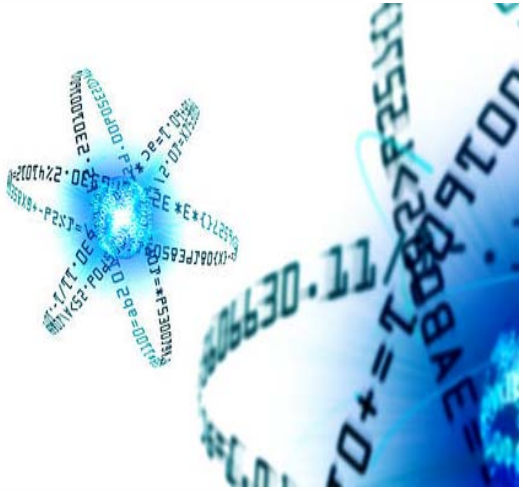
编译原理

Compilers – Principles, Techniques and Tools



计算机科学与技术学院

第五章　语法制导的翻译

- 本章内容

  1、介绍语义描述的一种形式方法：语法制导的翻译，它包括两种具体形式

  - 语法制导的定义
  - 翻译方案

  2、介绍语法制导翻译的实现方法

5.1 语法制导的定义

- 例 简单计算器的语法制导定义

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*

5.1 语法制导的定义

5.1.1 语法制导定义的形式

- 基础文法
- 每个文法符号有一组属性
- 每个文法产生式$A \rightarrow \alpha$有

  一组形式为$b=f(c_1, c_2, \ldots, c_k)$的语义规则，其中$b$和$c_1, c_2, \ldots, c_k$是该产生式文法符号的属性，$f$是函数
- 综合属性：如果b是A的属性，$c_1, c_2, \ldots, c_k$是产生式右部文法符号的属性或A的其它属性
- 继承属性：如果b是右部某文法符号X的属性

5.1 语法制导的定义

5.1.2 综合属性

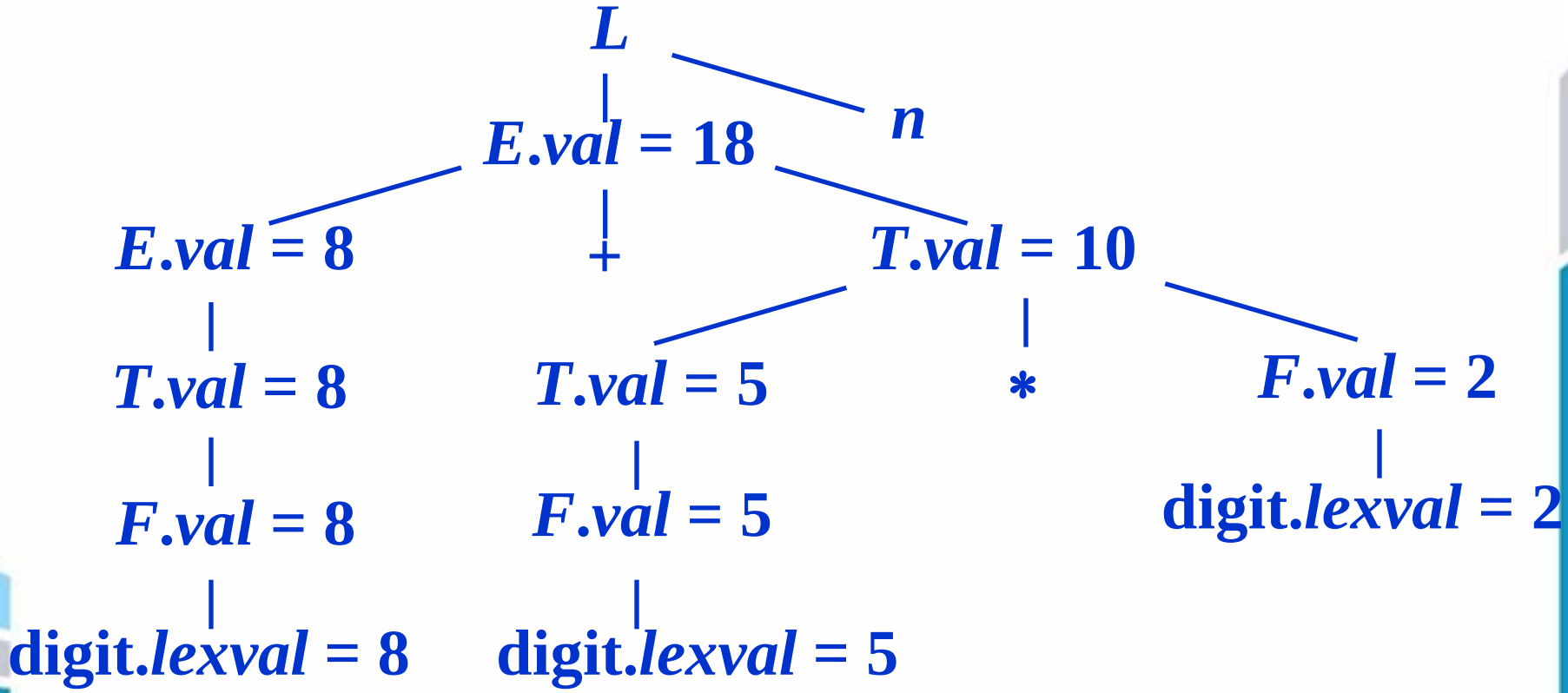
S属性定义：仅使用综合属性的语法制导定义

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*

5.1 语法制导的定义

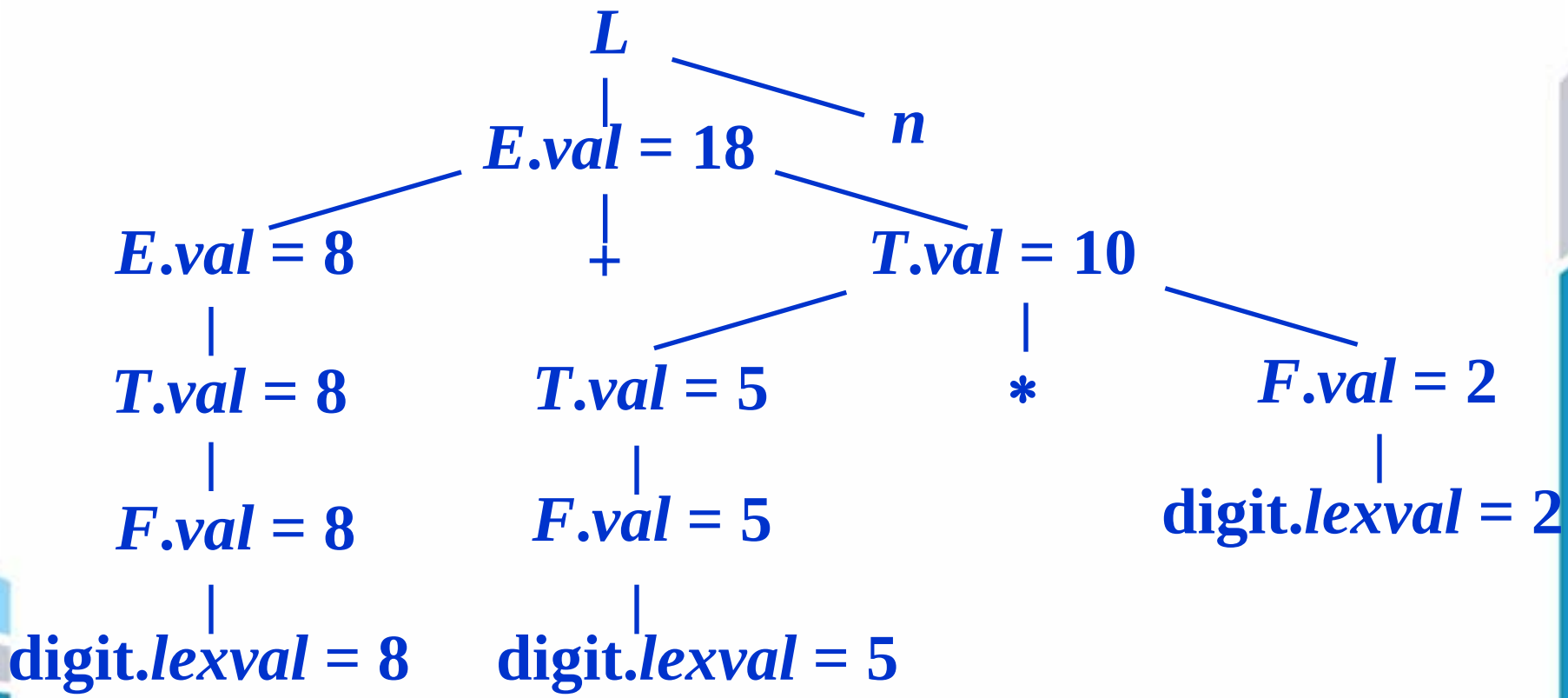
注释分析树：结点的属性值都标注出来的分析树

8+5*2 n的注释分析树



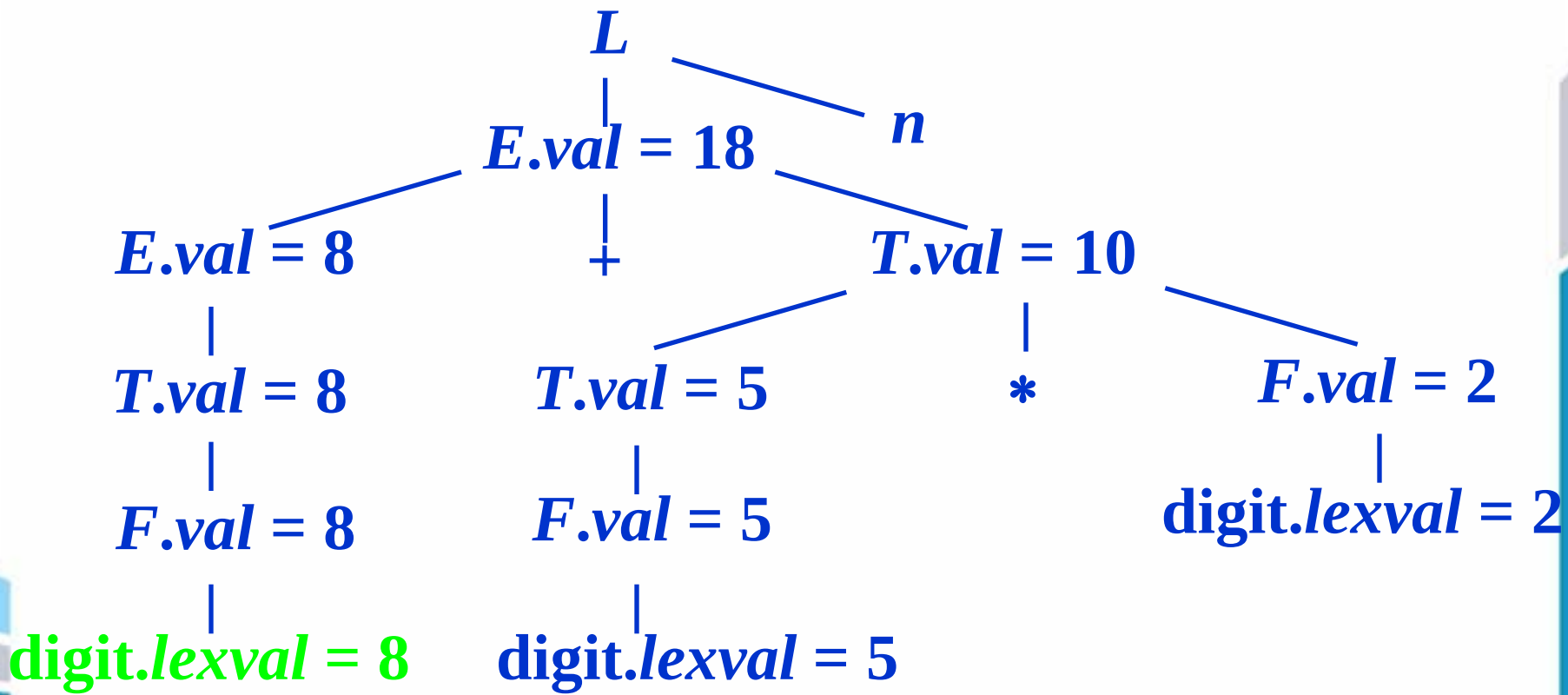
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



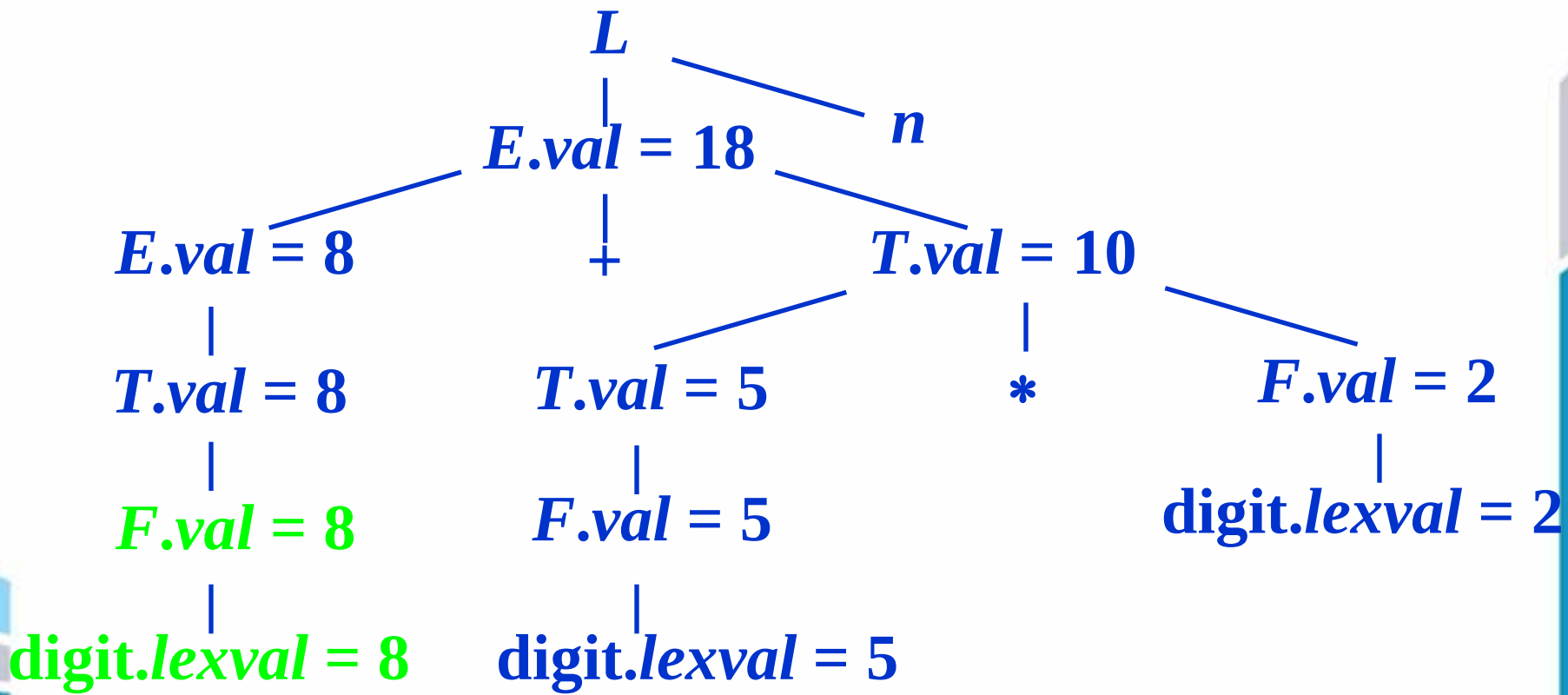
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



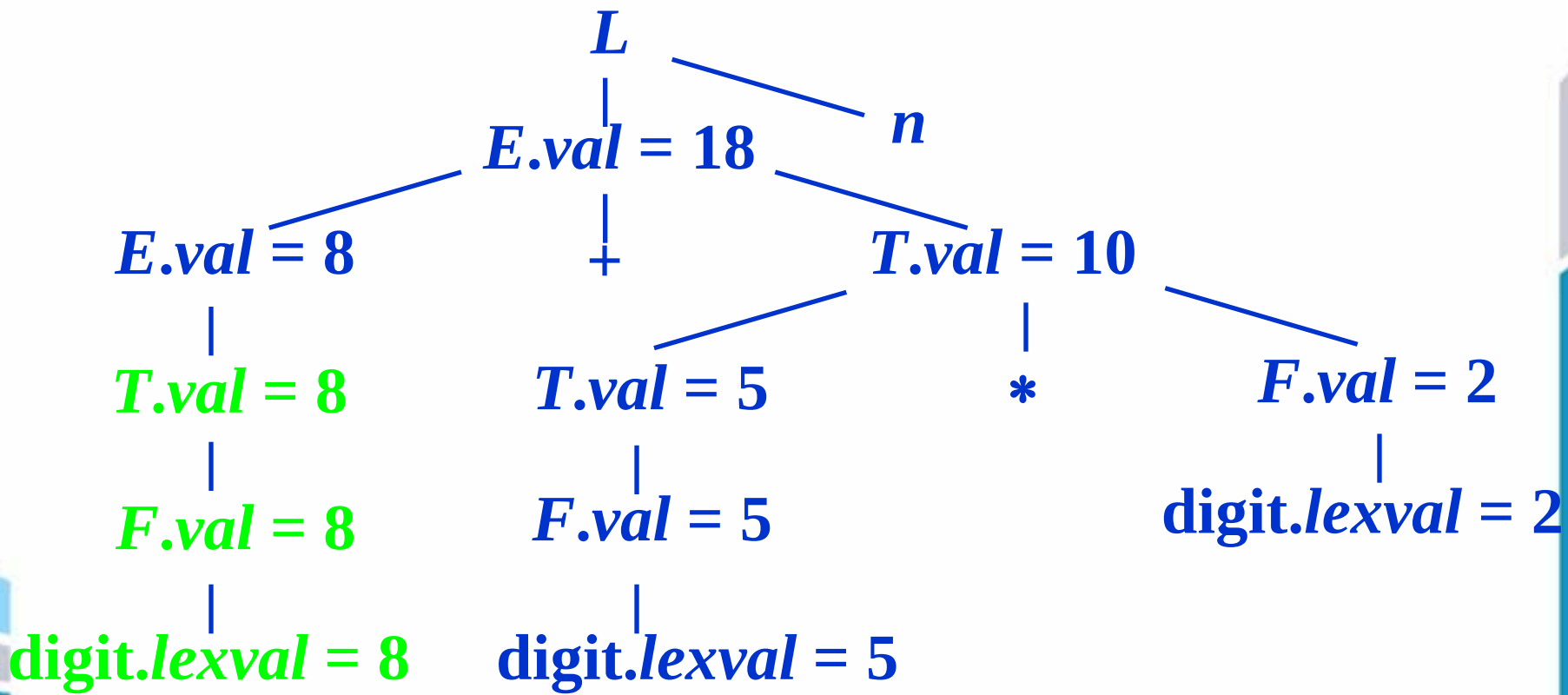
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



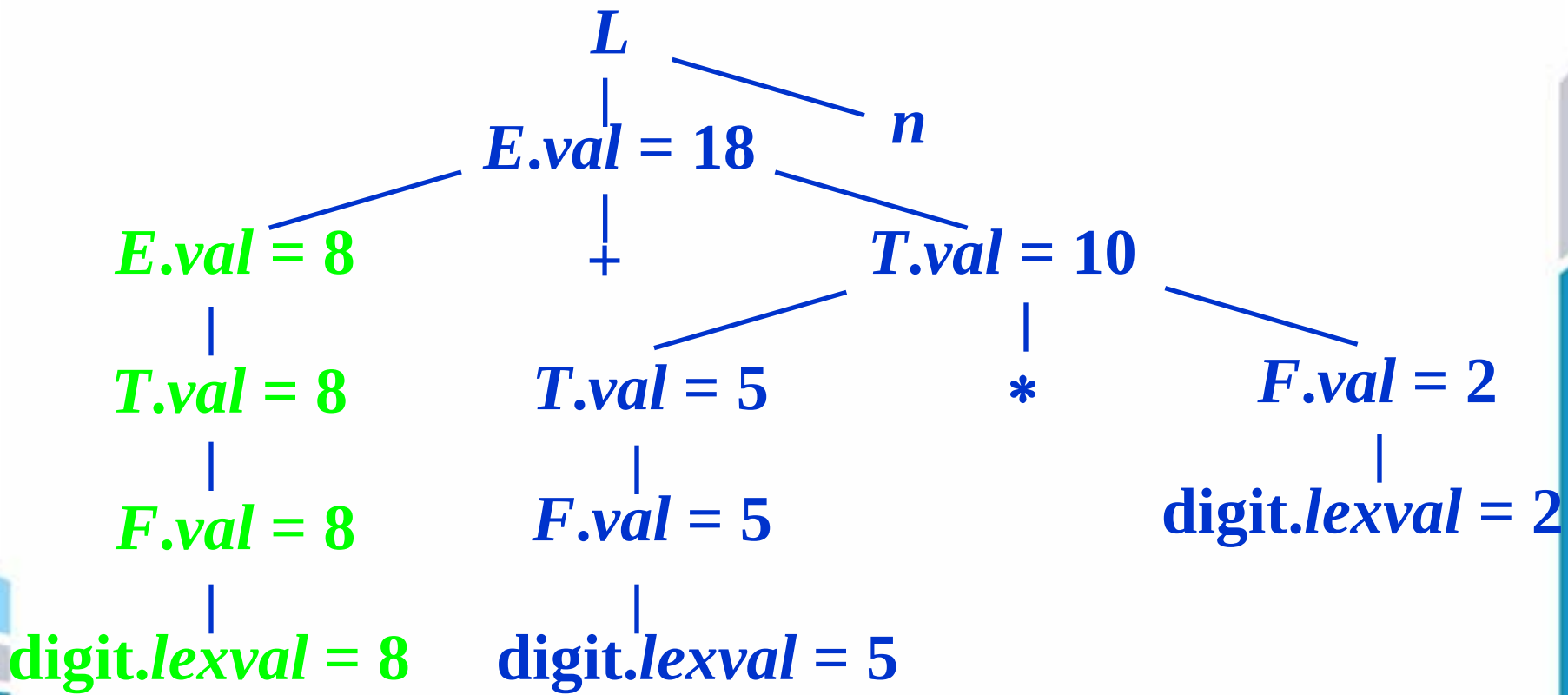
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



