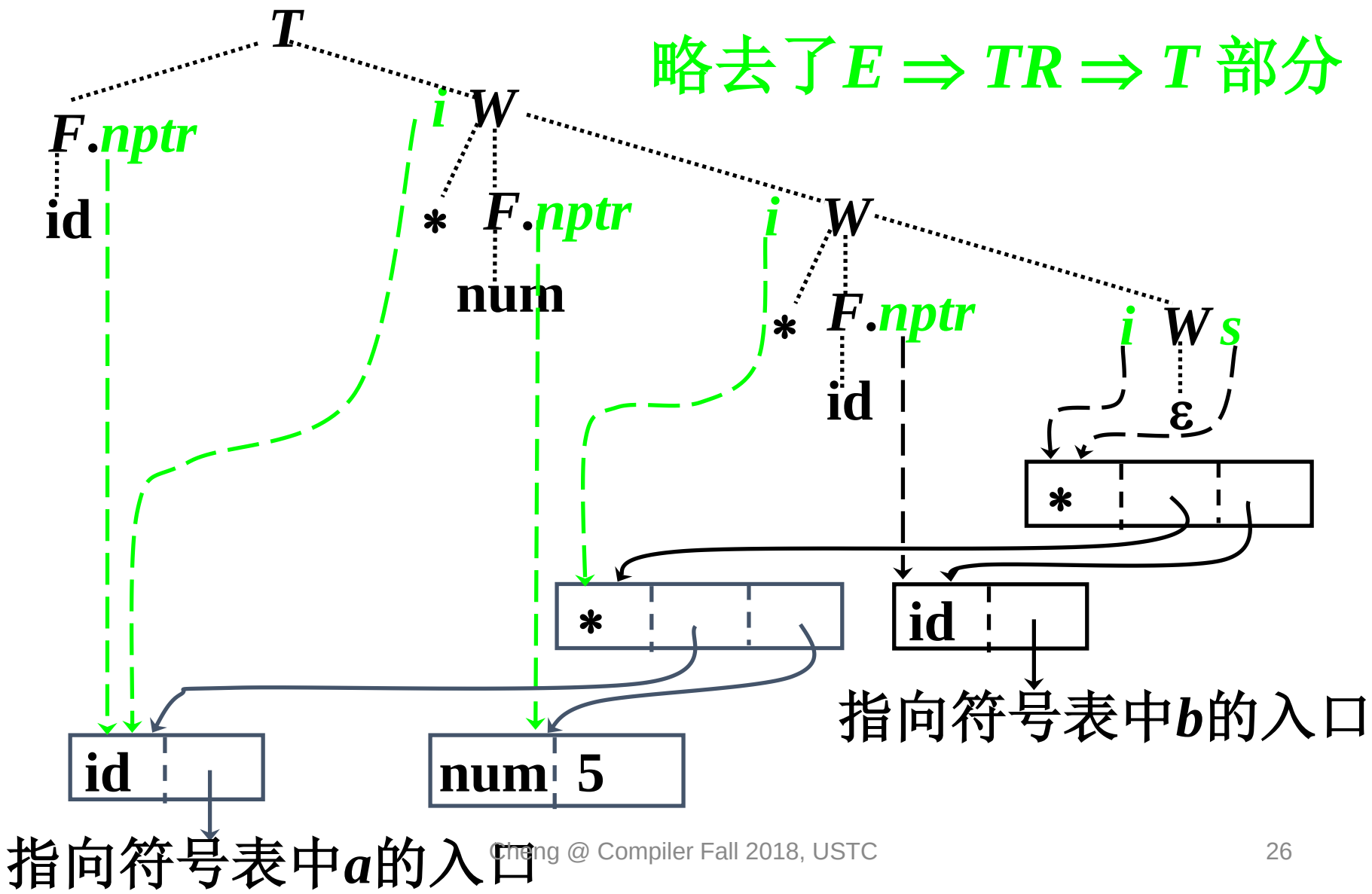
左递归的消除引起继承属性



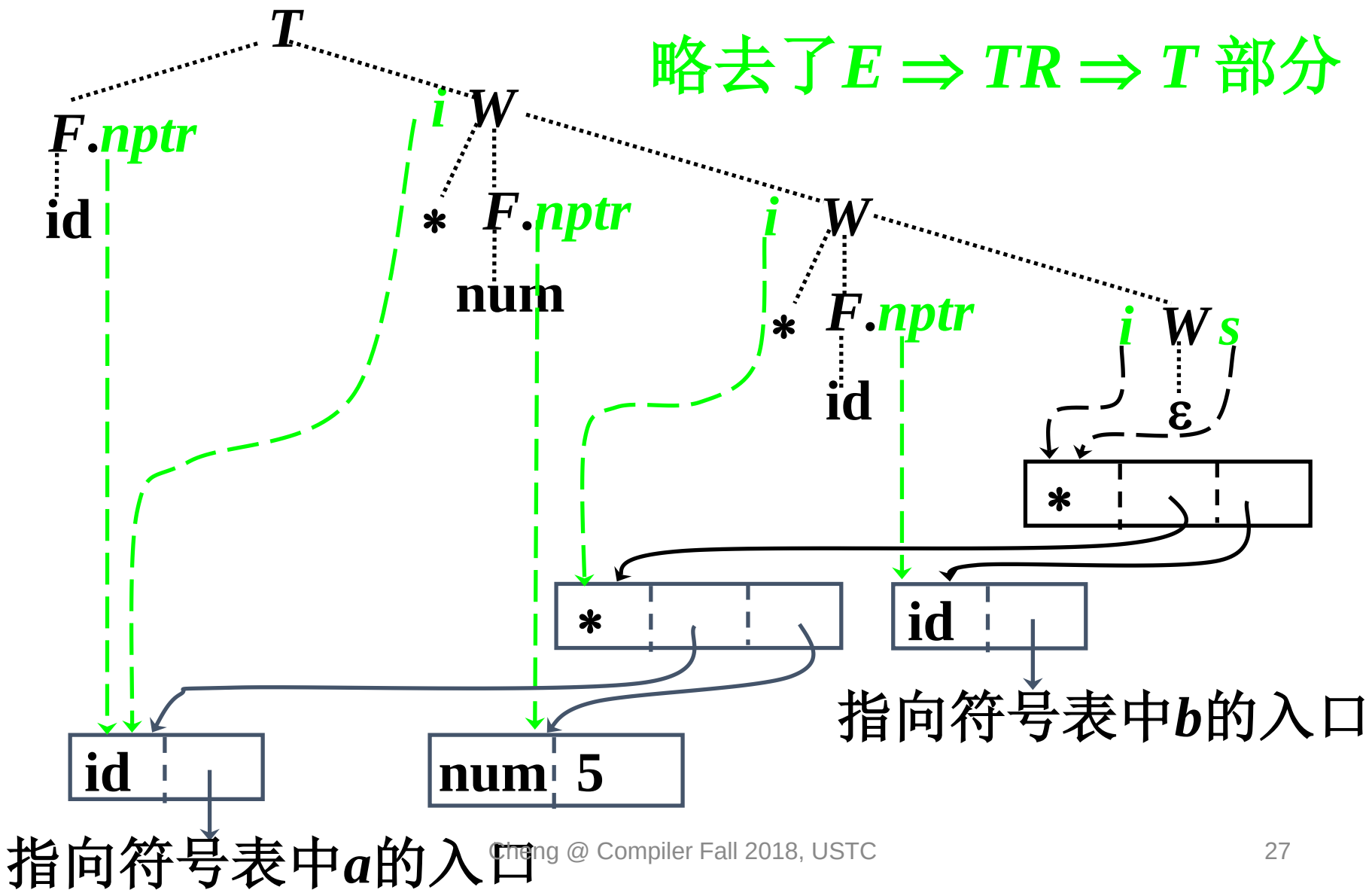
左递归的消除引起继承属性



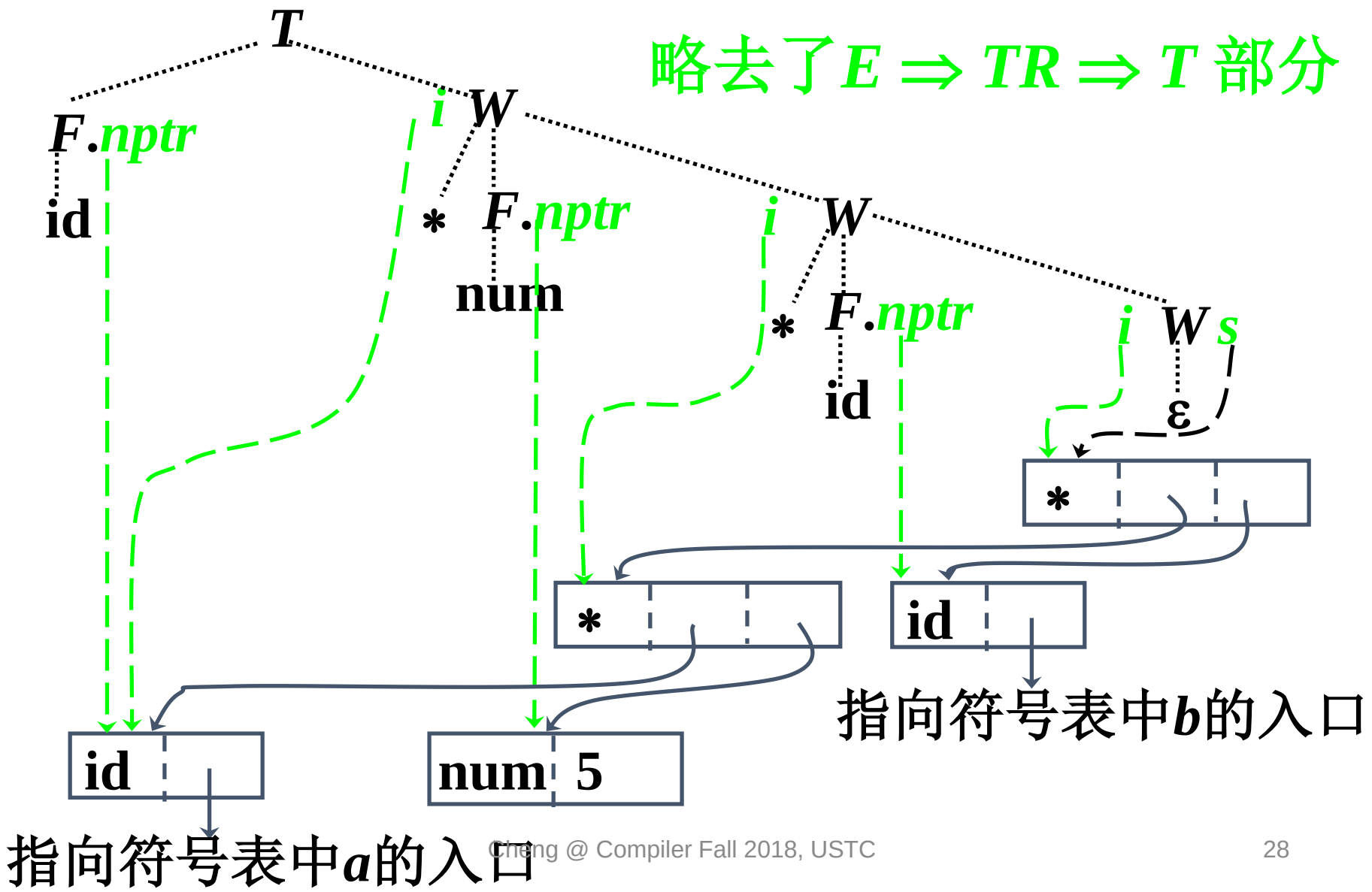
左递归的消除引起继承属性



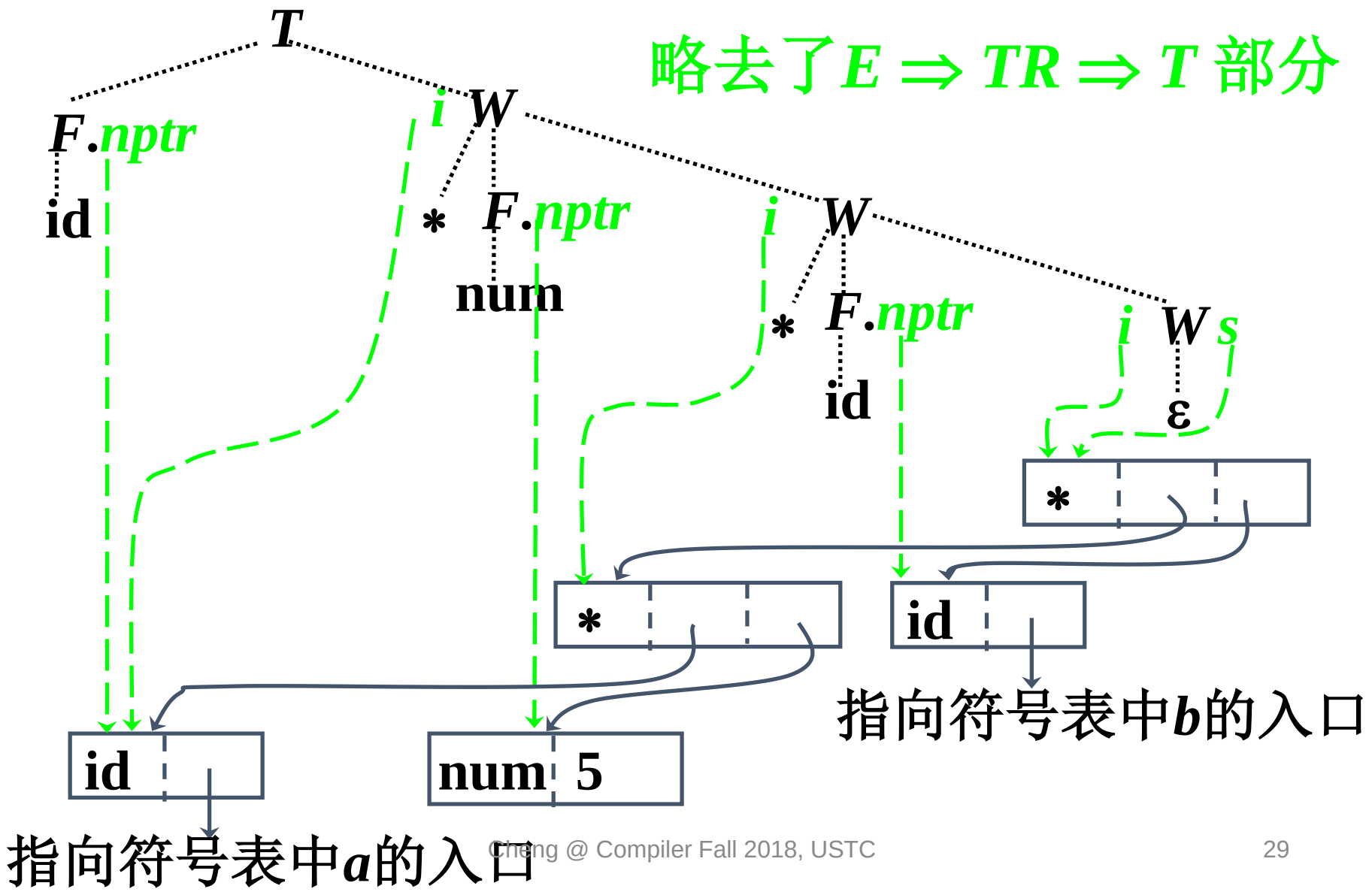
左递归的消除引起继承属性



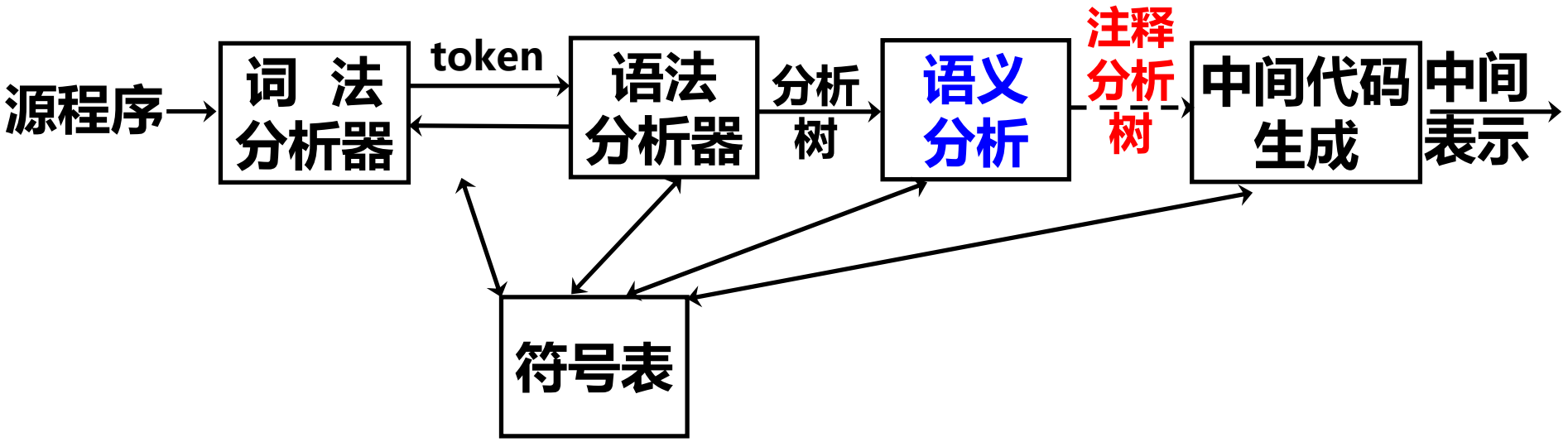
左递归的消除引起继承属性



左递归的消除引起继承属性



主要内容



- 语法分析树→ 抽象语法树
- 从语法制导定义到翻译方案
  - S属性定义的SDT
  - L属性定义的SDT