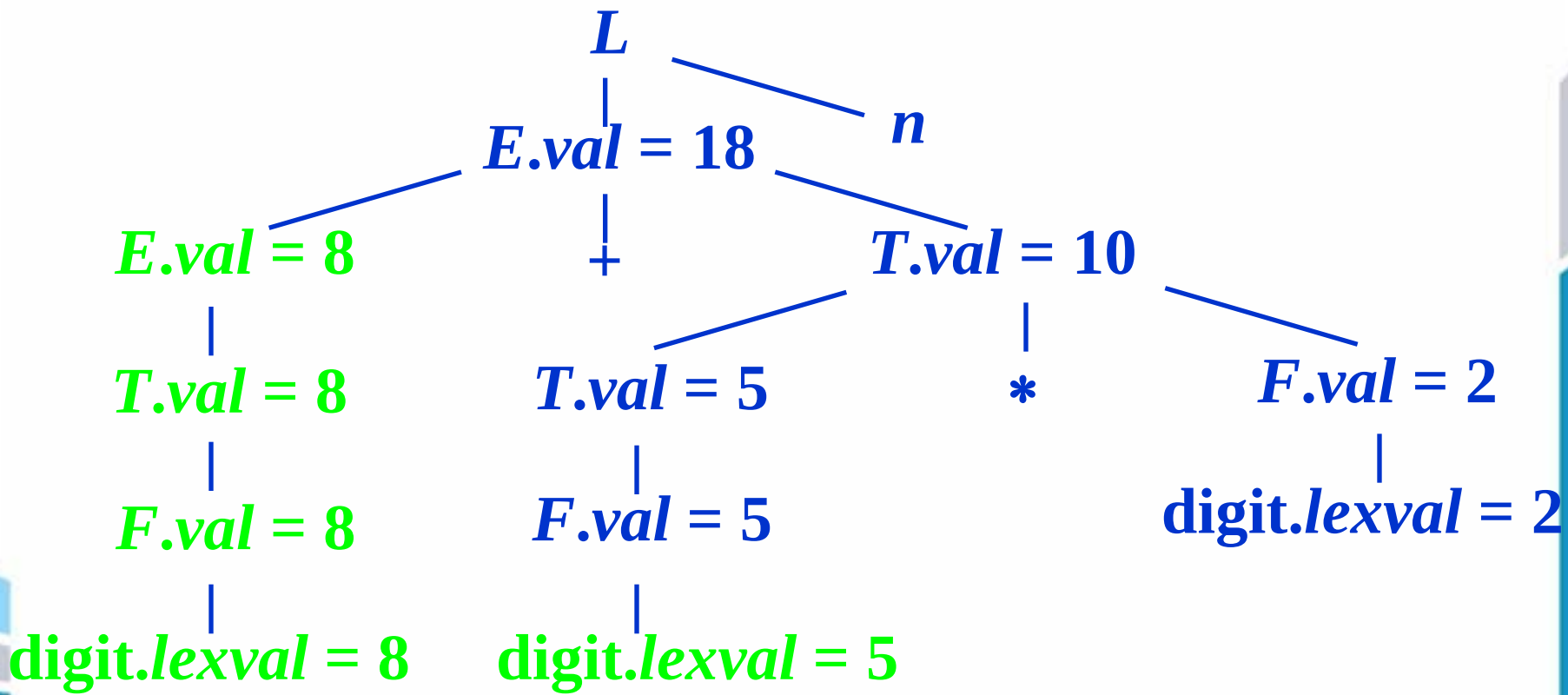
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



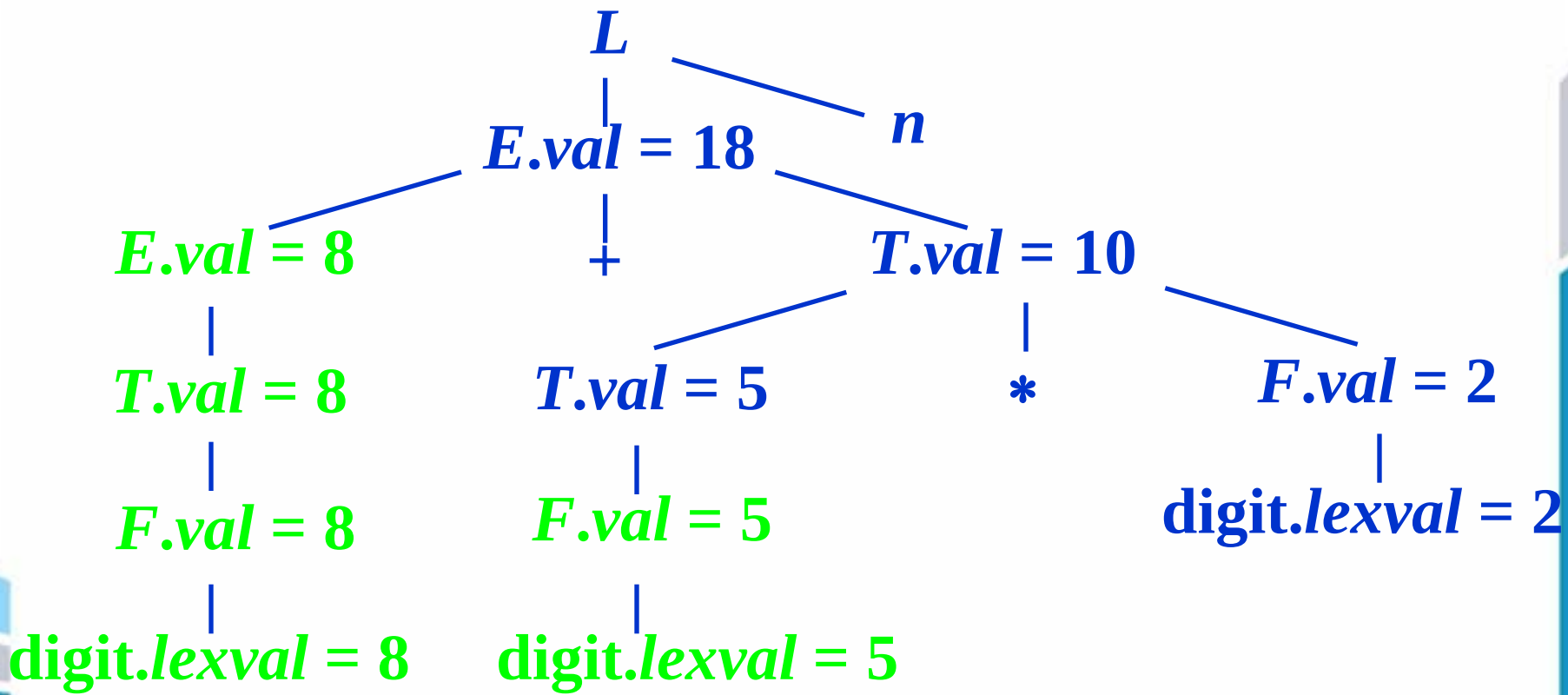
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



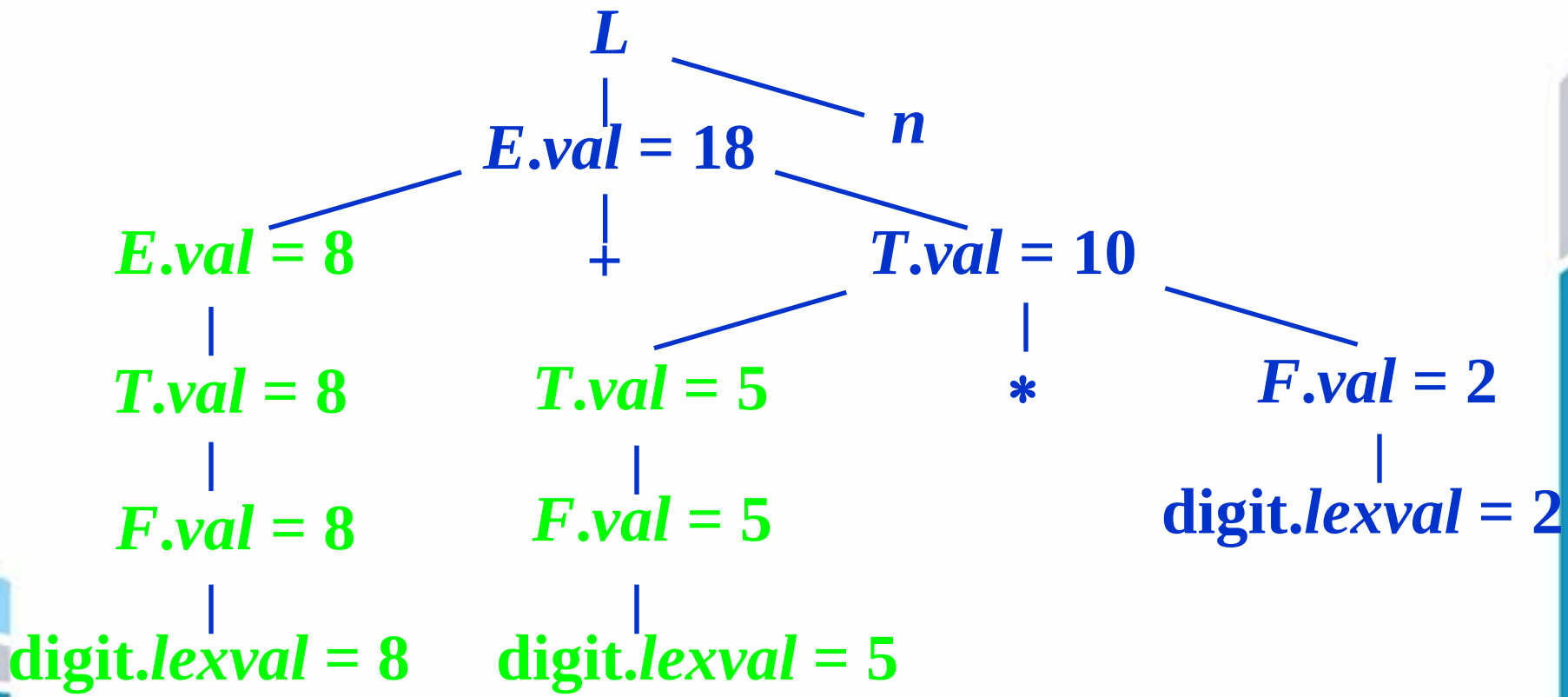
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



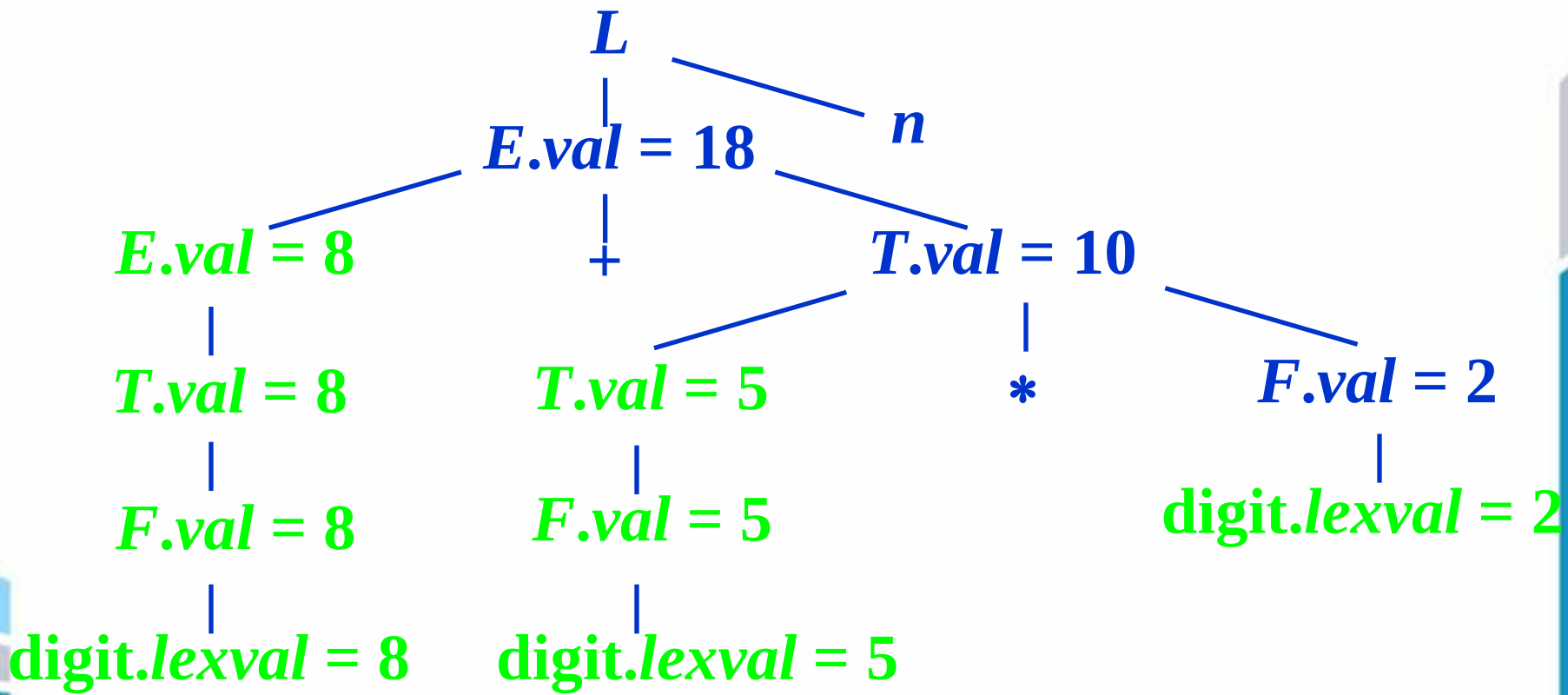
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



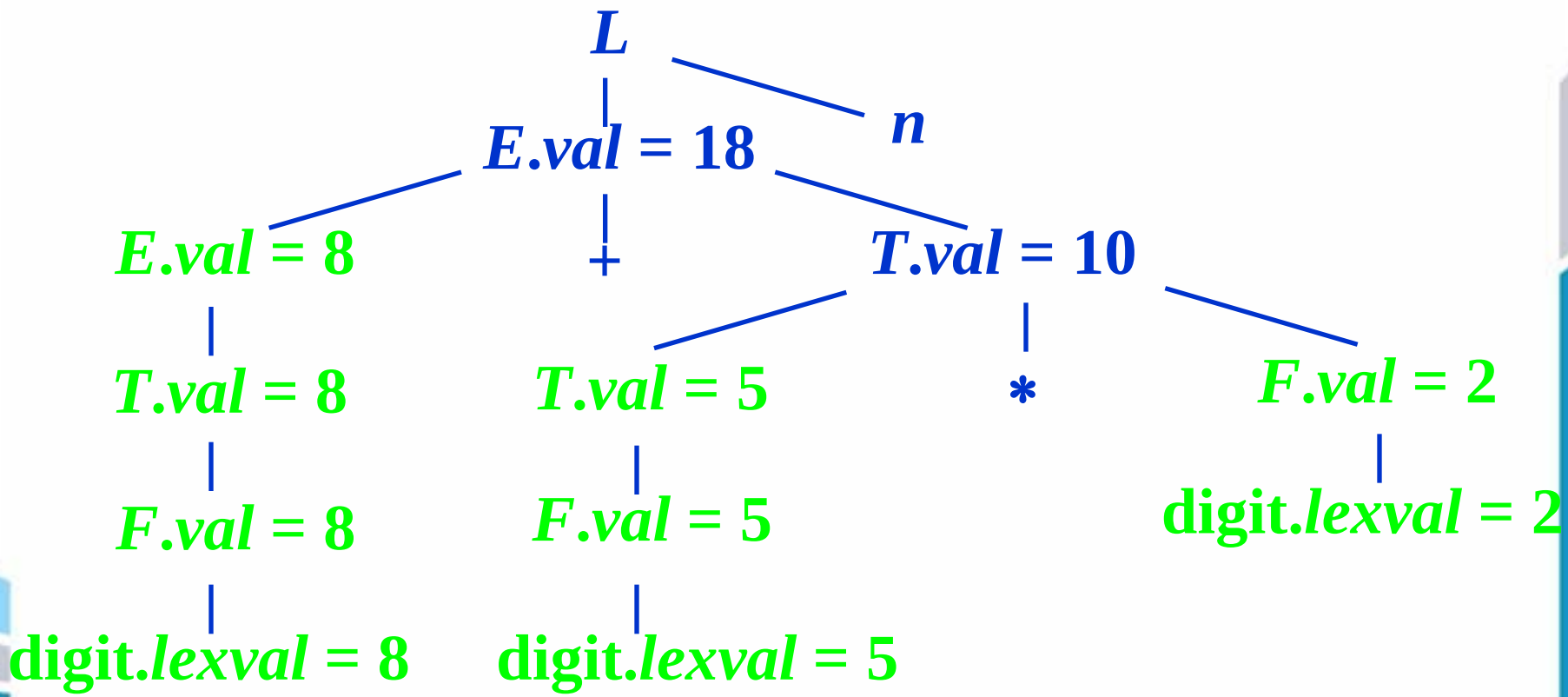
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



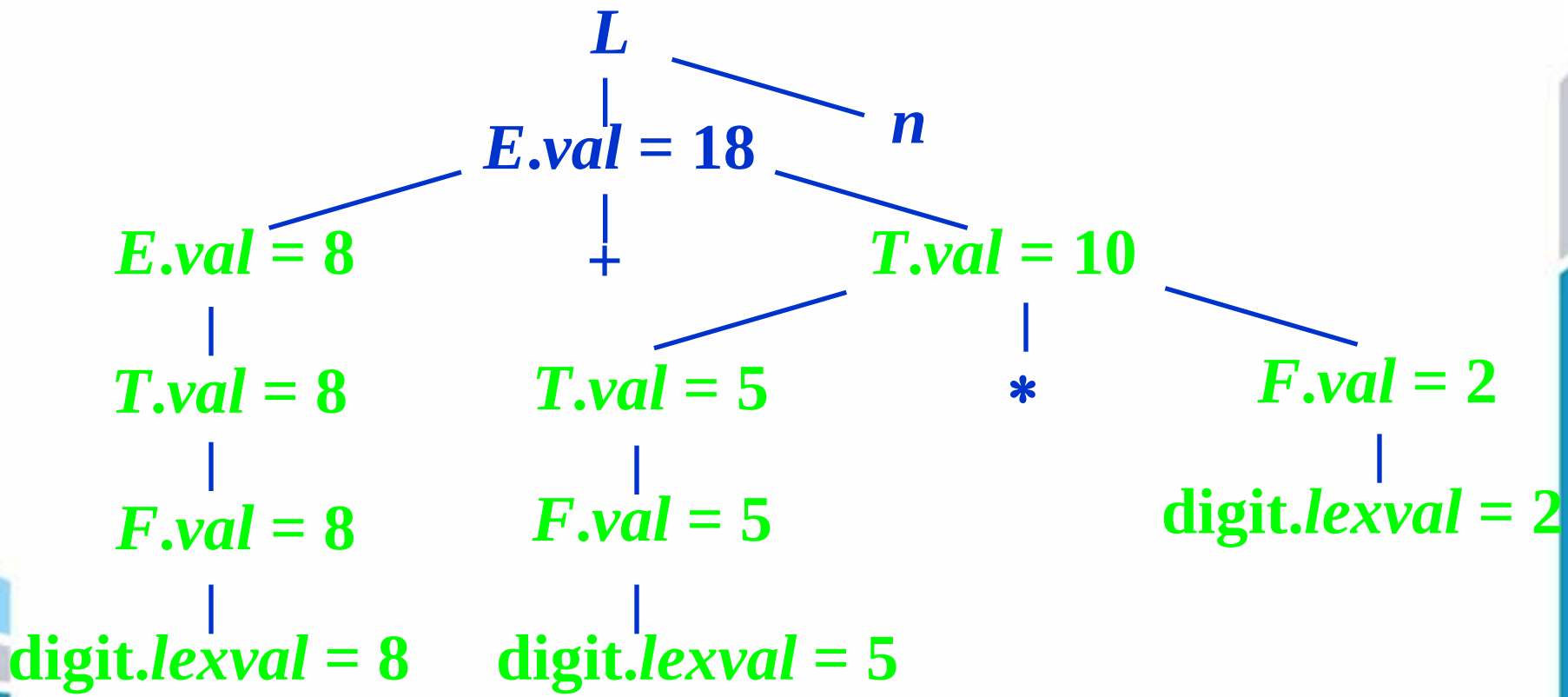
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



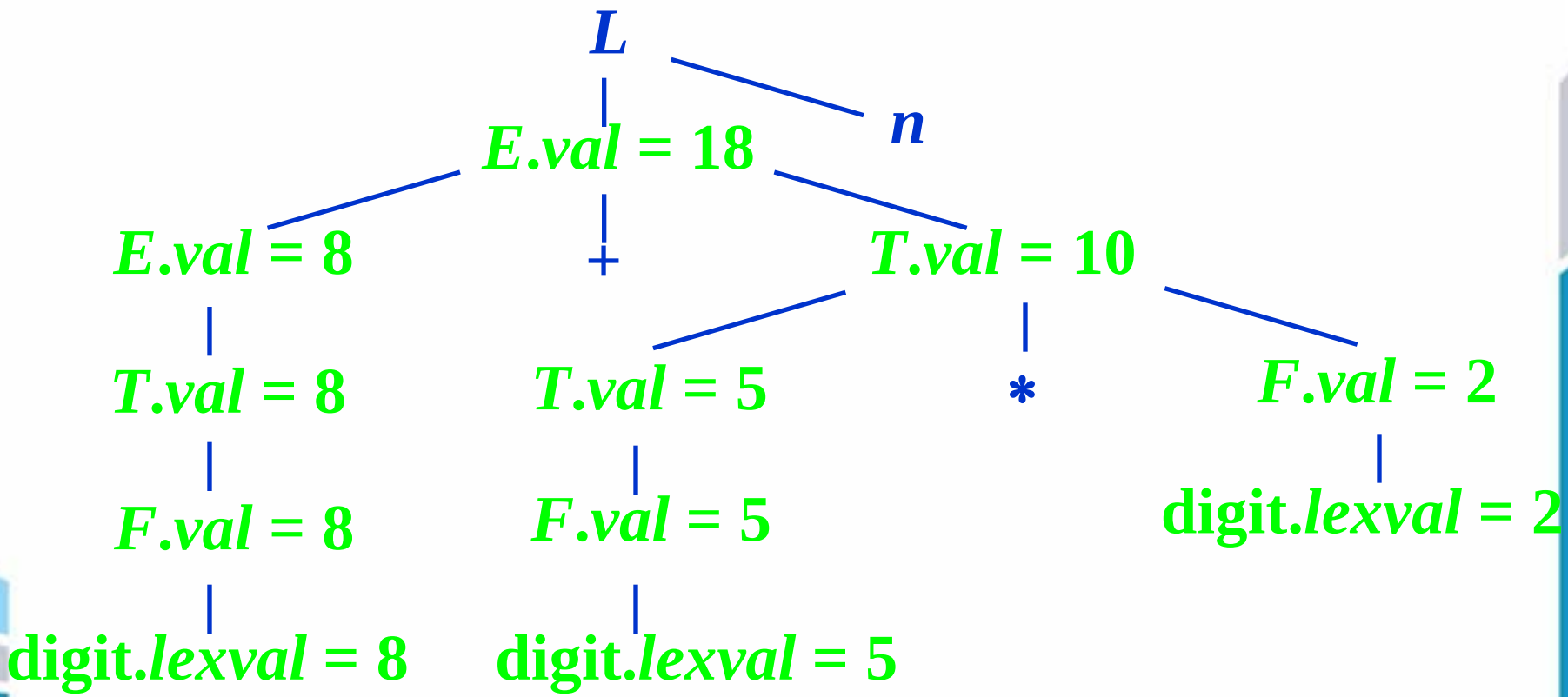
5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