语法制导翻译方案

- 语法制导翻译方案(SDT)是在产生式右部中嵌入了程序片段(称为语义动作)的CFG
- SDT可以看作是SDD的具体实施方案

将S-SDD转换为SDT

- 将一个S-SDD转换为SDT的方法：将每个语义动作都放在产生式的最后

S-SDD

产生式	语义规则
(1) $L \to E$ n	$L.val = E.val$
(2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \to T$	$E.val = T.val$
(4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \to F$	$T.val = F.val$
(6) $F \to (E)$	$F.val = E.val$
(7) $F \to$ digit	$F.val =$ digit.*lexval*

SDT

(1) $L \to E$ n { $L.val = E.val$}
(2) $E \to E_1 + T${$E.val = E_1.val + T.val$}
(3) $E \to T$ { $E.val = T.val$}
(4) $T \to T_1 * F$ { $T.val = T_1.val \times F.val$}
(5) $T \to F$ { $T.val = F.val$}
(6) $F \to (E)$ { $F.val = E.val$}
(7) $F \to$ digit { $F.val =$ digit.*lexval*}

S-属性定义的SDT实现

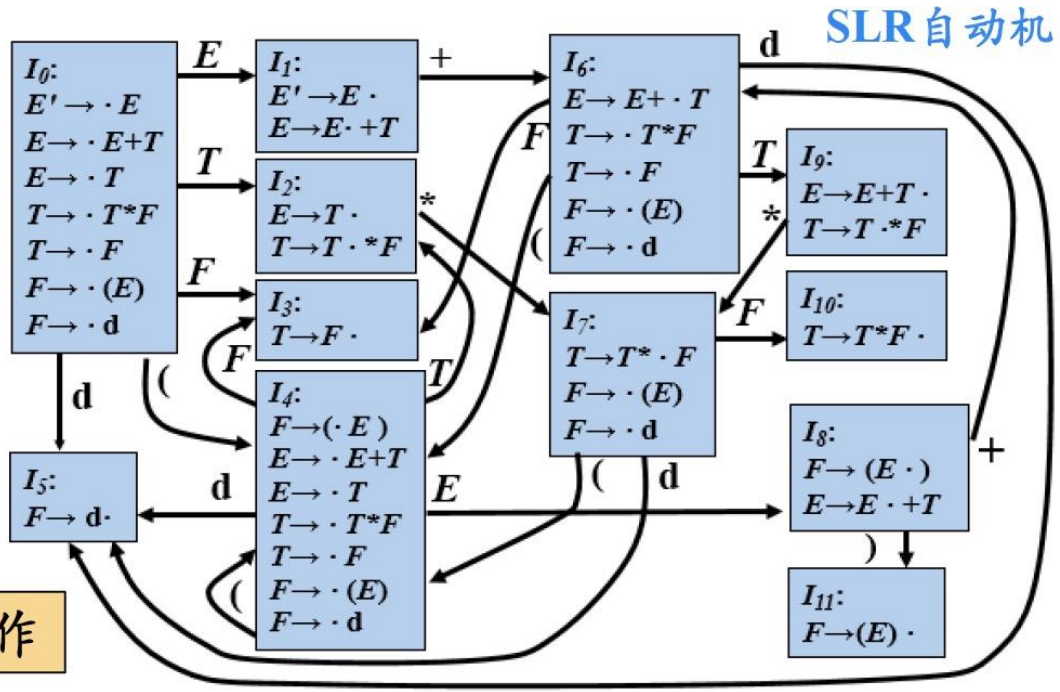
- 综合属性可通过自底向上的LR方法来计算
- 当归约发生时执行相应的语义动作

➢ 例

S-SDD

产生式	语义规则
(1) $L \to E$ n	$L.val = E.val$
(2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \to T$	$E.val = T.val$
(4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \to F$	$T.val = F.val$
(6) $F \to (E)$	$F.val = E.val$
(7) $F \to$ digit	$F.val =$ digit.*lexval*

当归约发生时执行相应的语义动作



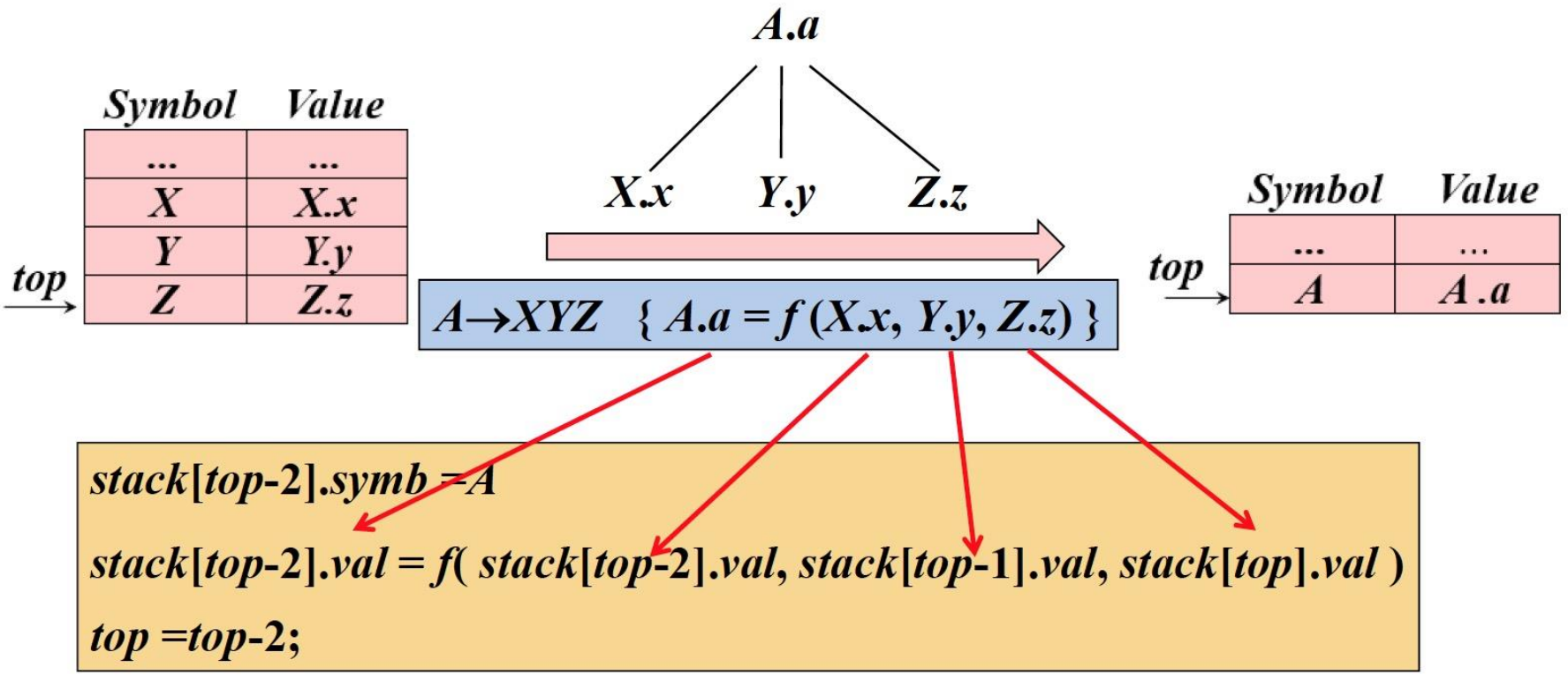
❑可以通过扩展的LR语法分析栈来实现

- ❖在分析栈中使用一个附加的域来存放综合属性值。若支持多个属性，那么可以在栈中存放指针
- ❖此时，分析栈可以看成一个栈，栈元素包含状态、文法符号、综合属性三个域；分析栈也可以看成三个栈，分别是状态栈、文法符号栈、综合属性栈，分开看的理由是，入栈出栈并不完全同步

S-属性定义的SDT实现

❑可以通过扩展的LR语法分析栈来实现

❖语义翻译对应栈的操作



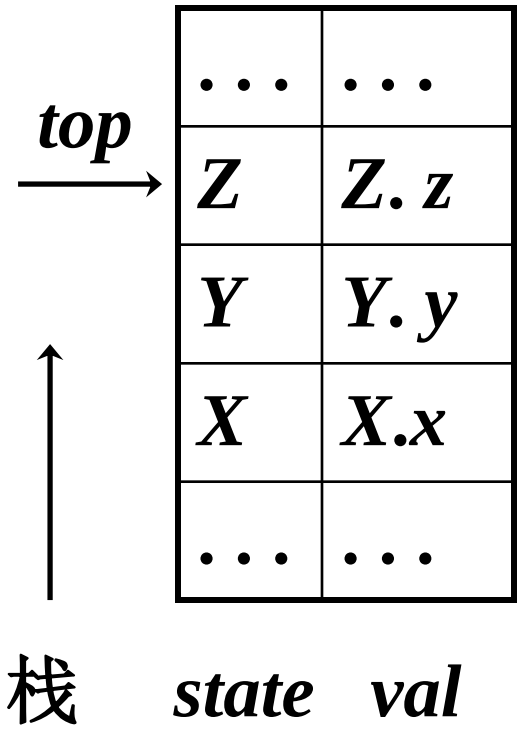
SLR分析栈中实现计算器

❑桌面计算器的SDD和SDT定义如下：

产生式	语义动作	
(1) $E' \rightarrow E$	print(*E.val*)	{ print (*stack*[*top*].*val*);}
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$	{ *stack*[*top*-2].*val* = *stack*[*top*-2].*val* + *stack*[*top*].*val* ; *top*=*top*-2; }
(3) $E \rightarrow T$	$E.val = T.val$	
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$	{ *stack*[*top*-2].*val* = *stack*[*top*-2].*val* × *stack*[*top*].*val* ; *top*=*top*-2; }
(5) $T \rightarrow F$	$T.val = F.val$	
(6) $F \rightarrow (E)$	$F.val = E.val$	{ *stack*[*top*-2].*val* = *stack*[*top*-1].*val*; *top*=*top*-2; }
(7) $F \rightarrow$ digit	$F.val =$ digit.*lexval*	