5.1 语法制导的定义

分析树各结点属性的计算可以自下而上地完成



5.1 语法制导的定义

5.1.3 继承属性

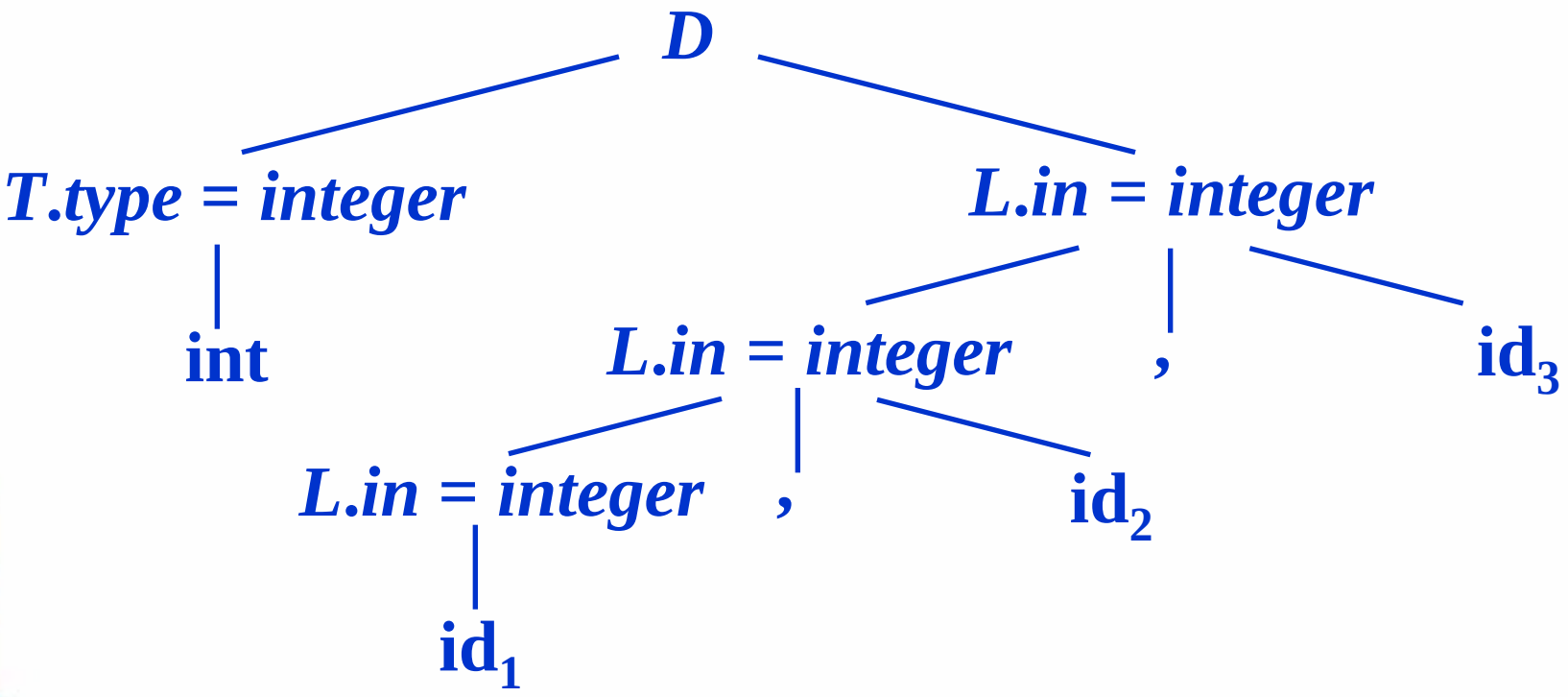
int id, id, id

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	$addType(\text{id}.entry, L.in)$

5.1 语法制导的定义

- 例 int id_1, id_2, id_3的标注了部分属性的分析树

  不可能像像综合属性那样自下而上标注属性

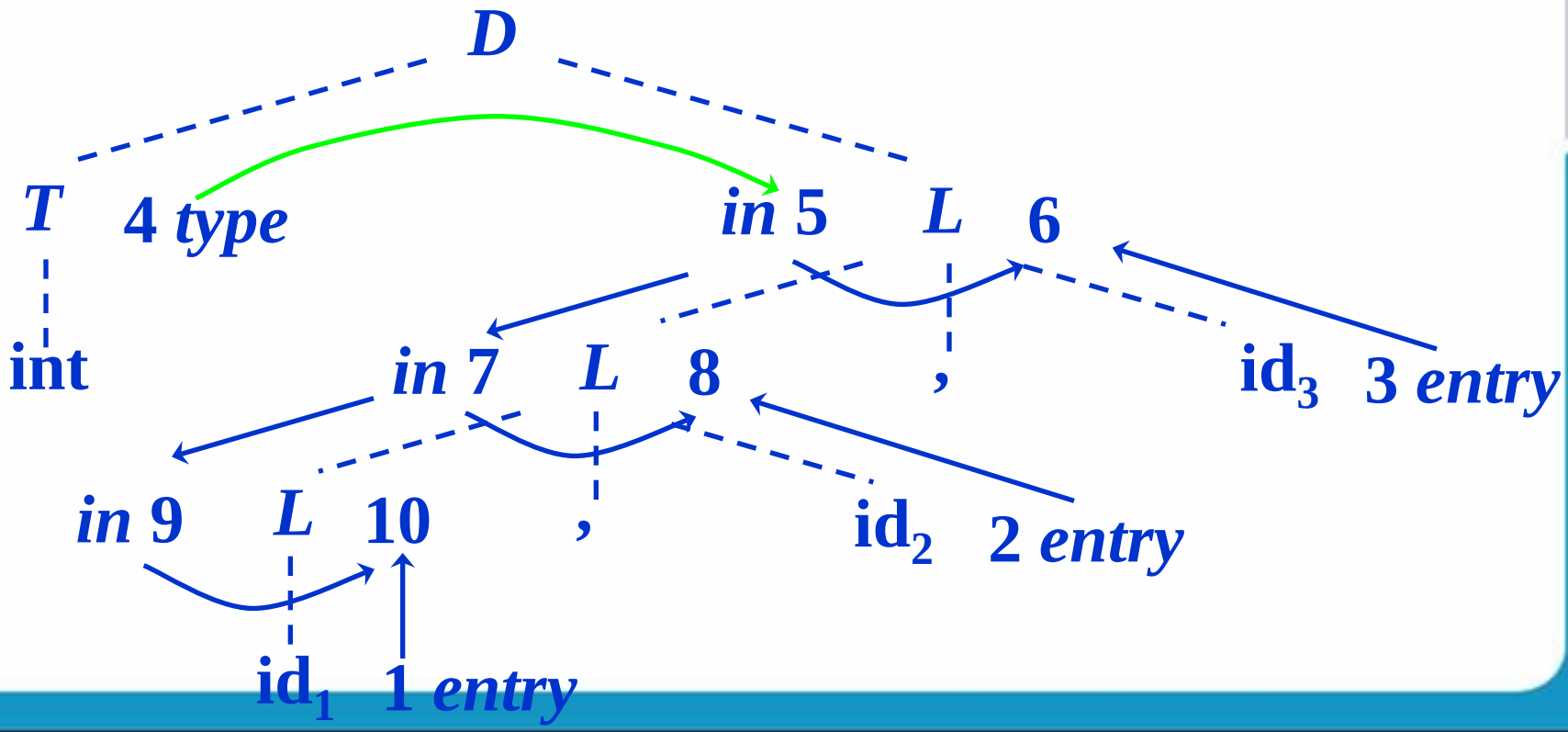


5.1 语法制导的定义

5.1.4 属性依赖图

- 例 int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

$D \rightarrow TL$　　$L.in = T.type$

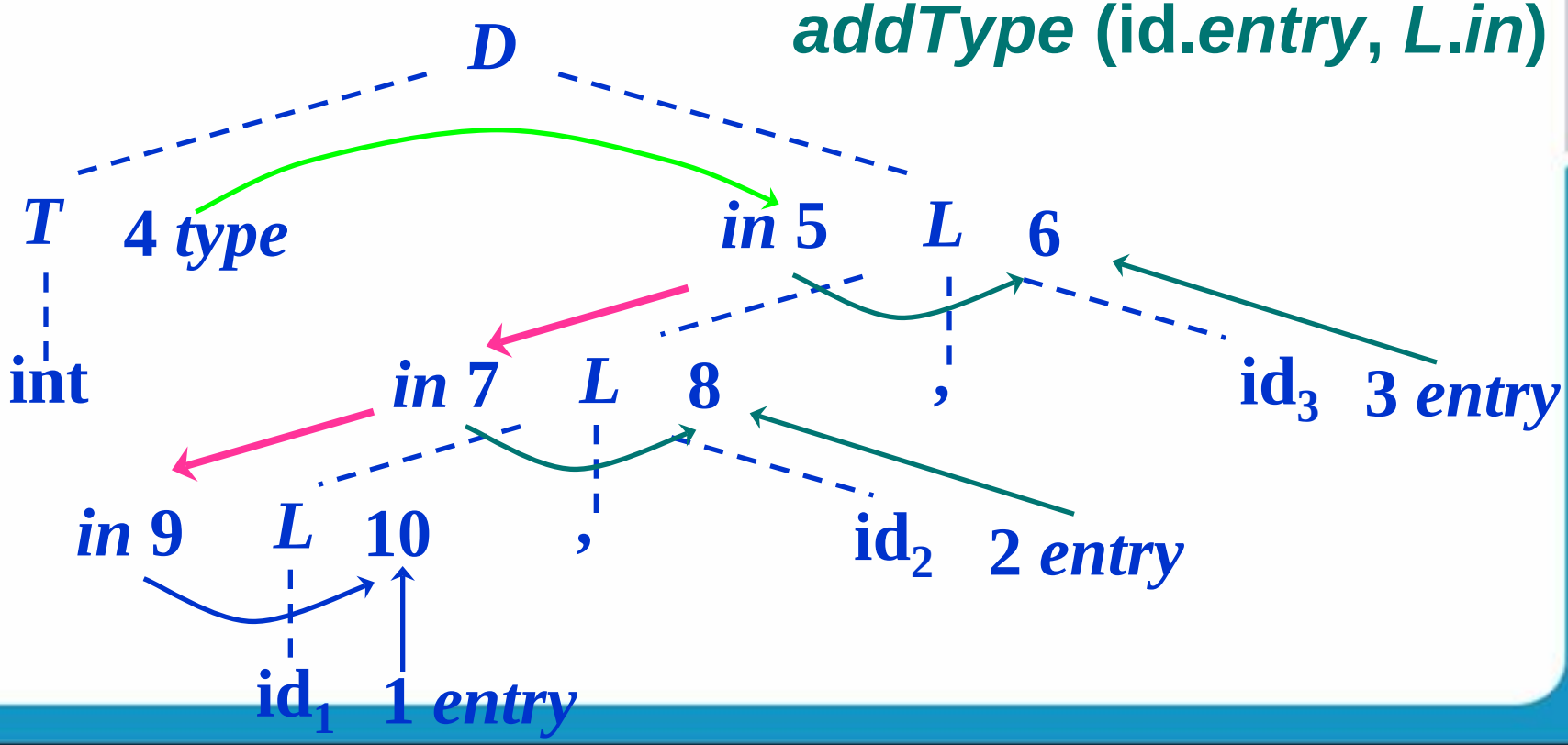


5.1 语法制导的定义

5.1.4 属性依赖图

- 例 int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

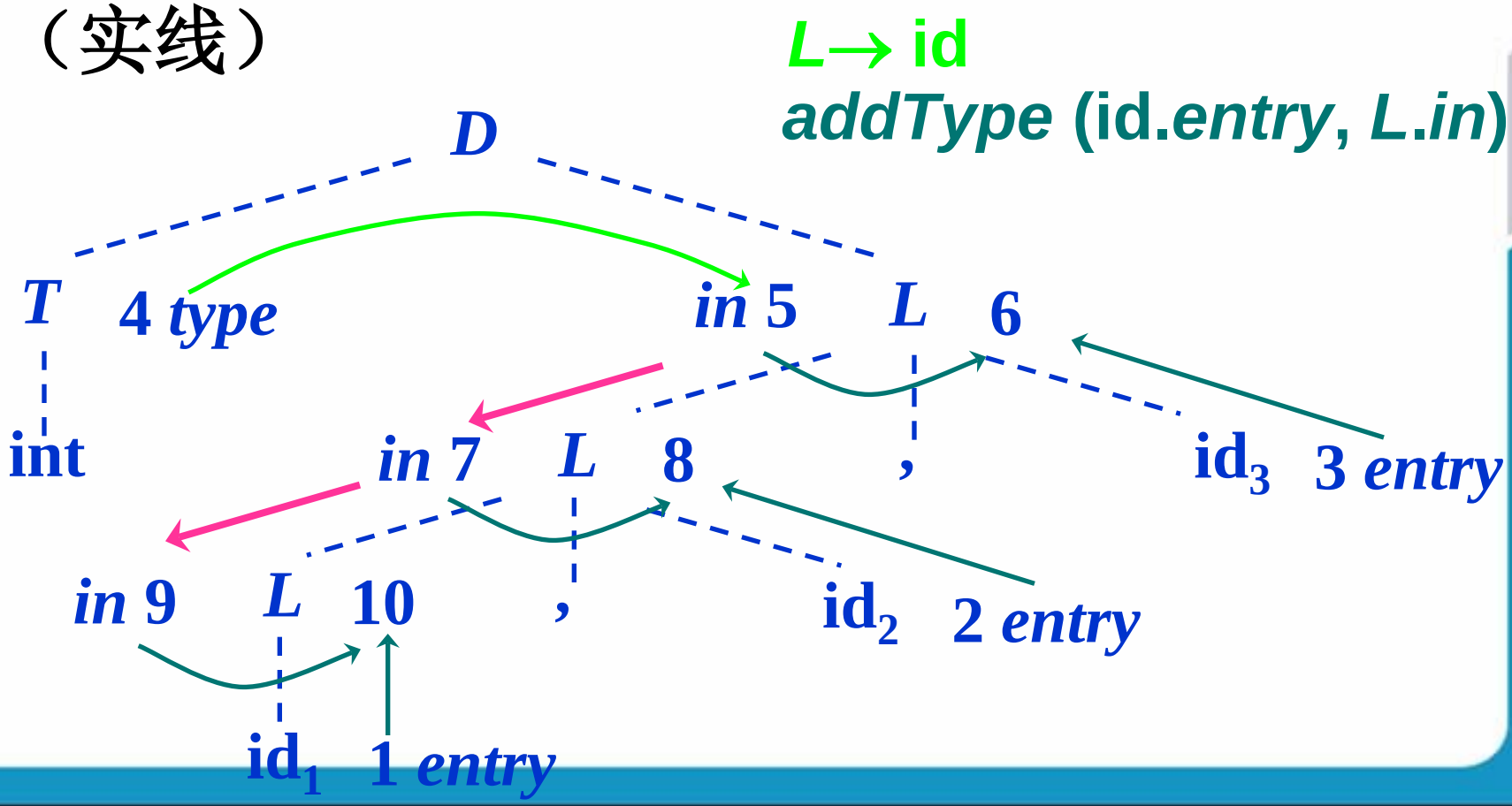
$L \rightarrow L_1$, id　$L_1.in = L.in$;
addType (id.*entry*, *L.in*)



5.1 语法制导的定义

5.1.4 属性依赖图

- 例 int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

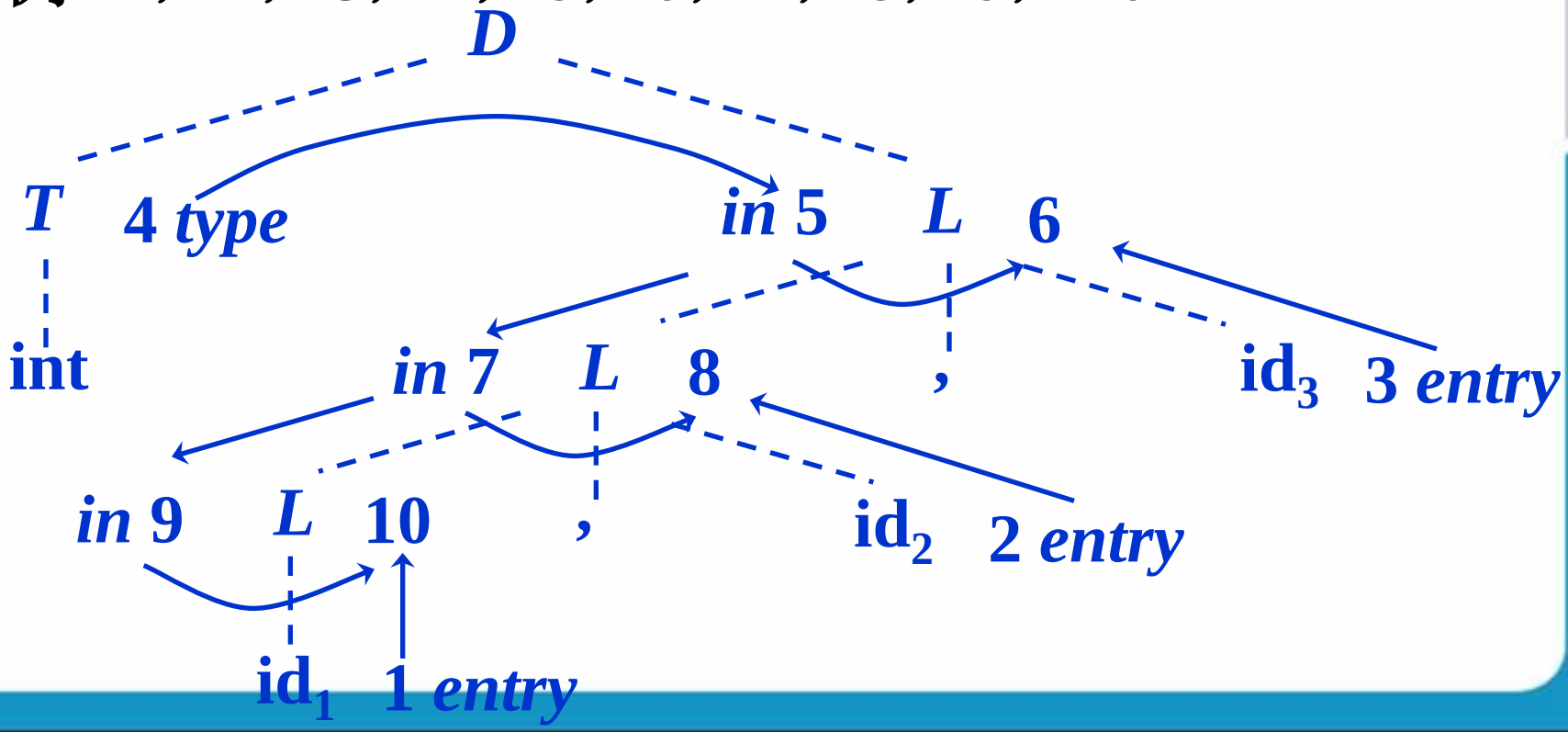


5.1 语法制导的定义

5.1.5 属性计算次序

1、拓扑排序：结点的一种排序，使得边只会从该次序中先出现的结点到后出现的结点

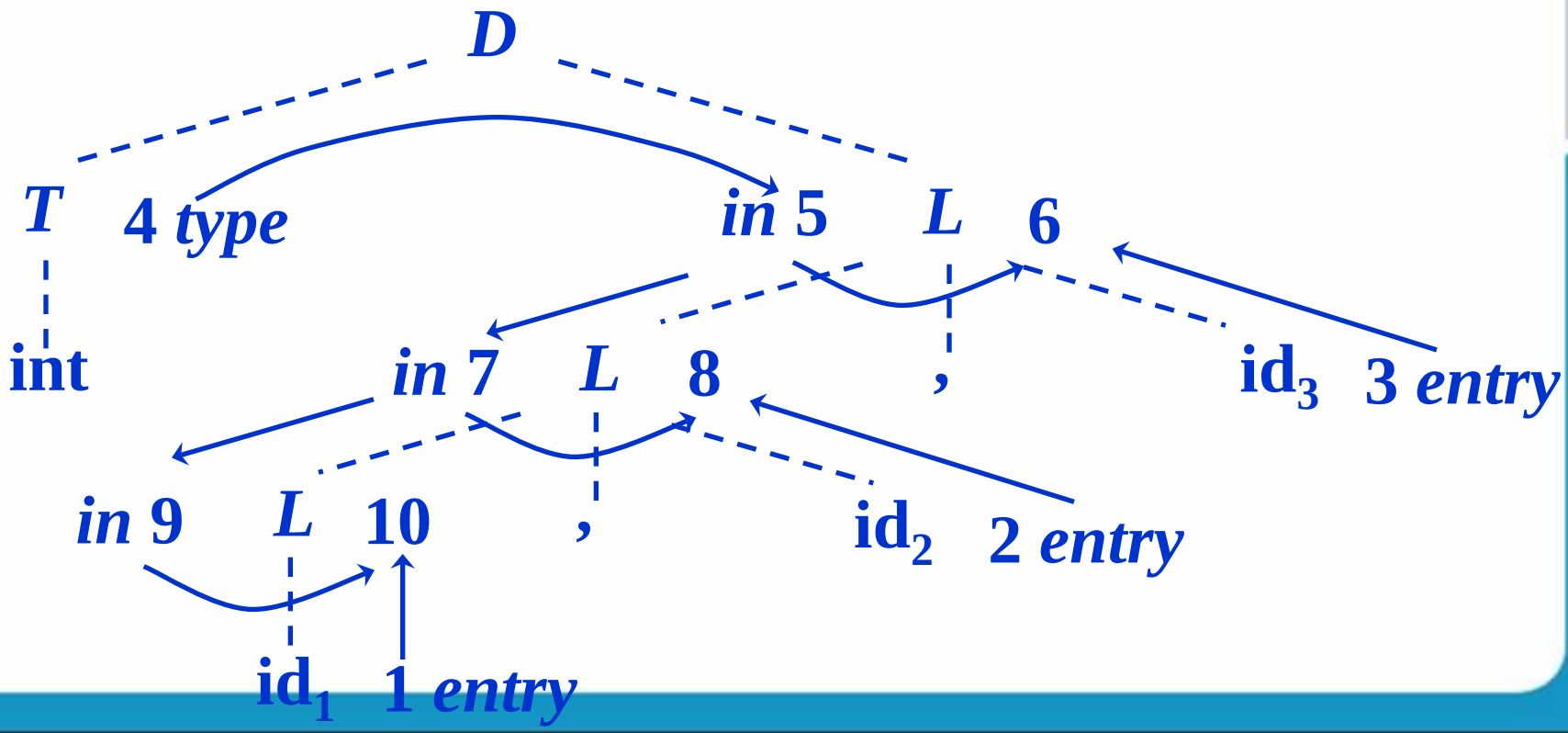
- 例 1，2，3，4，5，6，7，8，9，10



5.1 语法制导的定义

5.1.5 属性计算次序

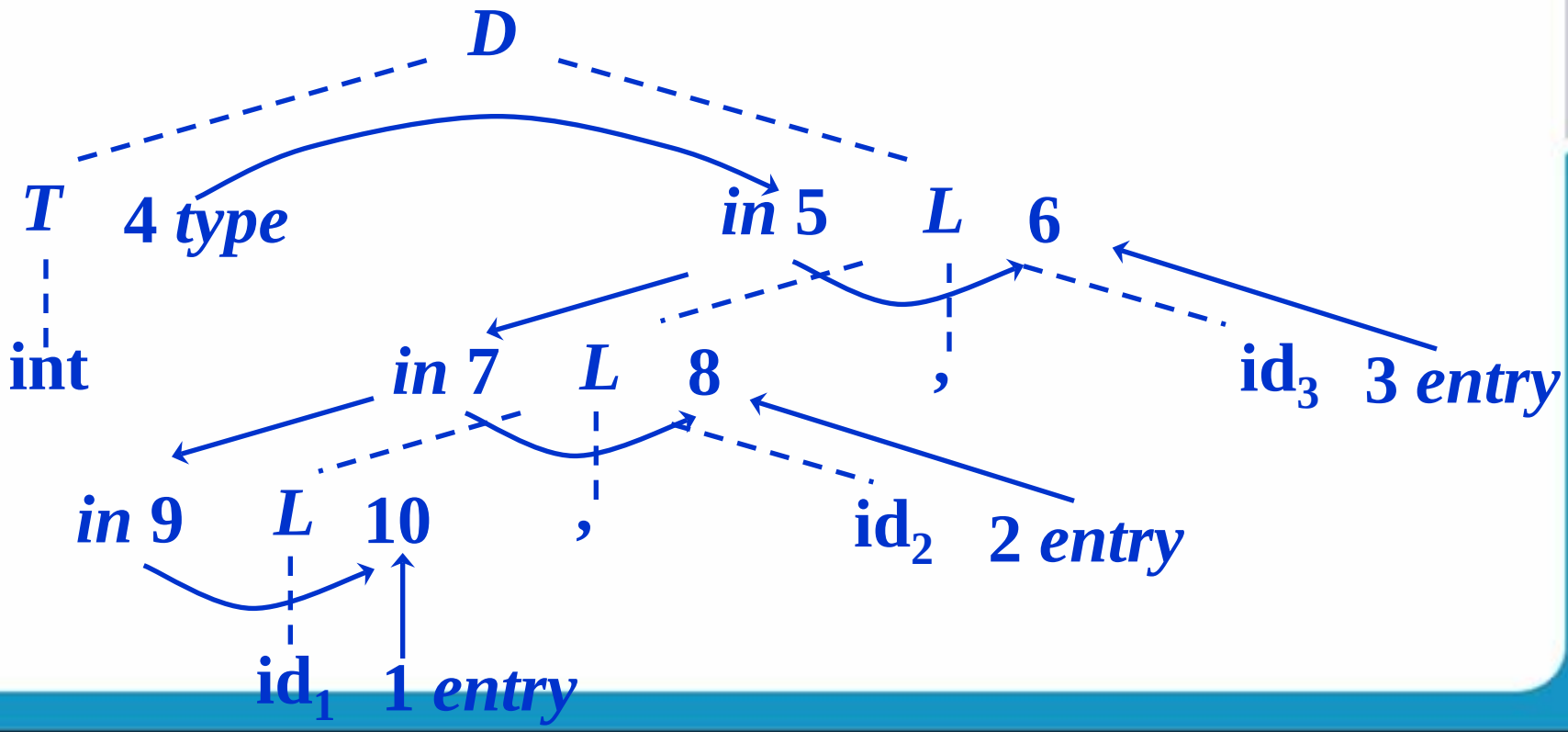
2、属性计算次序：构造输入的分析树



5.1 语法制导的定义

5.1.5 属性计算次序

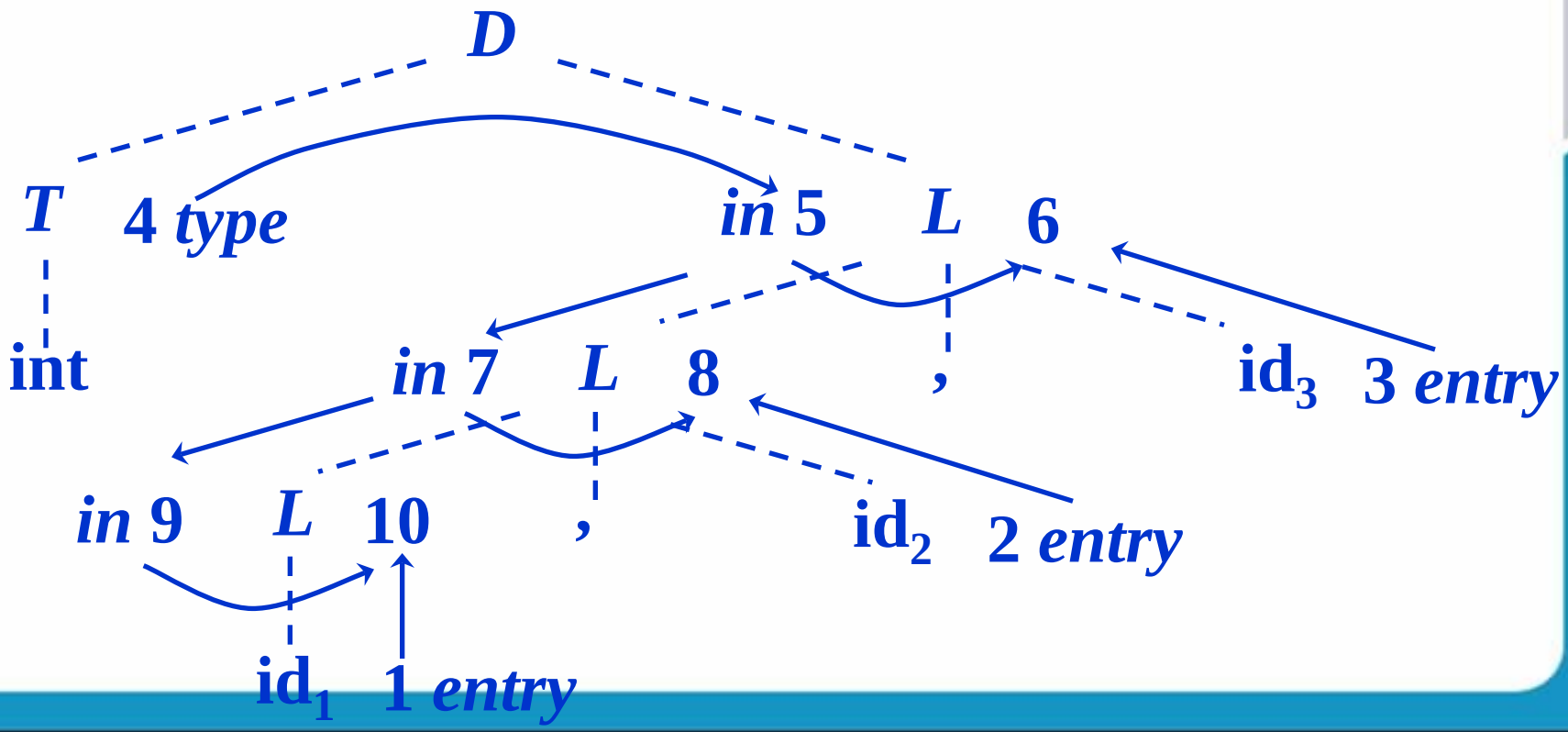
2、属性计算次序：构造输入的分析树，构造属性依赖图



5.1 语法制导的定义

5.1.5 属性计算次序

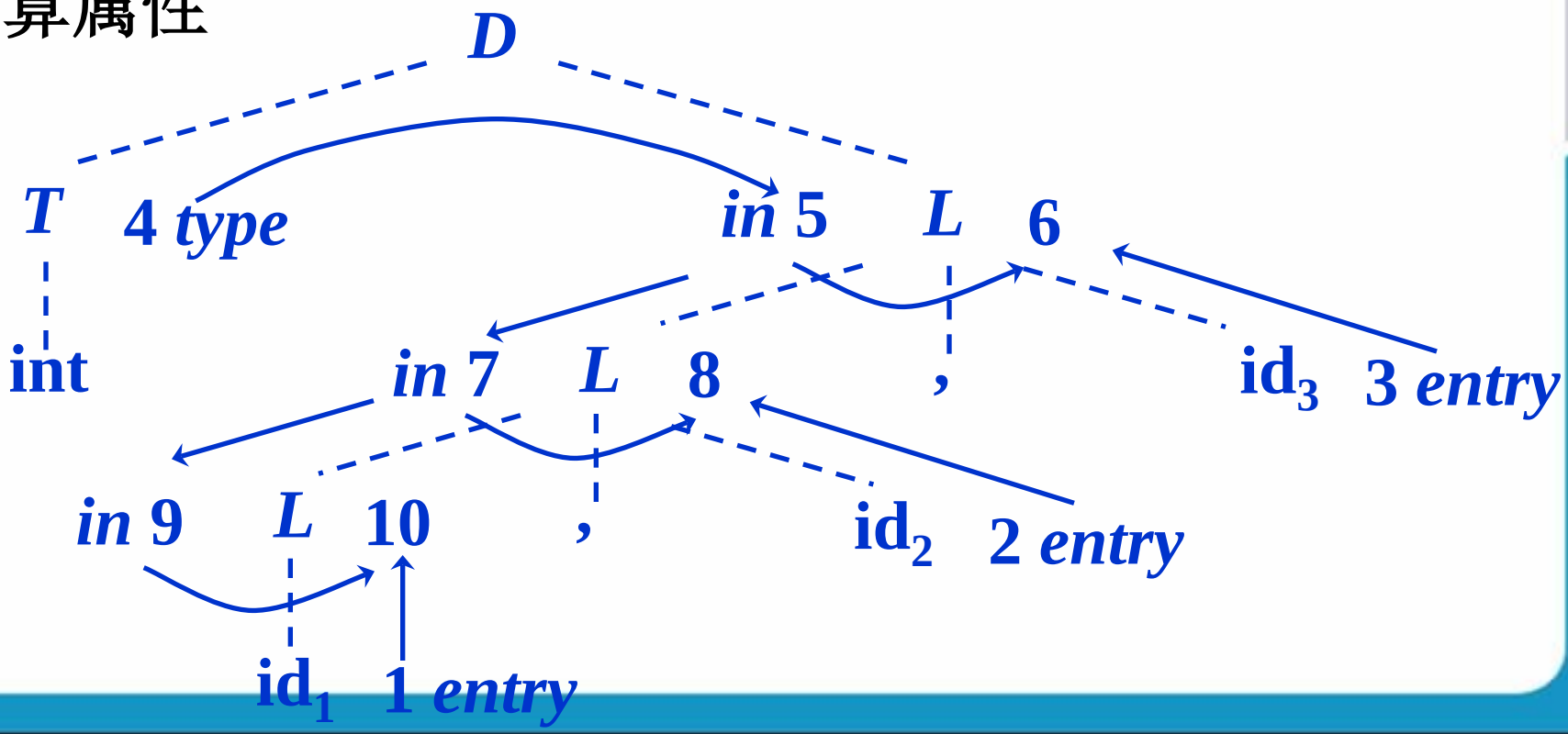
2、属性计算次序：构造输入的分析树，构造属性依赖图，对结点进行拓扑排序



5.1 语法制导的定义

5.1.5 属性计算次序

2、属性计算次序：构造输入的分析树，构造属性依赖图，对结点进行拓扑排序，按拓扑排序的次序计算属性



5.1 语法制导的定义

语义规则的计算方法

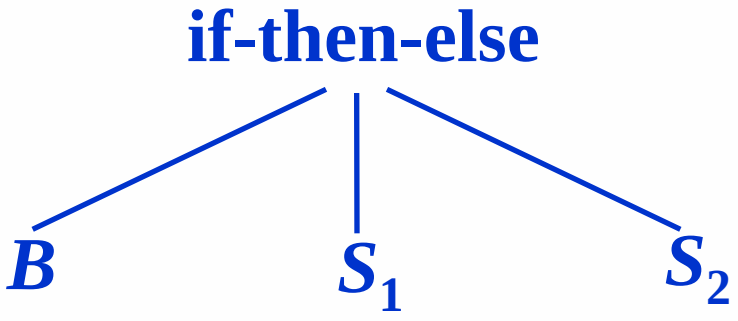
- 分析树方法：刚才介绍的方法，动态确定计算次序，效率低　　概念上的一般方法
- 基于规则的方法：（编译器实现者）静态确定（编译器设计者提供的）语义规则的计算次序　　适用于手工构造的方法
- 忽略规则的方法：（编译器实现者）事先确定属性的计算策略（如边分析边计算），（编译器设计者提供的）语义规则必须符合所选分析方法的限制　　适用于自动生成的方法

5.2 S属性定义的自下而上计算

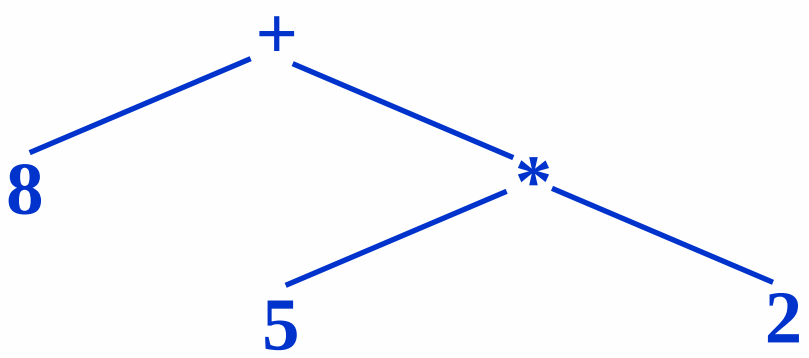
5.2.1 语法树

- 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
- 语法树的例子：

if B then S_1 else S_2



8 + 5 * 2



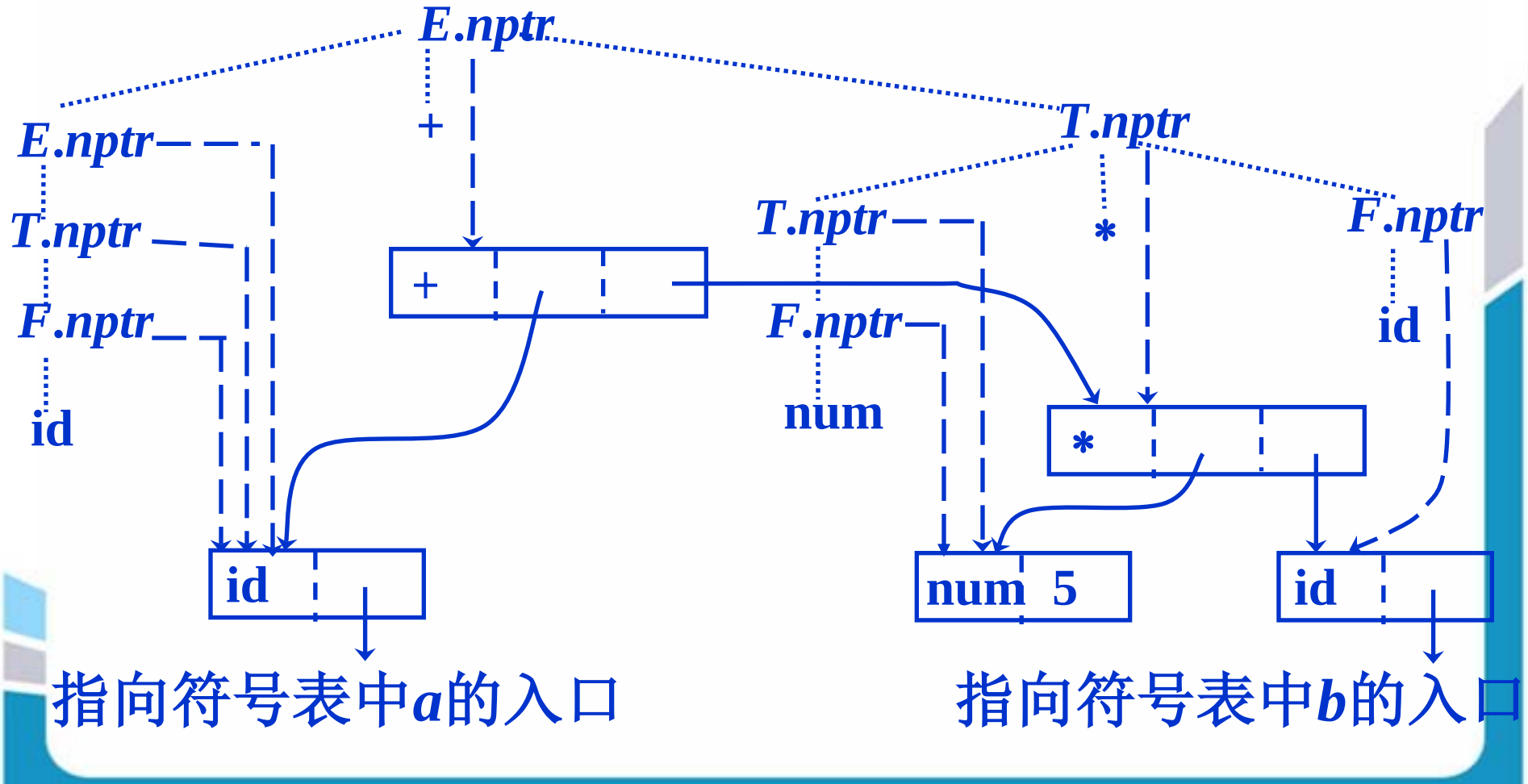
5.2 S属性定义的自下而上计算

5.2.2 构造语法树的语法制导定义

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

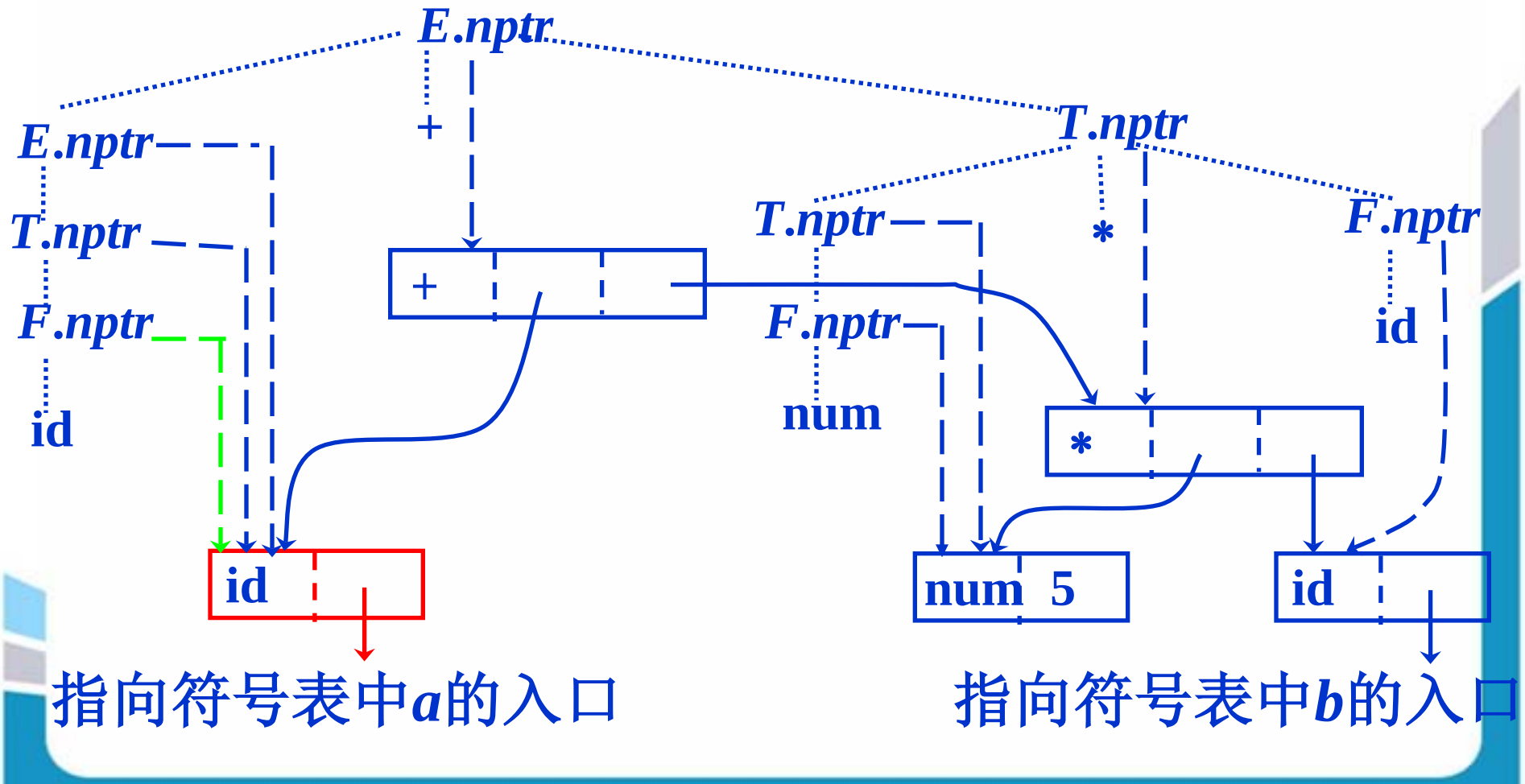
5.2 S属性定义的自下而上计算

a+5**b*的语法树的构造



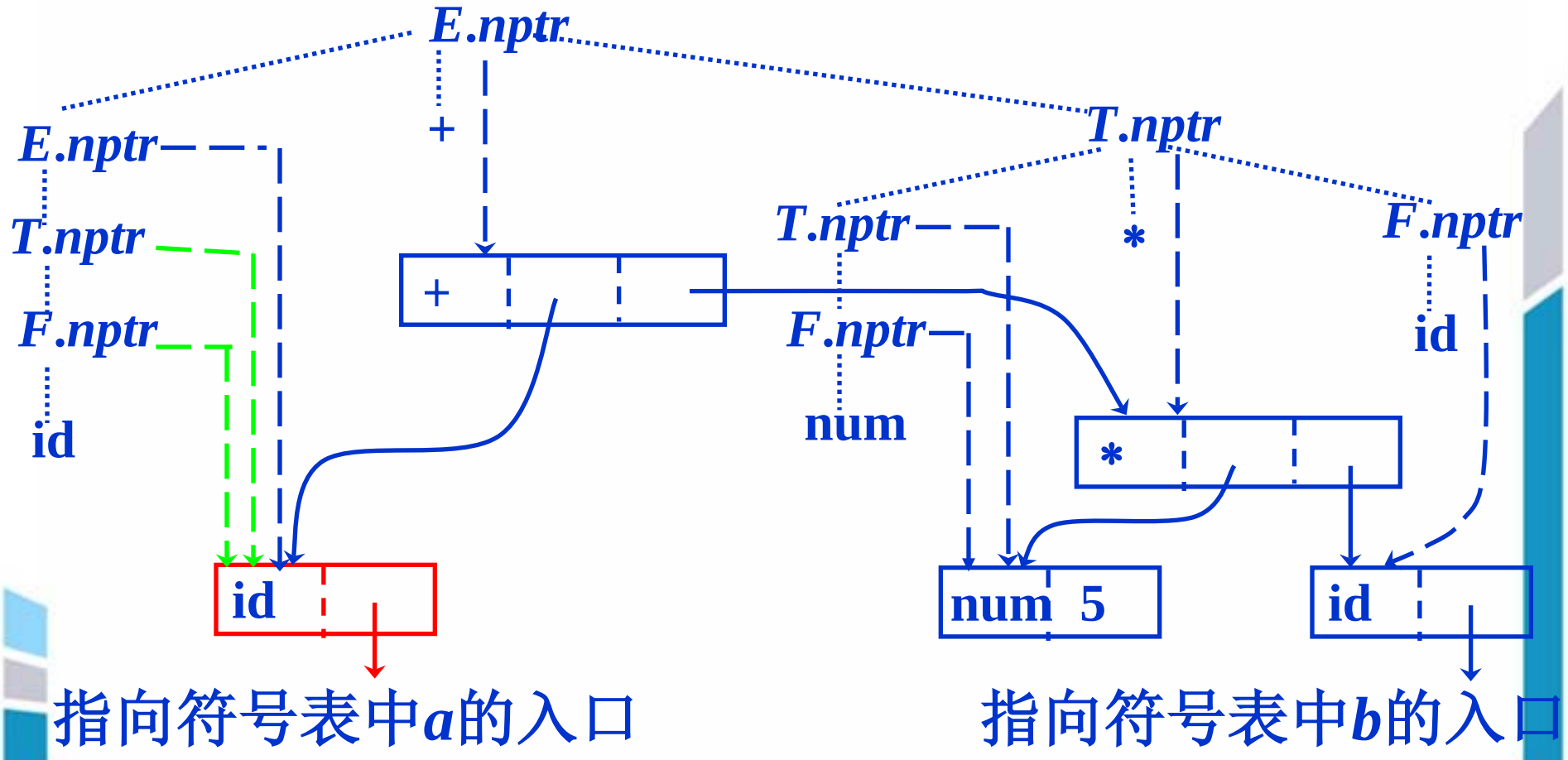
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