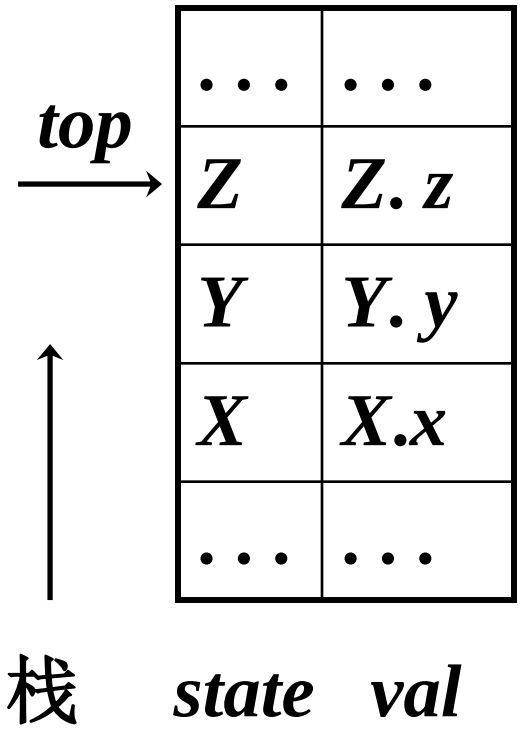
SLR分析栈中实现计算器

❑简单计算器的语法制导定义改成栈操作代码



产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

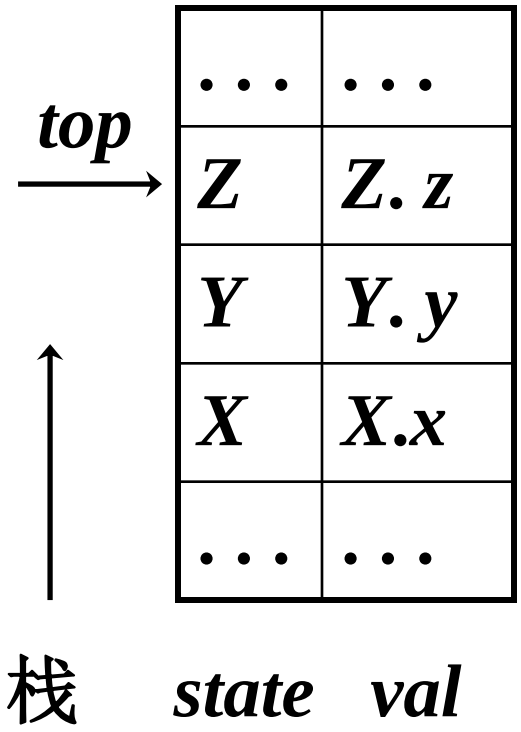
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

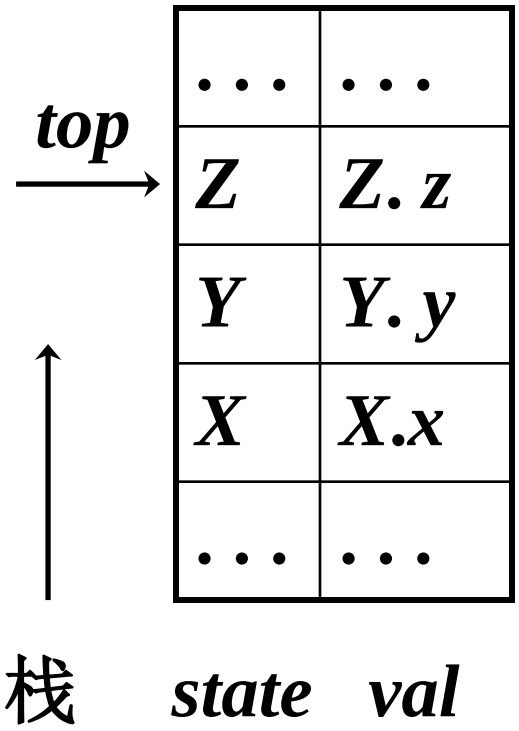
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

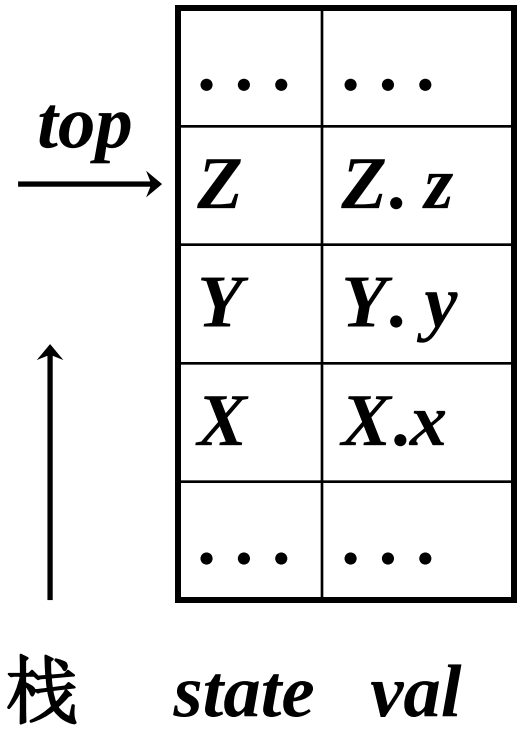
SLR分析栈中实现计算器

❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	*T.val* = T_1*.val* * *F.val*
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