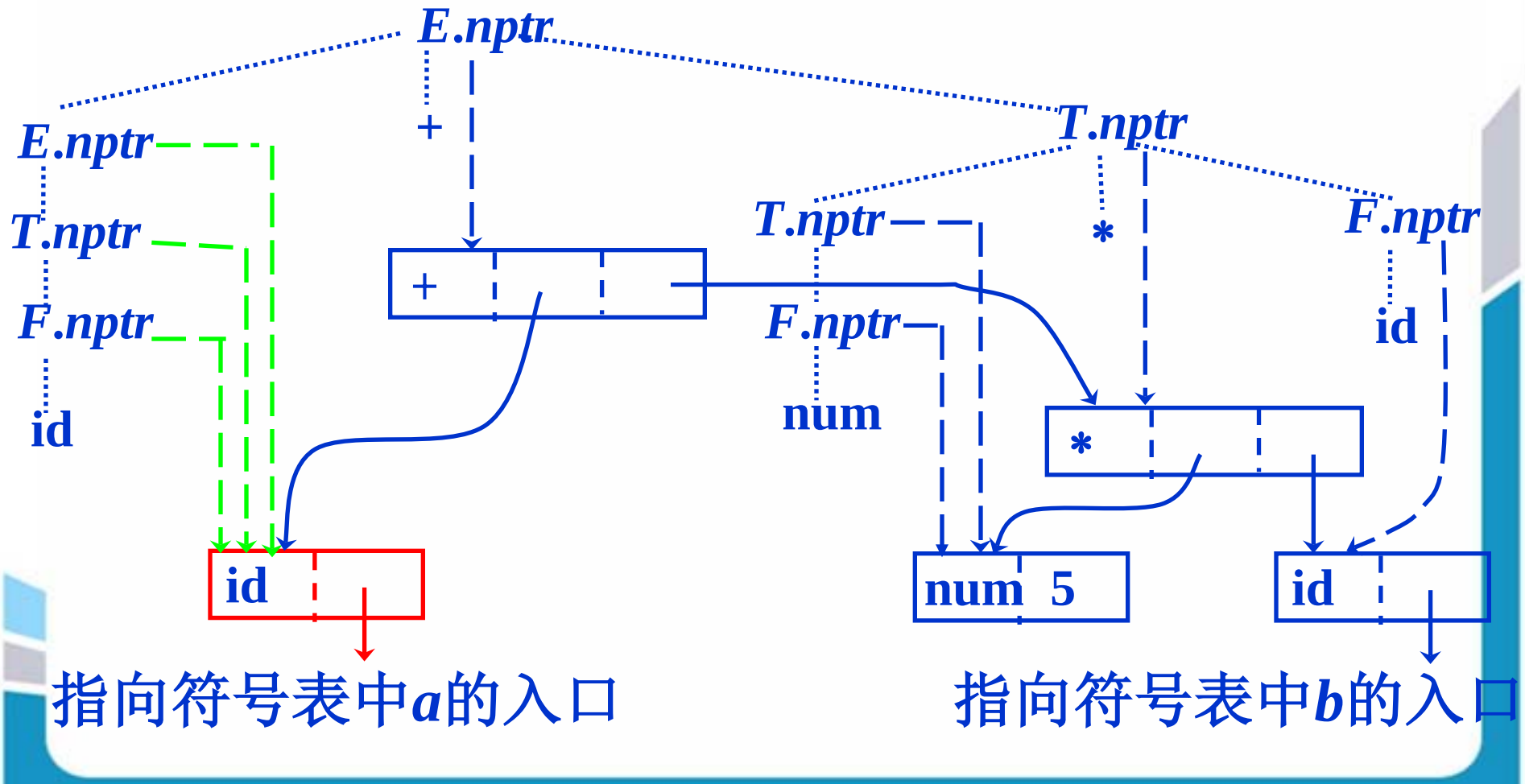
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



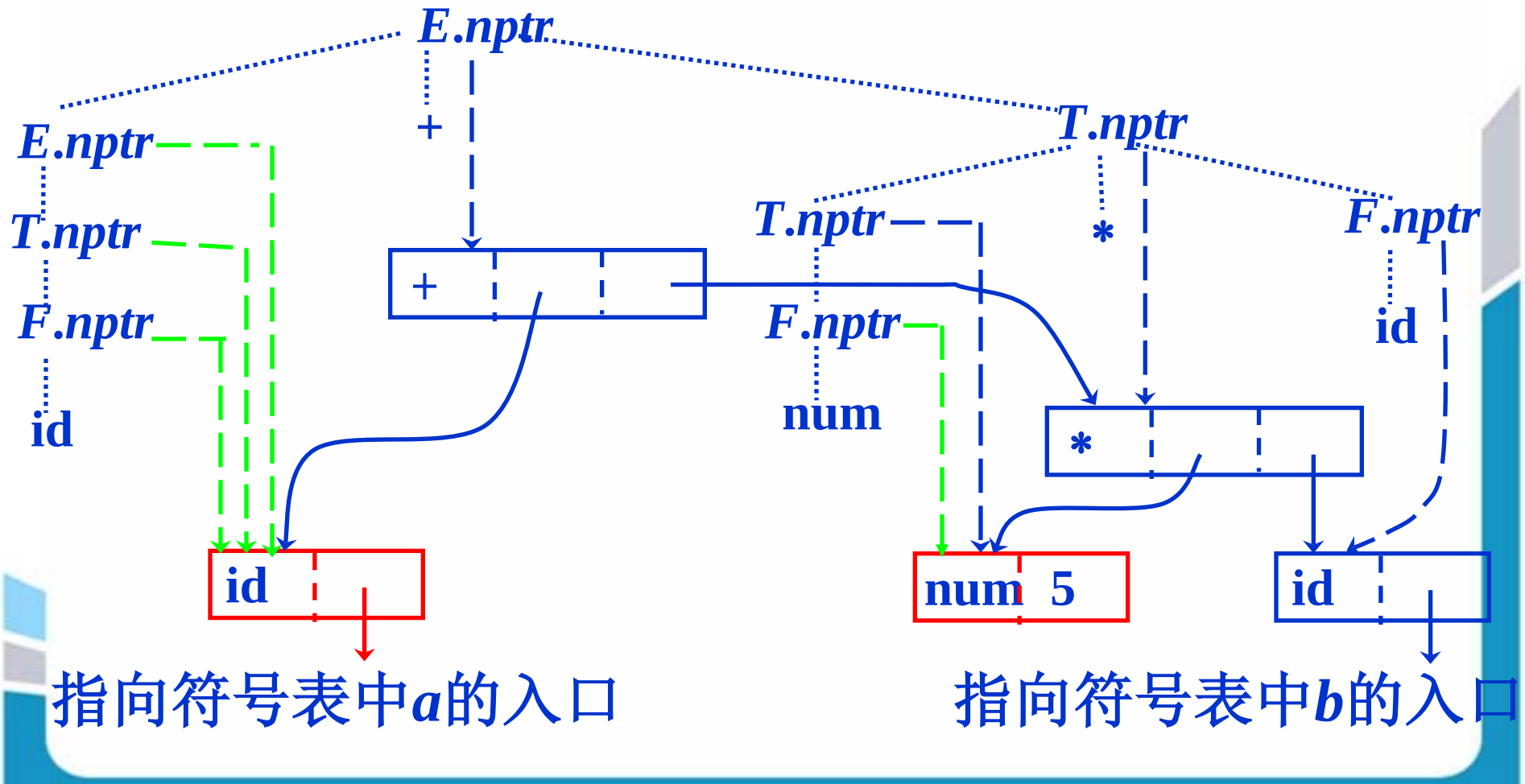
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



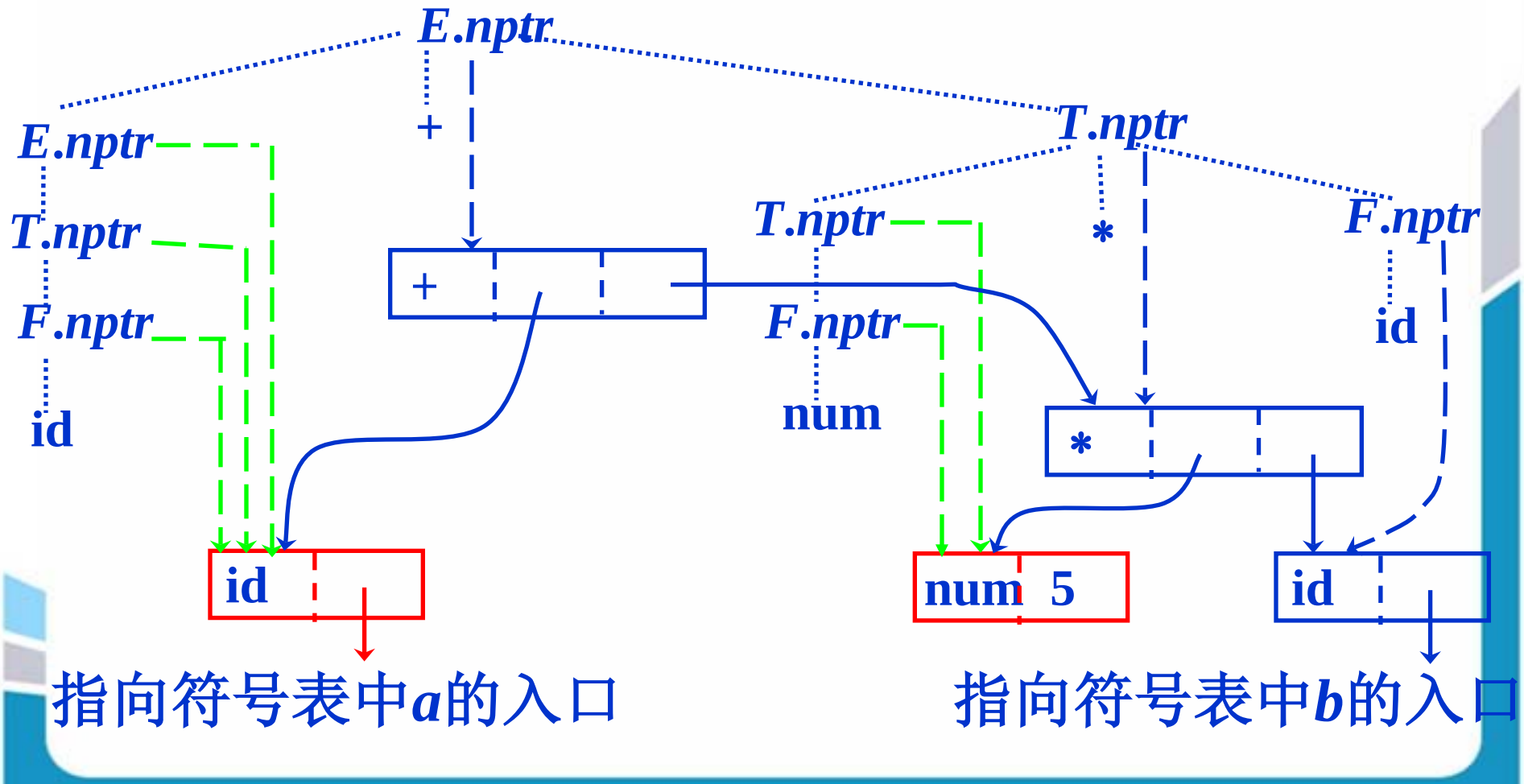
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



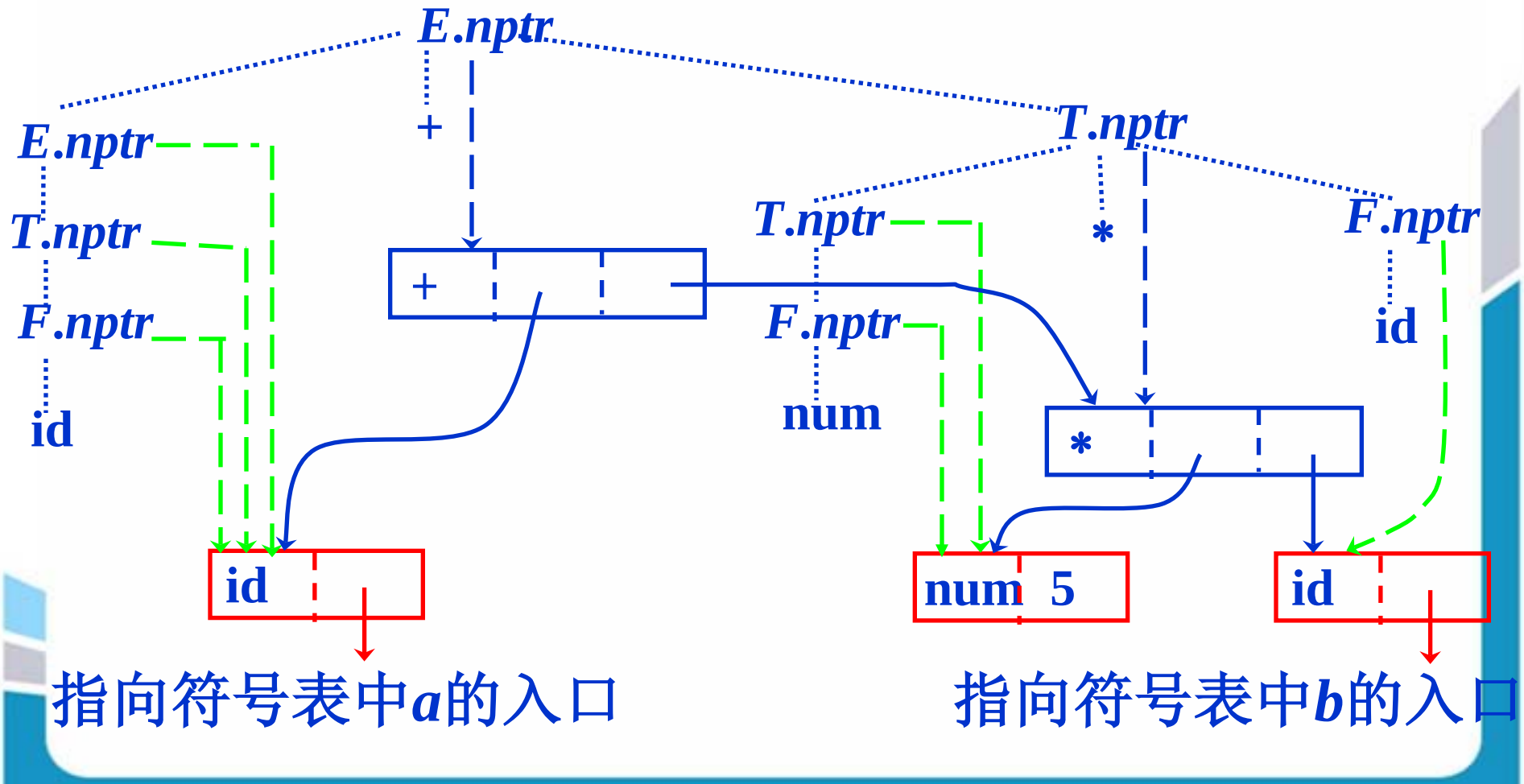
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



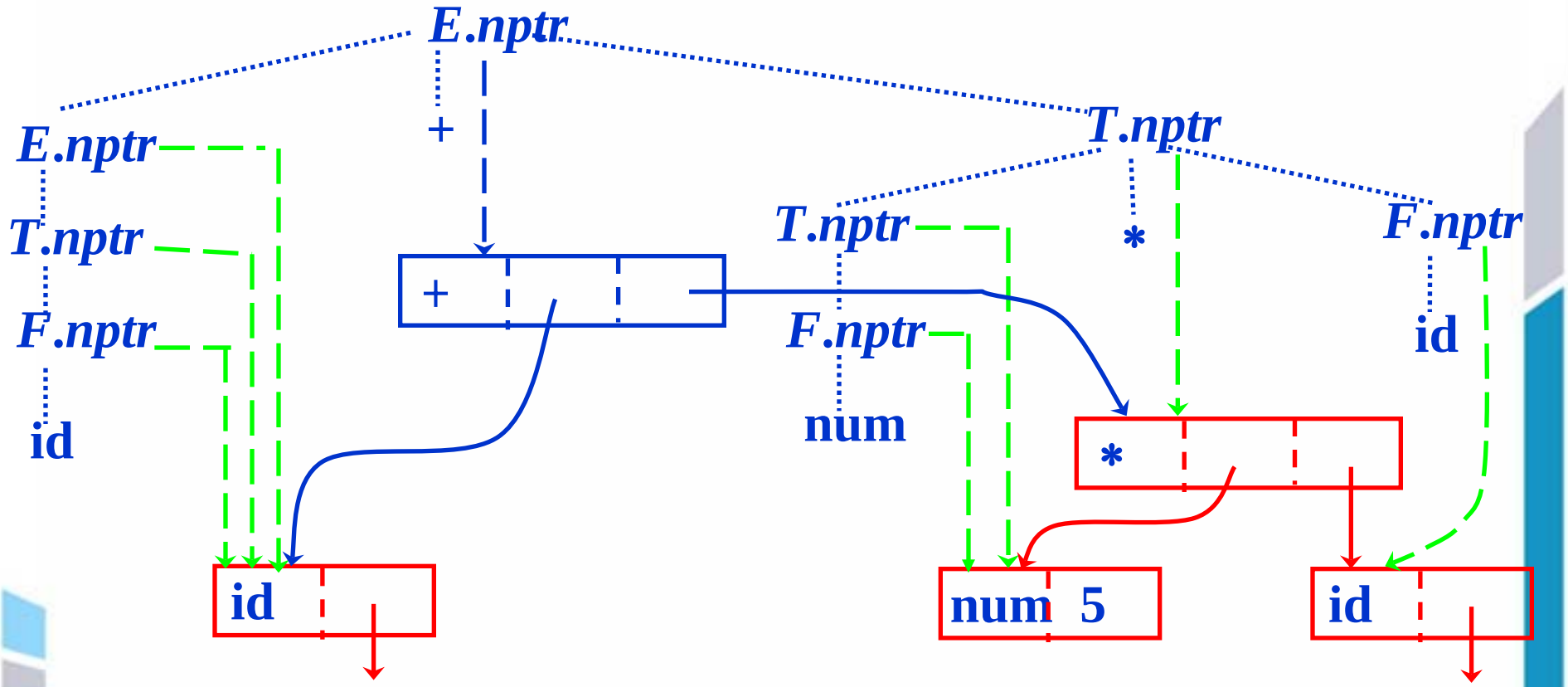
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