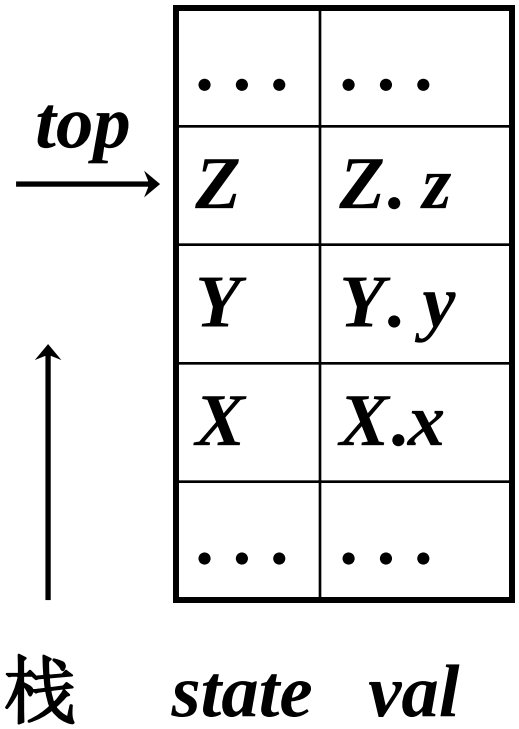
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

SLR分析栈中实现计算器

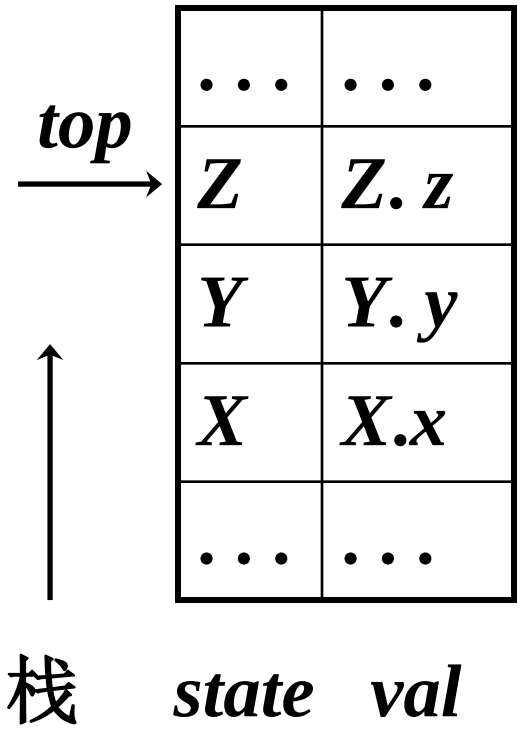
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

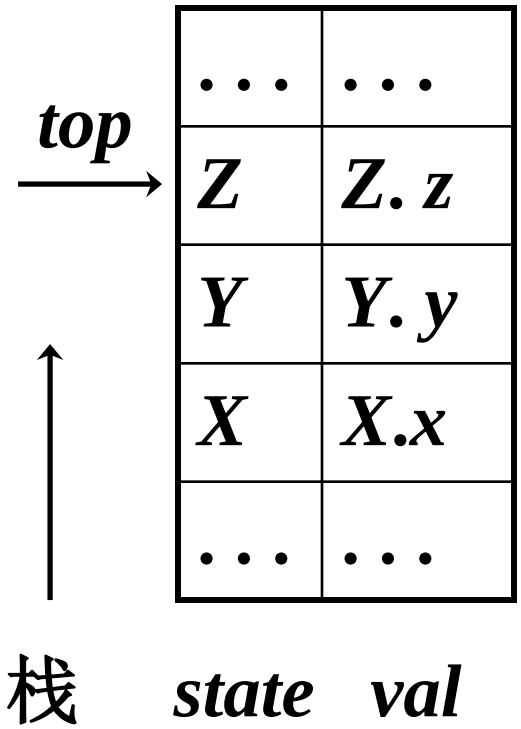
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \to E$ n	*print* (*val* [*top*–1])
$E \to E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \to T$	值不变，无动作
$T \to T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \to F$	值不变，无动作
$F \to (E)$	*F.val* = *E.val*
$F \to$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

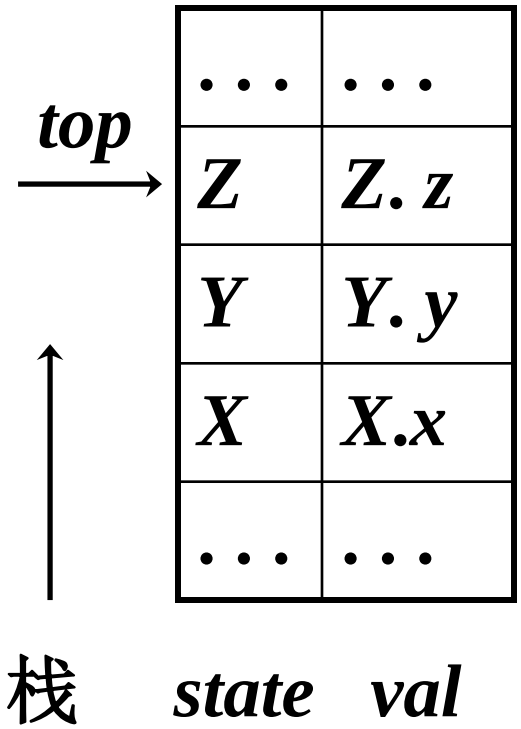
❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	值不变，无动作
$F \rightarrow (E)$	*val* [top –2] =*val* [top –1]
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

❑简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	值不变，无动作
$F \rightarrow (E)$	*val* [top –2] =*val* [top –1]
$F \rightarrow$ digit	值不变，无动作

翻译输入3*5+4n

输入	state	val	使用的产生式
3*5+4n	-	-	
*5+4n	3	3	
*5+4n	F	3	F→digit
*5+4n	T	3	T→F
5+4n	T*	3*	
+4n	T* 5	3*5	
+4n	T* F	3*5	F →digit

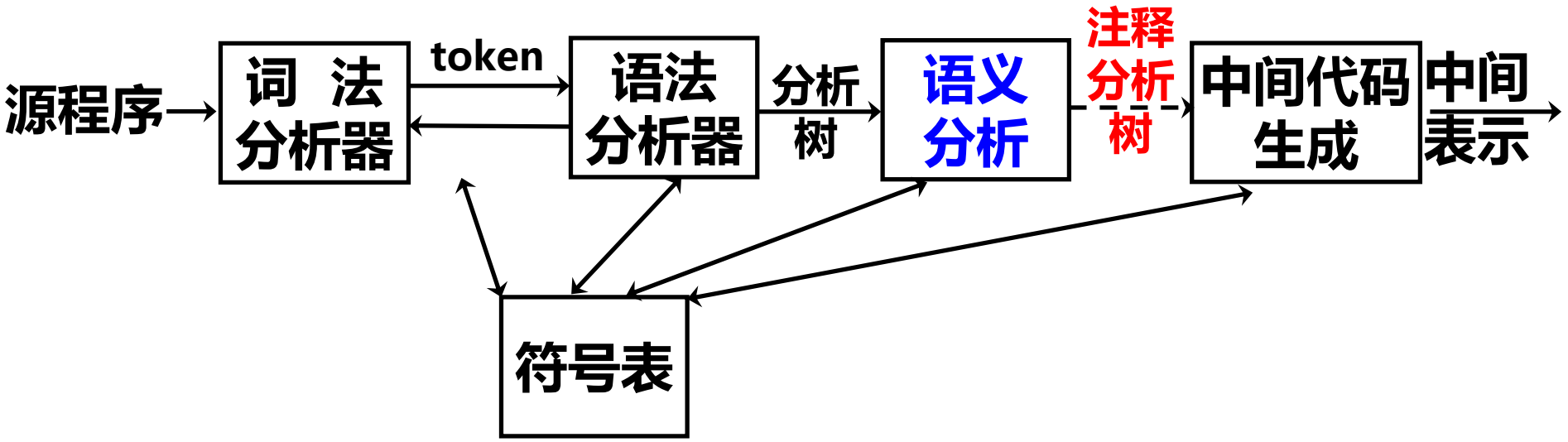
翻译输入3*5+4n

+4n	T	15	T→ T*F
+4n	E	15	E→ T
4n	E+	15+	
n	E+4	15+4	
n	E+F	15+4	F → digit
n	E+T	15+4	T→ F
n	E	19	E→ E+T
	En	19 -	
	L	19	L→ En

总结

- 采用自底向上分析，例如LR分析，首先给出S-属性定义，然后，把S-属性定义变成可执行的代码段，这就构成了翻译程序。
- 随着语法分析的进行，归约前调用相应的语义子程序，完成翻译的任务。

主要内容



- 语法分析树→ 抽象语法树
- 从语法制导定义到翻译方案
  - S属性定义的SDT
  - L属性定义的SDT

L属性定义的自上而下计算

- 边分析边翻译的方式能否用于继承属性?
  - 属性的计算次序一定受分析方法所限定的分析树结点建立次序的限制
  - 分析树的结点是自左向右生成
  - 如果属性信息是自左向右流动，那么就有可能在分析的同时完成属性计算

L属性定义的自上而下计算

- 如果每个产生式$A \rightarrow X_1 \ldots X_{j-1} X_j \ldots X_n$的每条语义规则计算的属性是$A$的综合属性；或者是$X_j$的继承属性，但它仅依赖：
  - 该产生式中X_j左边符号$X_1, X_2, \ldots, X_{j-1}$的属性；
  - A的继承属性

- S属性定义属于L属性定义

变量类型声明的语法制导定义是一个L属性定义

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addType(\text{id}.entry, L.in)$

L属性定义的自上而下计算

❑翻译方案

例　把有加和减的中缀表达式翻译成后缀表达式

如果输入是8+5 –2，则输出是8 5 + 2 –

$E \rightarrow T\ R$

$R \rightarrow$ addop T {*print* (addop.*lexeme*)} $R_1 \mid \varepsilon$

$T \rightarrow$ num {*print* (num.*val*)}

$E \Rightarrow T\ R \Rightarrow$ num {*print* (8)} R

$\Rightarrow$ num{*print* (8)}addop T{*print* (+)}R

$\Rightarrow$ num{*print*(8)}addop num{*print*(5)}{*print* (+)}R

… {*print*(8)}{*print*(5)}{*print*(+)}addop T{*print*(–)}R

… {*print*(8)}{*print*(5)}{*print*(+)}{*print*(2)}{*print*(–)}

中国科学技术大学
University of Science and Technology of China
USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
语法制导翻译 II

有些时候不是因为看到希望才坚持，而是因为坚持久了才看到了希望。

—— Loved by Cheng Li