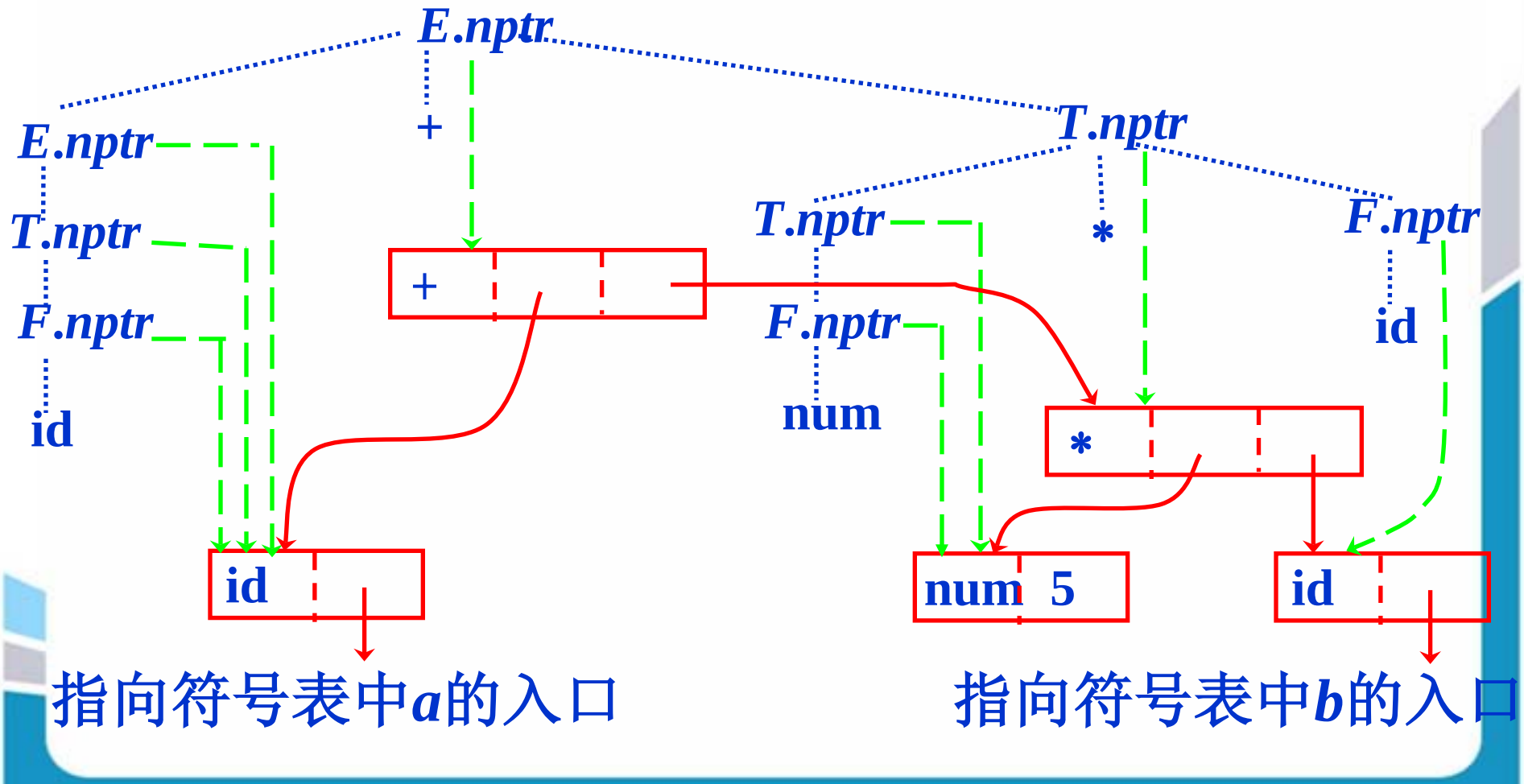
5.2 S属性定义的自下而上计算

a+5*b的语法树的构造



5.2 S属性定义的自下而上计算

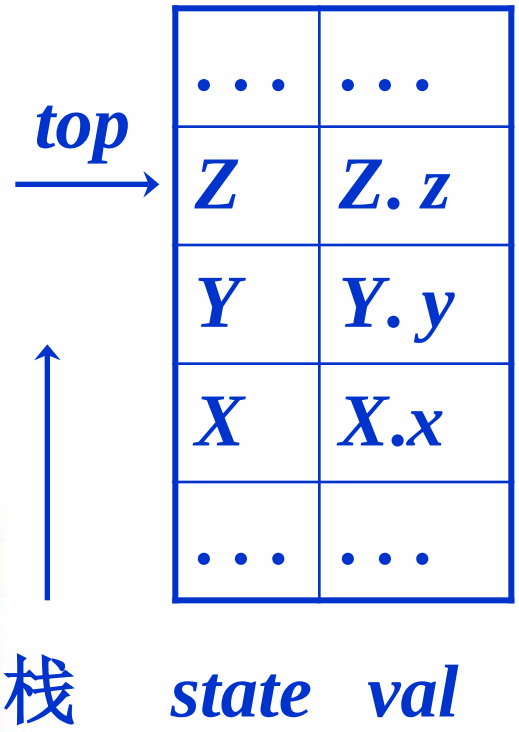
a+5*b的语法树的构造



5.2 S属性定义的自下而上计算

5.2.3 S属性的自下而上计算

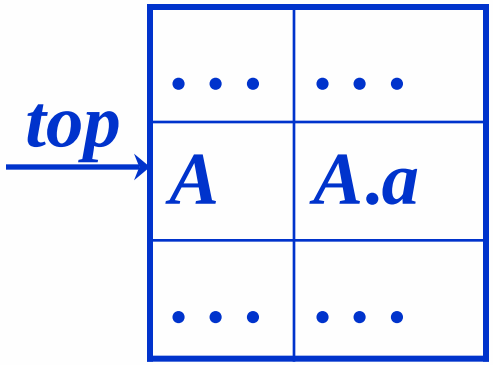
LR分析器的栈增加一个域来保存综合属性值



若产生式$A \rightarrow XYZ$的语义规则是

$A.a = f(X.x, Y.y, Z.z)$，

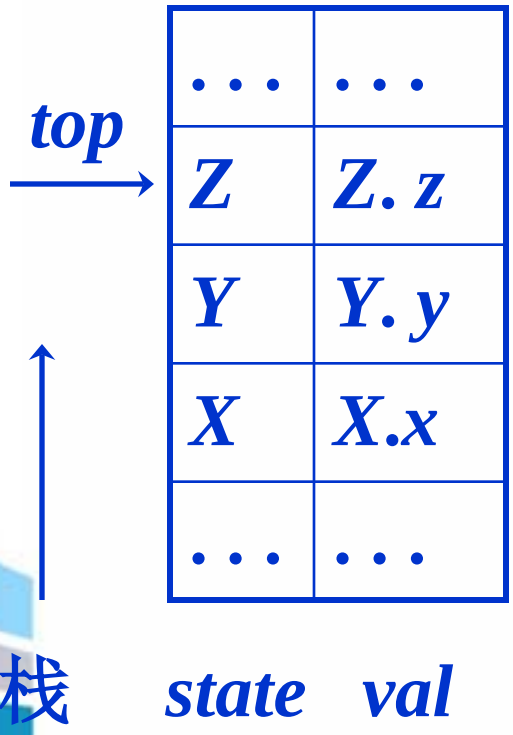
那么归约后：



5.2 S属性定义的自下而上计算

简单计算器的语法制导定义改成栈操作代码

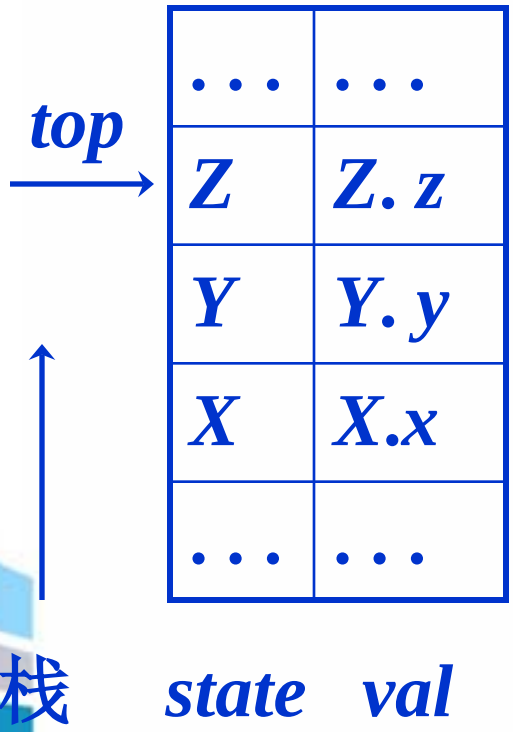
产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*



5.2 S属性定义的自下而上计算

简单计算器的语法制导定义改成栈操作代码

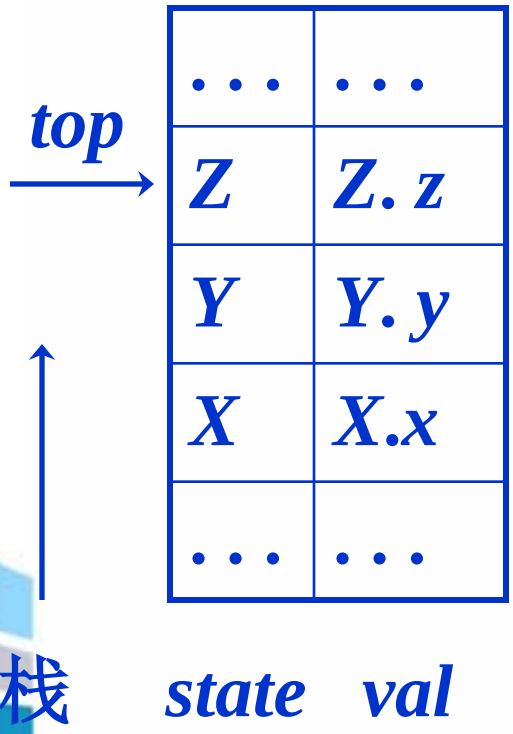
产 生 式	代 码 段
$L \to E$ n	*print* (*E.val*)
$E \to E_1 + T$	*E.val* = E_1*.val* + *T.val*
$E \to T$	*E.val* = *T.val*
$T \to T_1 * F$	*T.val* = T_1*.val* * *F.val*
$T \to F$	*T.val* = *F.val*
$F \to (E)$	*F.val* = *E.val*
$F \to$ digit	*F.val* = digit.*lexval*



5.2 S属性定义的自下而上计算

简单计算器的语法制导定义改成栈操作代码

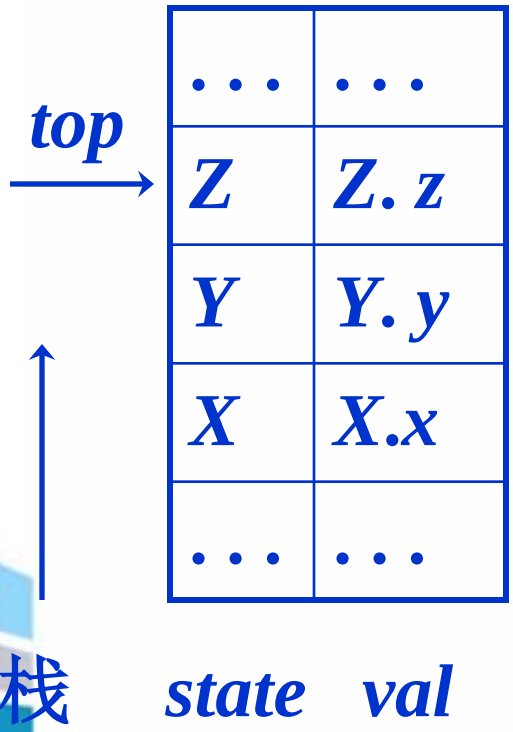
产 生 式	代 码 段
$L \to E$ n	*print* (*val* [*top*–1])
$E \to E_1 + T$	$E.val = E_1.val + T.val$
$E \to T$	$E.val = T.val$
$T \to T_1 * F$	$T.val = T_1.val * F.val$
$T \to F$	$T.val = F.val$
$F \to (E)$	$F.val = E.val$
$F \to$ digit	$F.val =$ digit.*lexval*



5.2 S属性定义的自下而上计算

简单计算器的语法制导定义改成栈操作代码

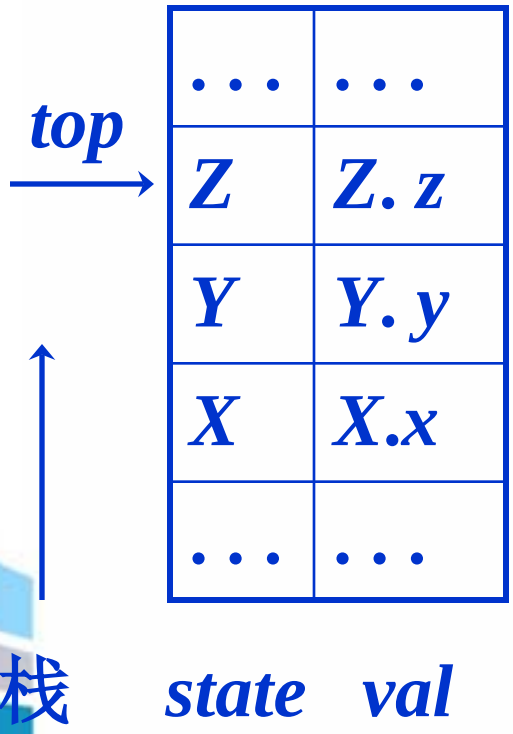
产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	*T.val* = T_1*.val* * *F.val*
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$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



5.2 S属性定义的自下而上计算

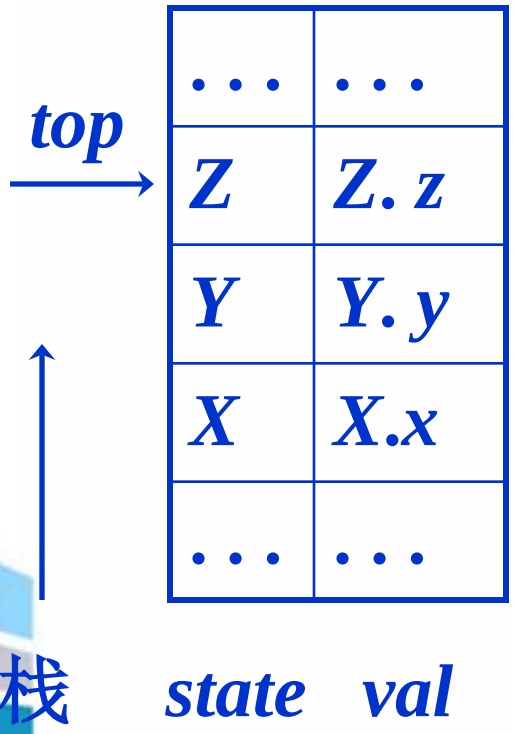
简单计算器的语法制导定义改成栈操作代码

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*]
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*



5.2 S属性定义的自下而上计算

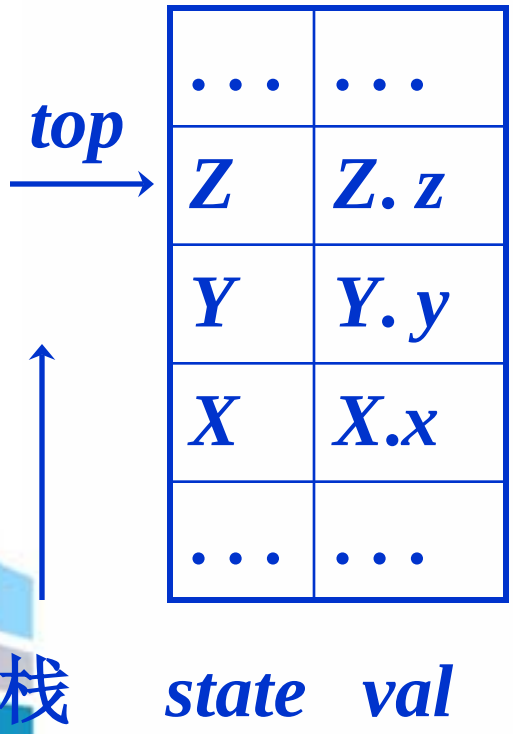
简单计算器的语法制导定义改成栈操作代码



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*

5.2 S属性定义的自下而上计算

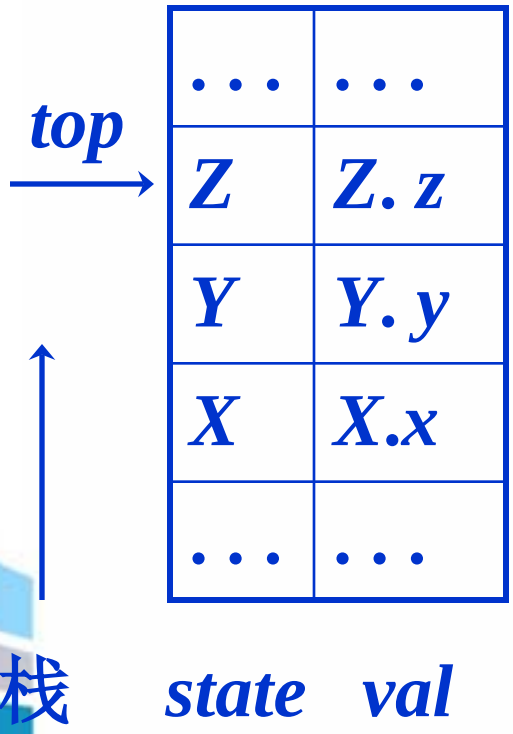
简单计算器的语法制导定义改成栈操作代码



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*]
$T \rightarrow F$	
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

5.2 S属性定义的自下而上计算

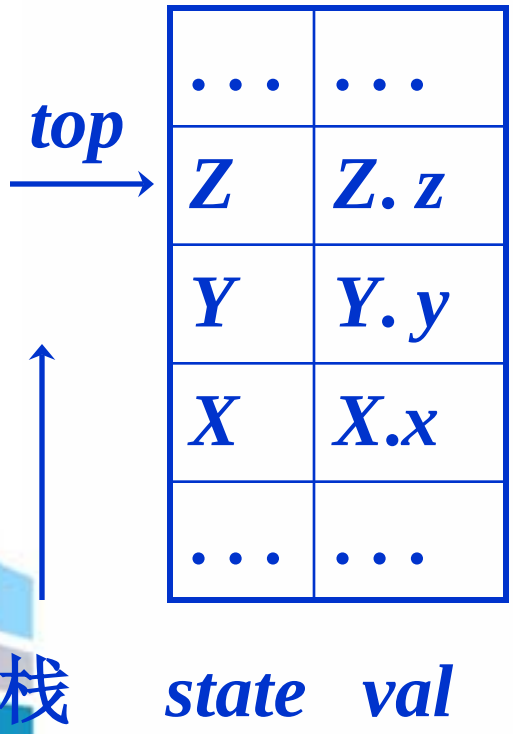
简单计算器的语法制导定义改成栈操作代码



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*]
$E \to T$	
$T \to T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \to F$	
$F \to (E)$	*val* [top –2] = *val* [top –1]
$F \to$ digit	*F.val* = digit.*lexval*

5.2 S属性定义的自下而上计算

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*]
$E \to T$	
$T \to T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \to F$	
$F \to (E)$	*val* [top –2] = *val* [top –1]
$F \to$ digit	

5.3 *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

5.3 L属性定义的自上而下计算

5.3.1 L属性定义

- 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

5.3 L属性定义的自上而下计算

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	$addType(\text{id}.entry, L.in)$

5.3 *L*属性定义的自上而下计算

5.3.2 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

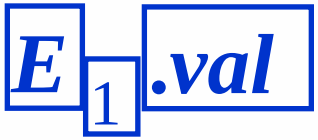
… {*print*(8)}{*print*(5)}{*print*(+)}addop T{*print*(–)}R

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

5.3 *L*属性定义的自上而下计算

- 例 数学排版语言EQN

E sub 1 .val



$S \rightarrow B$

$B \rightarrow B_1\ B_2$

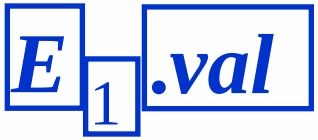
$B \rightarrow B_1$ sub B_2

$B \rightarrow$ text

5.3 L属性定义的自上而下计算

- 例 数学排版语言EQN（语法制导定义）

E sub 1 .val



产 生 式	语 义 规 则
$S \rightarrow B$	$B.ps = 10;\ S.ht = B.ht$
$B \rightarrow B_1\ B_2$	$B_1.ps = B.ps;\ B_2.ps = B.ps;$ $B.ht = max(B_1.ht, B_2.ht)$
$B \rightarrow B_1$ sub B_2	$B_1.ps = B.ps;\ B_2.ps = shrink(B.ps);$ $B.ht = disp\ (B_1.ht, B_2.ht)$
$B \rightarrow$ text	$B.ht = \text{text}.h \times B.ps$

5.3 *L*属性定义的自上而下计算

- 例 数学排版语言EQN（翻译方案）

$S \rightarrow$ $\{B.ps = 10\}$ *B*继承属性的计算

B $\{S.ht = B.ht\}$ 位于*B*的左边

5.3 *L*属性定义的自上而下计算

- 例 数学排版语言EQN（翻译方案）

$S \rightarrow$ $\{B.ps = 10\}$ B $\{S.ht = B.ht\}$

*B*综合属性的计算放在右部末端

5.3 L属性定义的自上而下计算

- 例 数学排版语言EQN（翻译方案）

$S \rightarrow$ {$B.ps = 10$ }

B {$S.ht = B.ht$ }

$B \rightarrow$ {$B_1.ps = B.ps$ }

B_1 {$B_2.ps = B.ps$ }

B_2 {$B.ht = max(B_1.ht, B_2.ht)$ }

$B \rightarrow$ { $B_1.ps = B.ps$ }

B_1

sub { $B_2.ps = shrink(B.ps)$ }

B_2 {$B.ht = disp\ (B_1.ht, B_2.ht)$ }

$B \rightarrow$ text {$B.ht = \text{text}.h \times B.ps$ }

5.3 L属性定义的自上而下计算

- 例 左递归的消除引起继承属性

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

5.3 L属性定义的自上而下计算

$E \to T$ $\quad \{R.i = T.nptr\}$ $\qquad T + T + T + \ldots$

$\quad R$ $\quad \{E.nptr = R.s\}$

$R \to +$

$\quad T$ $\quad \{R_1.i = mkNode(\text{‘+’}, R.i, T.nptr)\}$

$\quad R_1$ $\quad \{R.s = R_1.s\}$

$R \to \varepsilon$ $\quad \{R.s = R.i\}$

$T \to F$ $\quad \{W.i = F.nptr\}$

$\quad W$ $\quad \{T.nptr = W.s\}$

$W \to *$

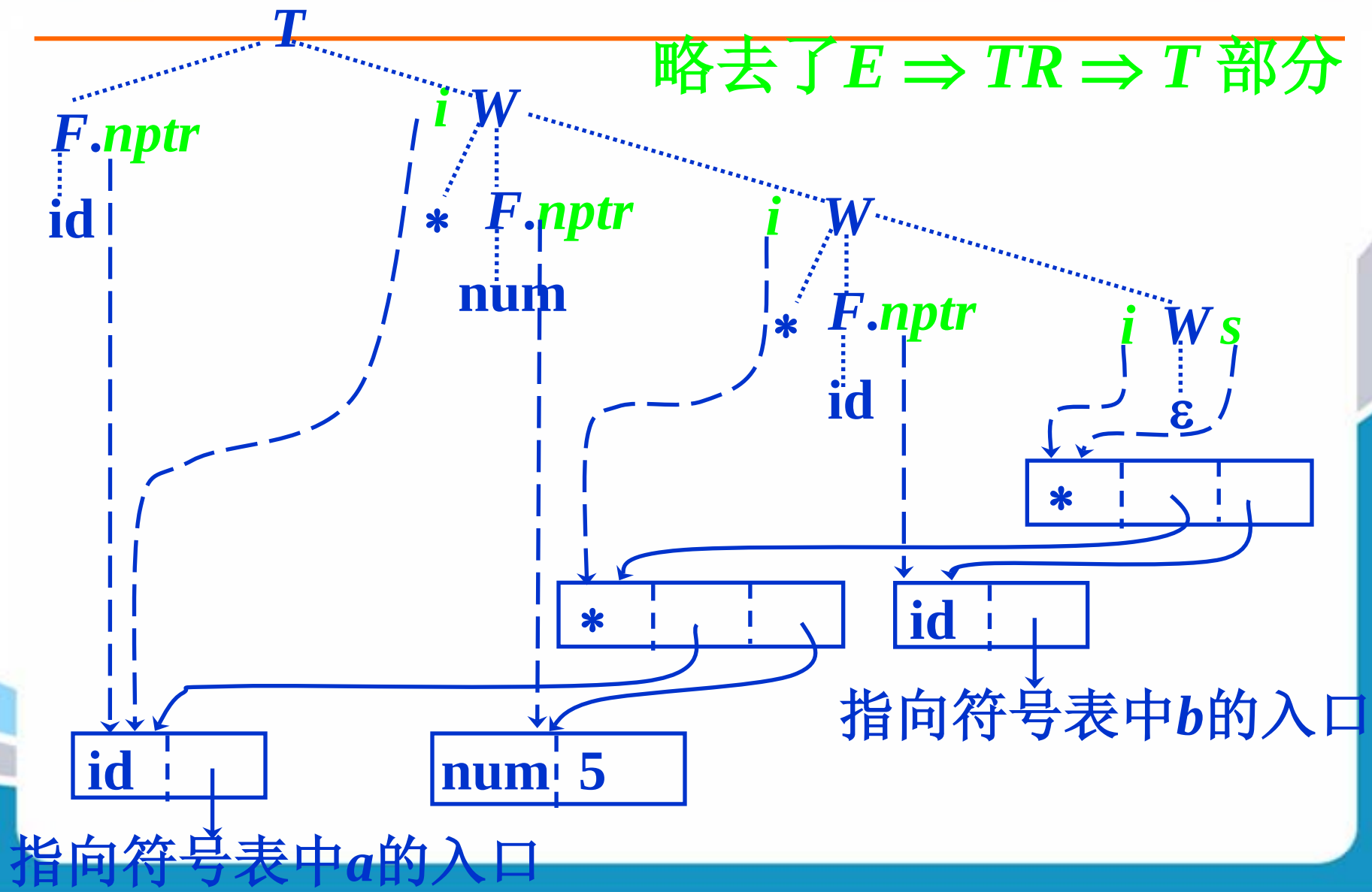
$\quad F$ $\quad \{W_1.i = mkNode(\text{‘*’}, W.i, F.nptr)\}$

$\quad W_1$ $\quad \{W.s = W_1.s\}$

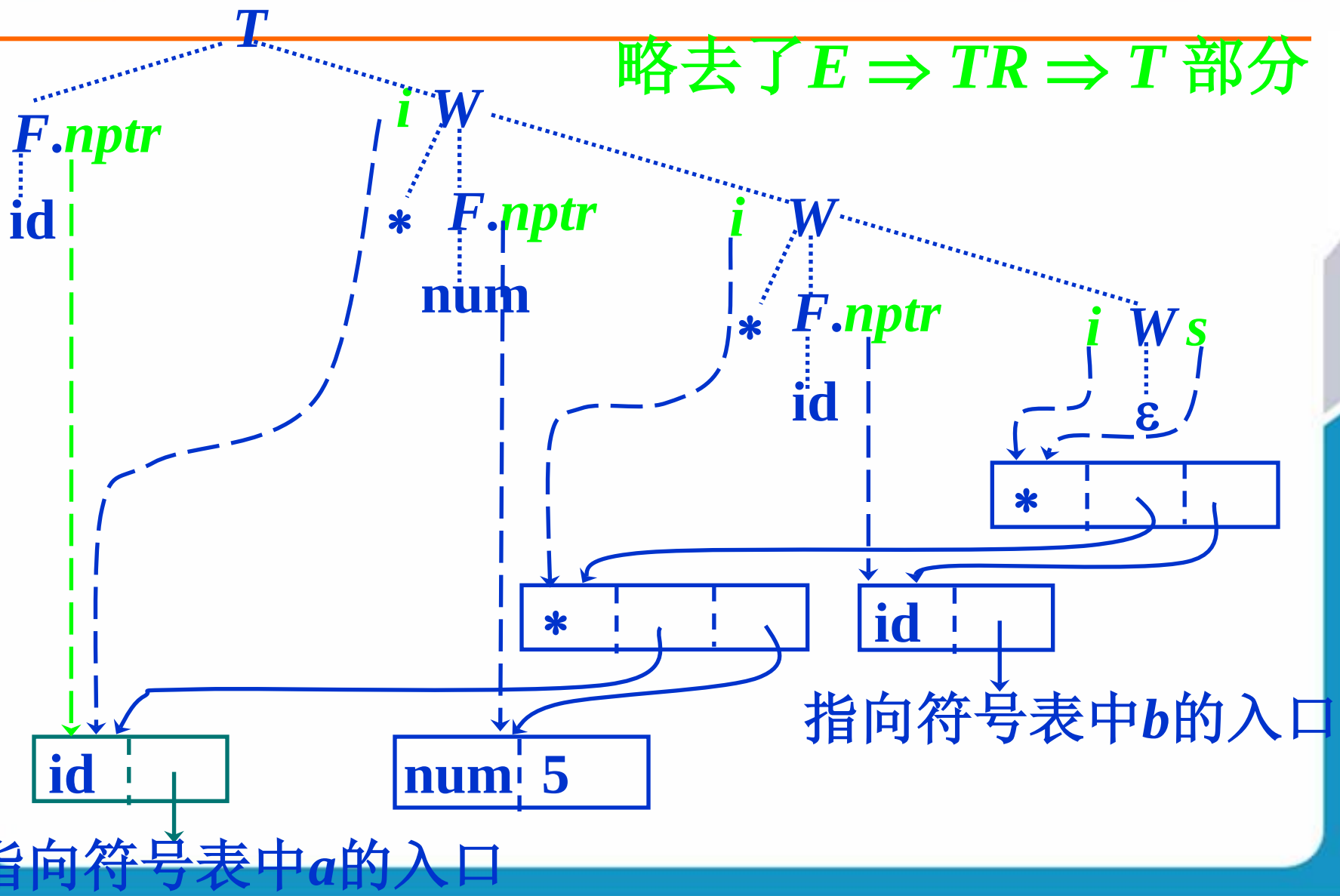
$W \to \varepsilon$ $\quad \{W.s = W.i\}$

F 产生式部分不再给出

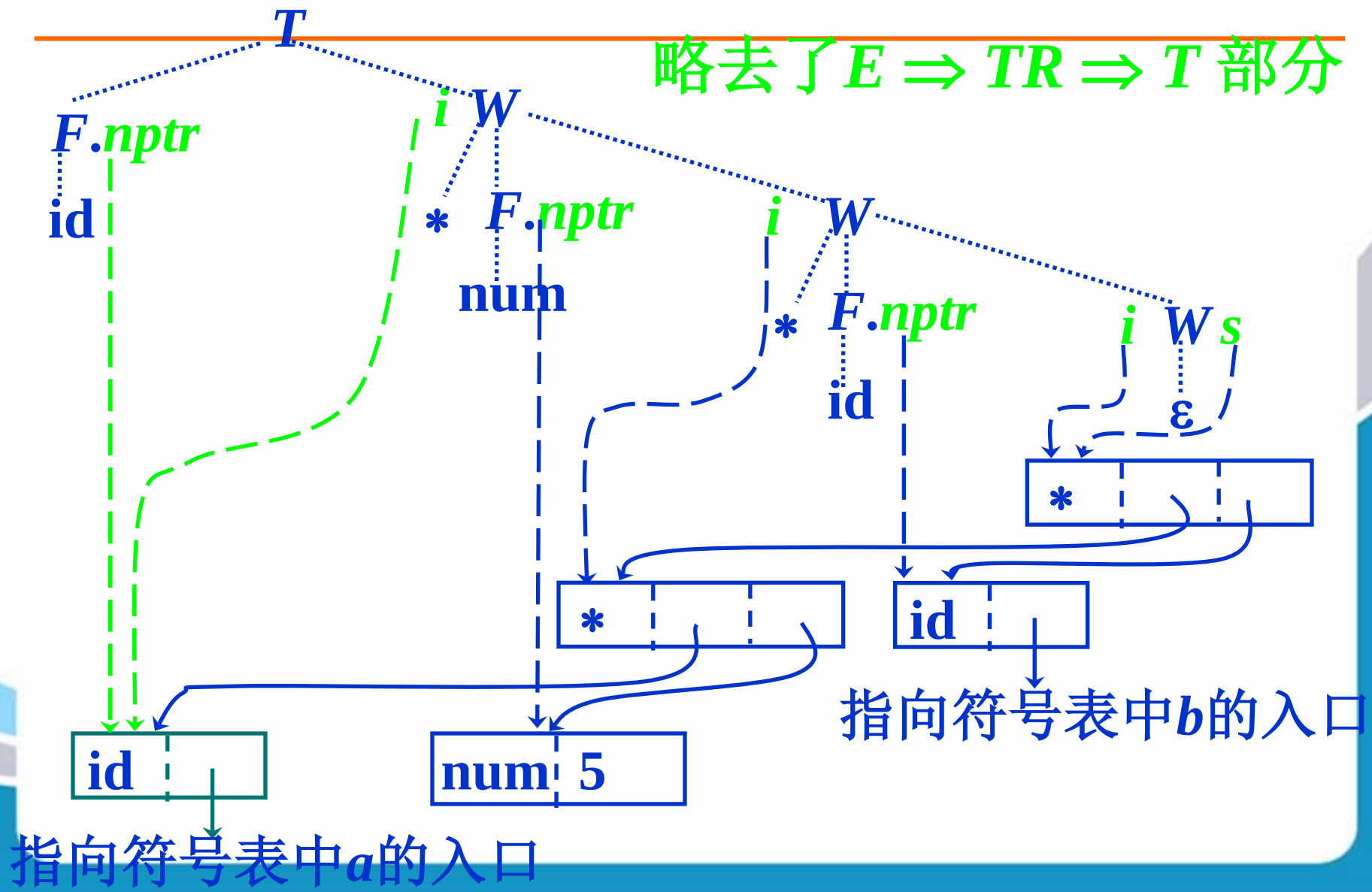
5.3 L属性定义的自上而下计算



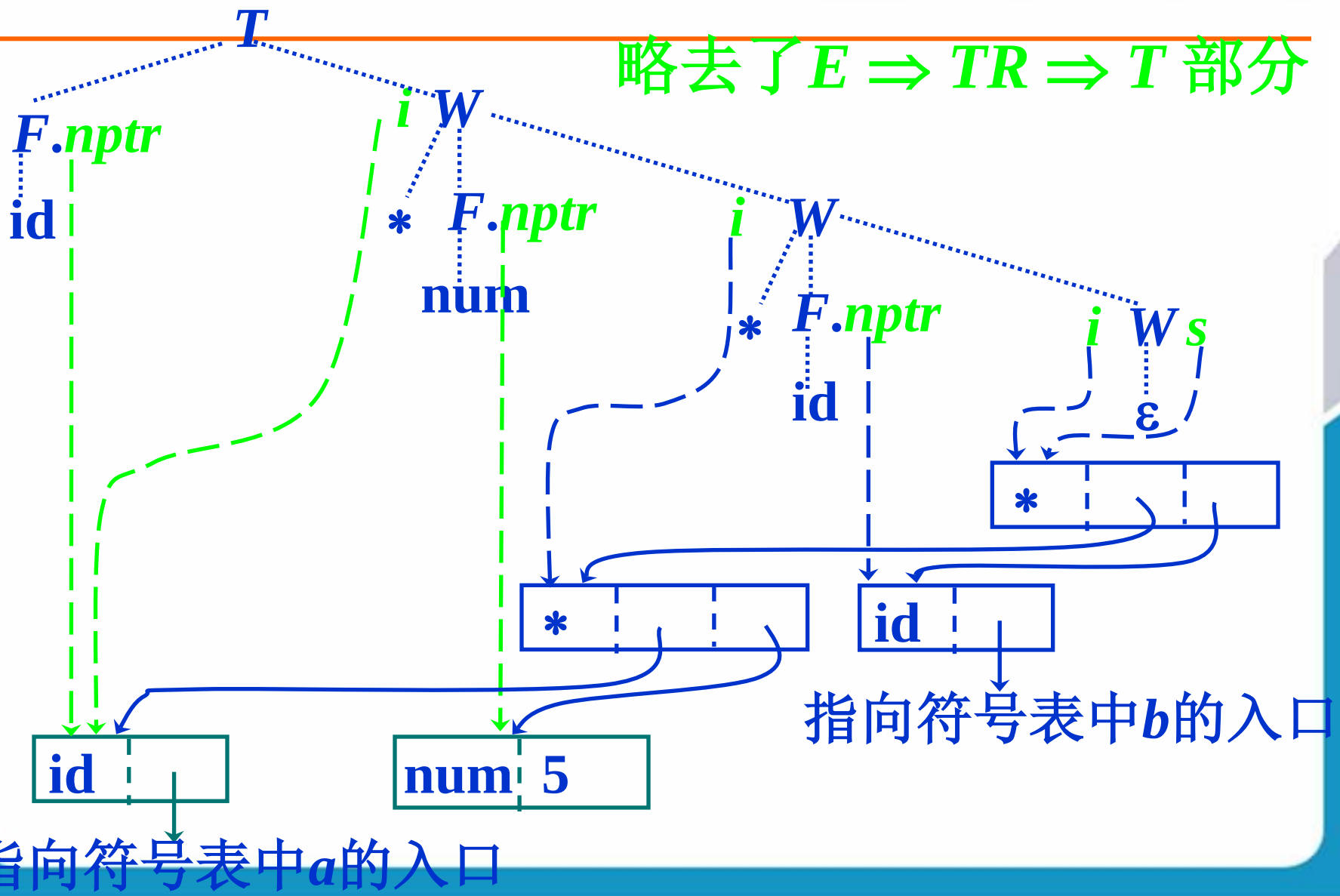
5.3 L属性定义的自上而下计算



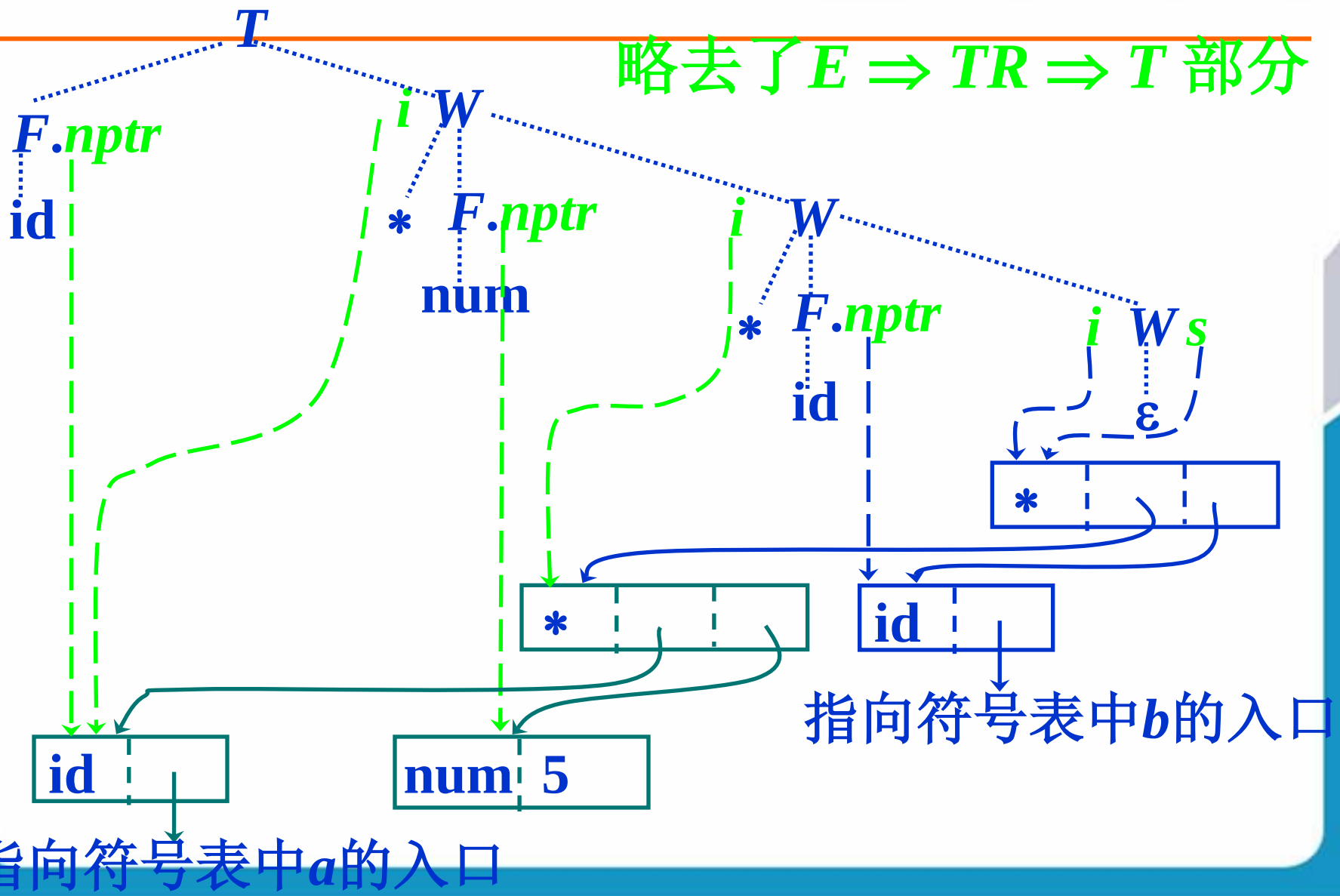
5.3 L属性定义的自上而下计算



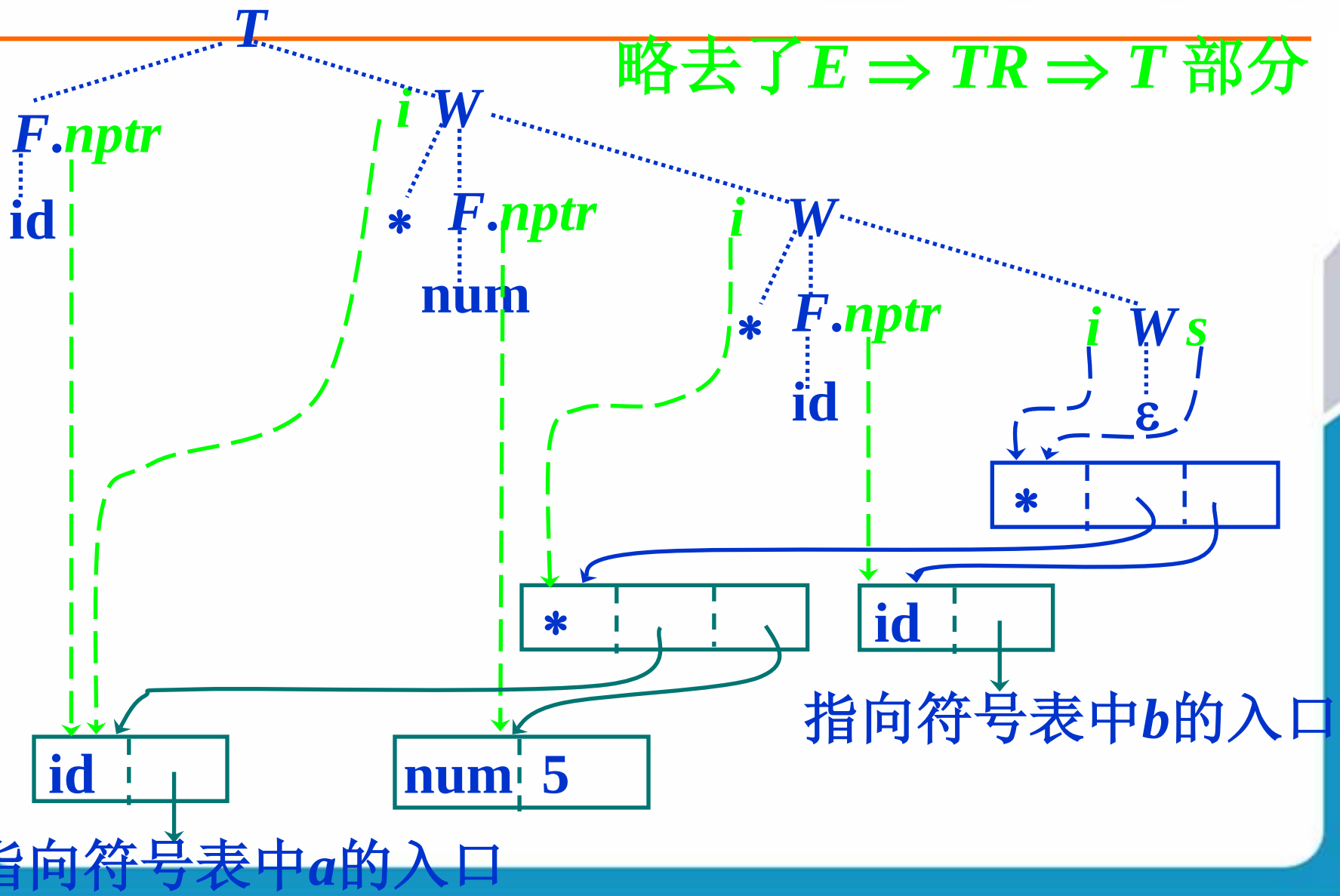
5.3 L属性定义的自上而下计算



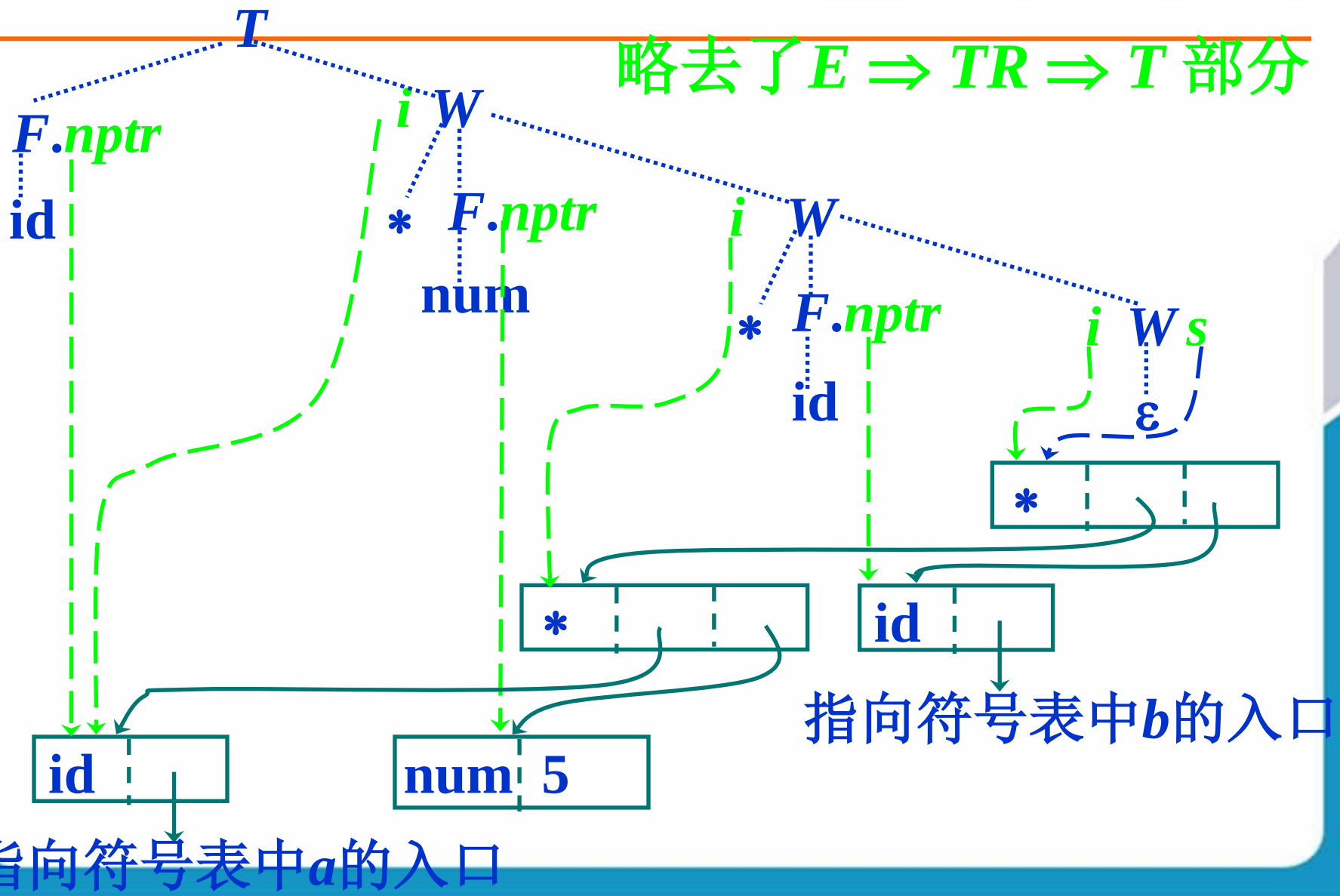
5.3 L属性定义的自上而下计算



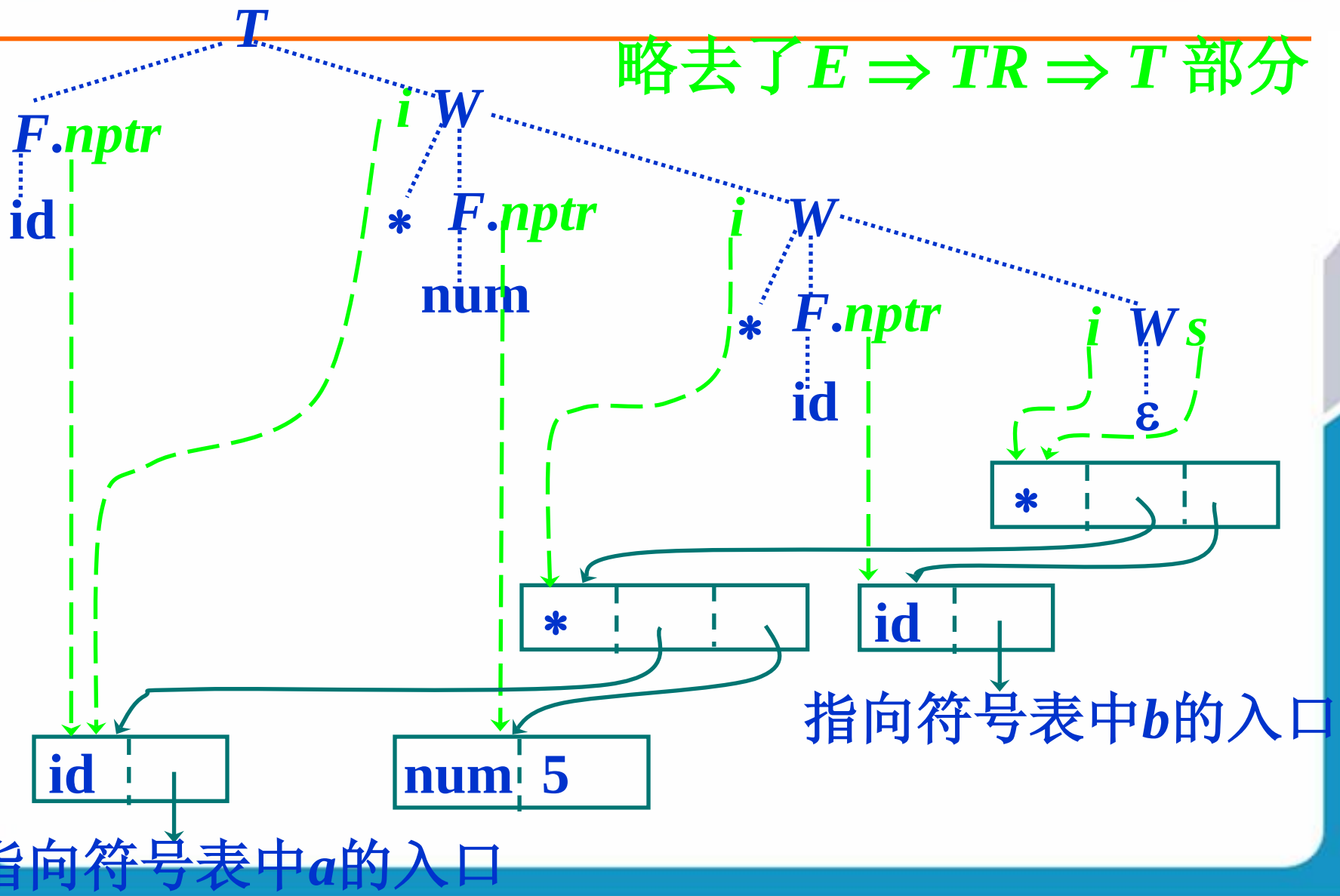
5.3 L属性定义的自上而下计算



5.3 L属性定义的自上而下计算



5.3 L属性定义的自上而下计算



5.3 *L*属性定义的自上而下计算

5.3.3 预测翻译器的设计

把预测分析器的构造方法推广到翻译方案的实现

产生式$R \rightarrow +TR \mid \varepsilon$ 的分析过程

```
void R( ) {
   if (lookahead == '+' ) {
        match ( '+' ); T( ); R( );
   }
   else  /* 什么也不做 */
}
```

5.3 *L*属性定义的自上而下计算

```
syntaxTreeNode* R (syntaxTreeNode* i) {
   syntaxTreeNode *nptr, *i1, *s1, *s;
   char addoplexeme;

   if (lookahead == '+' ) { /* 产生式 R → +T R  */
        addoplexeme = lexval;
        match('+' ); nptr = T();
        i1 = mkNode(addoplexeme, i , nptr);
        s1 = R (i1); s = s1;
   }
   else  s = i;  /* 产生式 R → ε  */
   return s;
}
```

R : *i*, *s*

T : *nptr*

\+ : *addoplexeme*

5.4 *L*属性的自下而上计算

在自下而上分析的框架中实现*L*属性定义的方法

- 它能实现任何基于**LL(1)**文法的*L*属性定义
- 也能实现许多（但不是所有的）基于**LR(1)** 的*L*属性定义

5.4 *L*属性的自下而上计算

5.4.1 删除翻译方案中嵌入的动作

$E \rightarrow T\ R$

$R \rightarrow +\ T\ \{print\ (\text{‘+’})\}R_1 \mid -\ T\ \{print\ (\text{‘–’})\}R_1 \mid \varepsilon$

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

在文法中加入产生ε的标记非终结符，让每个嵌入动作由不同标记非终结符*M*代表，并把该动作放在产生式$M \rightarrow \varepsilon$的右端

$E \rightarrow T\ R$

$R \rightarrow +\ T\ M\ R_1 \mid -\ T\ N\ R_1 \mid \varepsilon$

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

$M \rightarrow \varepsilon\ \{print\ (\text{‘+’})\}$

$N \rightarrow \varepsilon\ \{print\ (\text{‘–’})\}$

这些动作的一个重要特点：

没有引用原来产生式文法符号的属性

5.4 *L*属性的自下而上计算

5.4.2 分析栈上的继承属性

例 int p, q, r

$D \rightarrow T$ {*L.in* = *T.type*}

L

$T \rightarrow$ int {*T. type* = *integer*}

$T \rightarrow$ real {*T. type* = *real*}

$L \rightarrow$ {L_1.*in* = *L.in* }

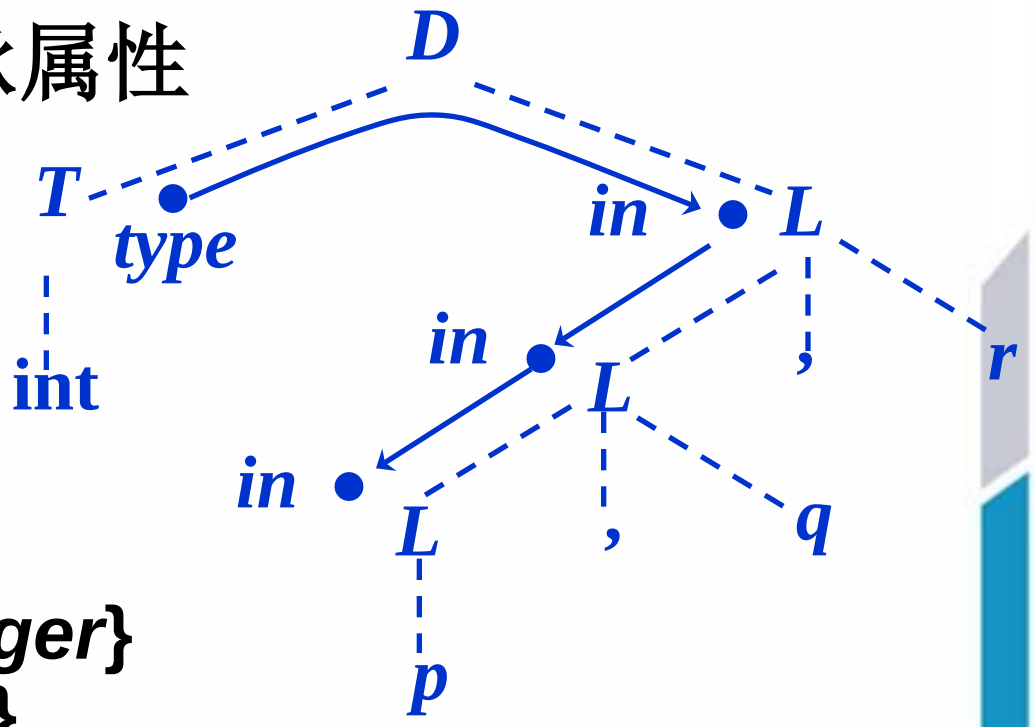
L_1, id {*addtype* (id.*entry*, *L.in*)}

$L \rightarrow$ id {*addtype* (id.*entry*, *L.in*)}

5.4 *L*属性的自下而上计算

5.4.2 分析栈上的继承属性

1、属性位置能预测



例 int p, q, r

D → *T* {*L.in* = *T.type*}
L

T→ int {*T. type* = *integer*}

T→ real {*T. type* = *real*}

L→ {L_1.*in* = *L.in* }
L_1, id {*addtype* (id.*entry*, *L.in*)}

L→ id {*addtype* (id.*entry*, *L.in*)}

5.4 L属性的自下而上计算

5.4.2 分析栈上的继承属性

1、属性位置能预测

例 int p, q, r

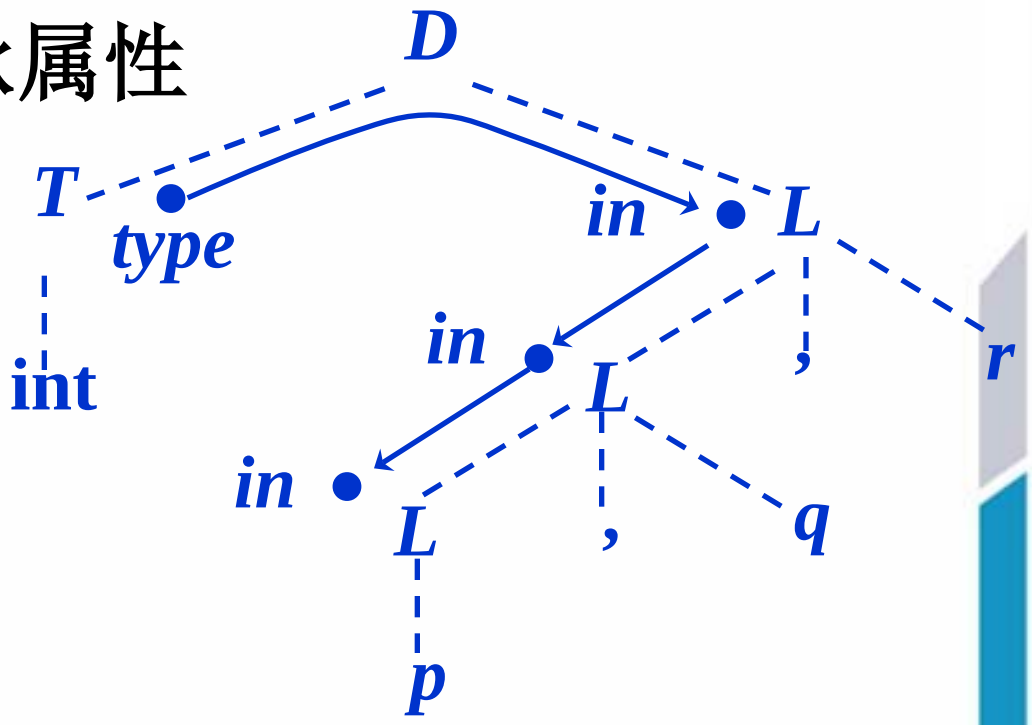
$D \rightarrow T$ {*L.in = T.type*}
　　L

$T \rightarrow$ int {*T. type = integer*}

$T \rightarrow$ real {*T. type = real*}

$L \rightarrow$ {L_1*.in = L.in* }
　L_1, id {*addtype* (id.*entry*, *L.in*)}

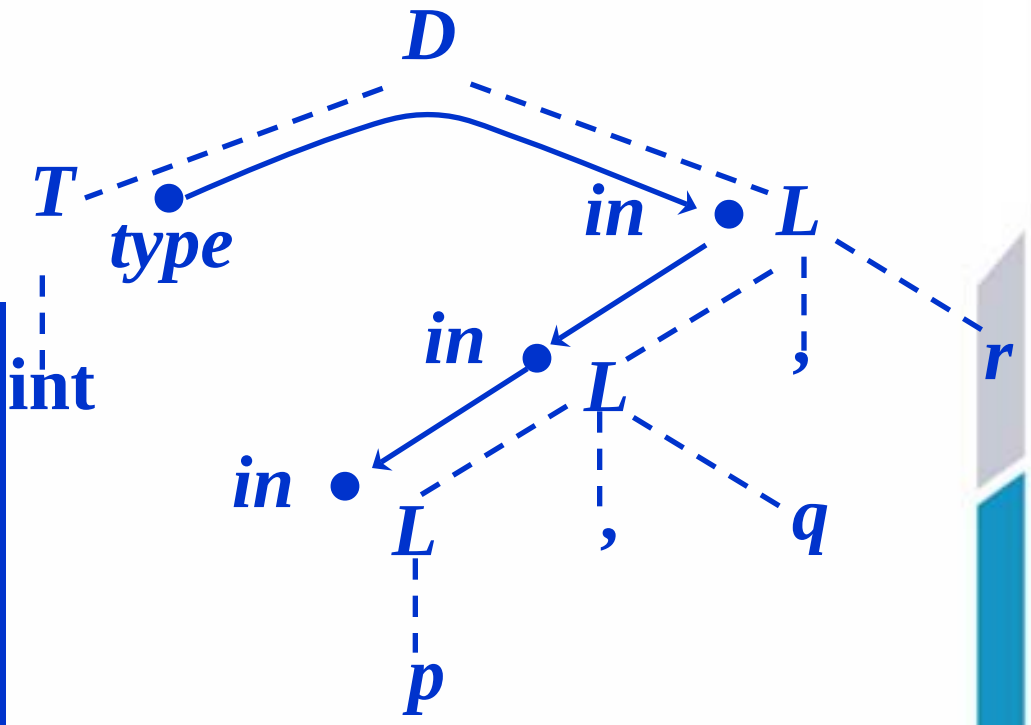
$L \rightarrow$ id {*addtype* (id.*entry*, *L.in*)}



继承属性的计算可以略去，引用继承属性的地方改成引用其他符号的综合属性

5.4 *L*属性的自下而上计算

产 生 式	代 码 段
$D \rightarrow TL$	
$T \rightarrow$ int	*val*[*top*] = *integer*
$T \rightarrow$ real	*val*[*top*] = *real*
$L \rightarrow L_1$, id	*addType*(*val*[*top*], *val*[*top*–3])
$L \rightarrow$ id	*addType*(*val*[*top*], *val*[*top*–1])



5.4 L属性的自下而上计算

2、属性的位置不能预测

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABC$	$C.i = A.s$
$C \rightarrow c$	$C.s = g(C.i)$

- 增加标记非终结符，使得位置可以预测

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABMC$	$M.i = A.s;\ C.i = M.s$
$C \rightarrow c$	$C.s = g(C.i)$
$M \rightarrow \varepsilon$	$M.s = M.i$

5.4 L属性的自下而上计算

2、属性的位置不能预测

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABC$	$C.i = A.s$
$C \rightarrow c$	$C.s = g(C.i)$

- 增加标记非终结符，使得位置可以预测

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABMC$	$M.i = A.s$; $C.i = M.s$
$C \rightarrow c$	$C.s = g(C.i)$ 还得考虑 $M.s$
$M \rightarrow \varepsilon$	$M.s = M.i$ 计算的可预测

5.4 L属性的自下而上计算

5.4.3 模拟继承属性的计算

继承属性是某个综合属性的一个函数

$S \rightarrow aAC$	$C.i = f(A.s)$
$C \rightarrow c$	$C.s = g(C.i)$

- 增加标记非终结符，把$f(A.s)$的计算移到对标记非终结符归约时进行

$S \rightarrow aANC$	$N.i = A.s;\ C.i = N.s$
$N \rightarrow \varepsilon$	$N.s = f(N.i)$
$C \rightarrow c$	$C.s = g(C.i)$

5.4 *L*属性的自下而上计算

- 例 数学排版语言EQN

$S \rightarrow$ { $B.ps = 10$ }
B { $S.ht = B.ht$ }

$B \rightarrow$ { $B_1.ps = B.ps$ }
B_1 { $B_2.ps = B.ps$ }
B_2 { $B.ht = max(B_1.ht, B_2.ht)$ }

$B \rightarrow$ { $B_1.ps = B.ps$ }
B_1
sub { $B_2.ps = shrink(B.ps)$ }
B_2 { $B.ht = disp(B_1.ht, B_2.ht)$ }

$B \rightarrow$ text { $B.ht = \text{text}.h \times B.ps$ }

5.4 L属性的自下而上计算

产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

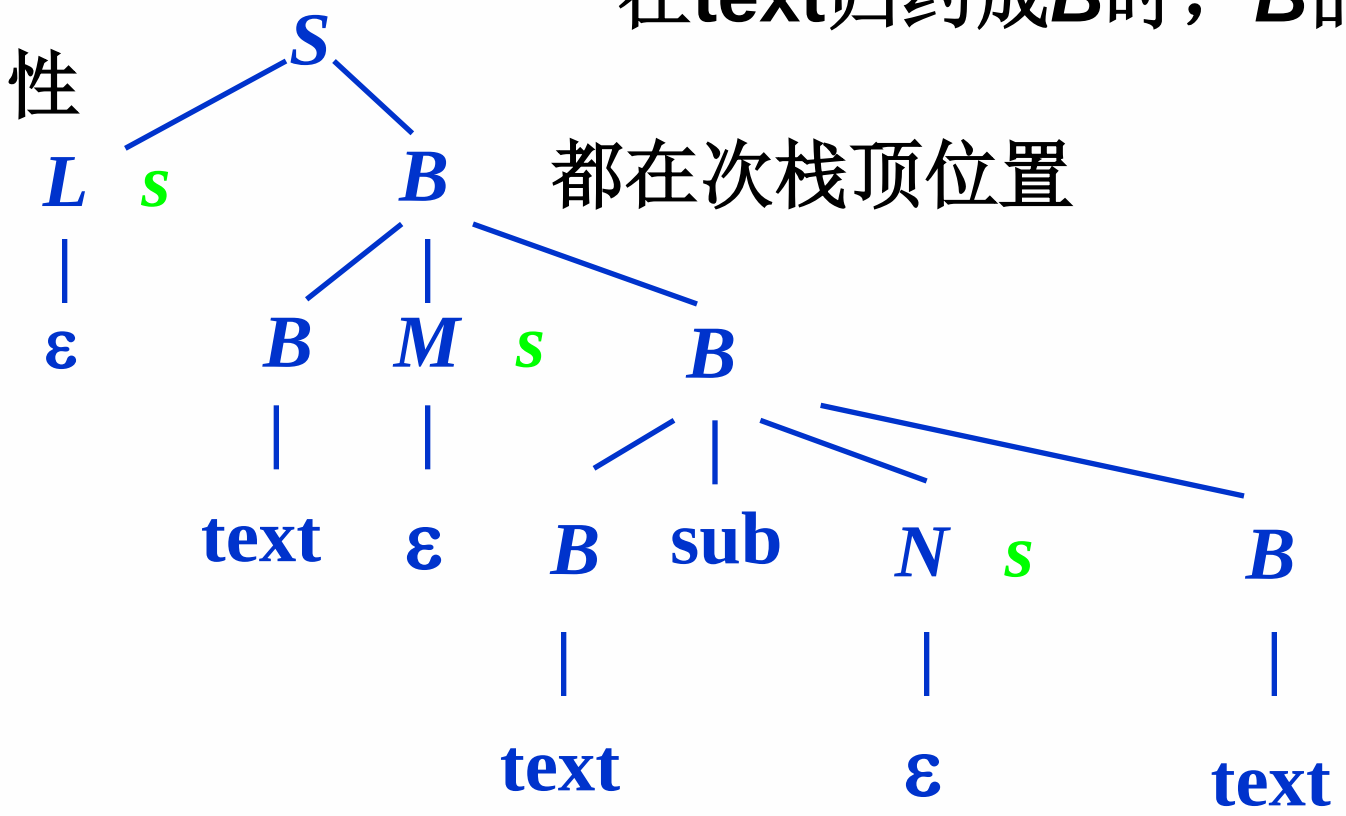
5.4 L属性的自下而上计算

产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$ 兼有计算功能
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 *L*属性的自下而上计算

举例说明

在text归约成*B*时，*B*的*ps*属性



都在次栈顶位置

5.4 L属性的自下而上计算

产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$; $B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht\ )$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht\ )$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$val[top-3] = disp\ (val[top-3], val[top]\ )$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht =$ text$.h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$val[top-3] = disp\ (val[top-3], val[top]\ )$
$N \rightarrow \varepsilon$	$val[top+1] = shrink(val[top-2])$
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

5.4 L属性的自下而上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$val[top-3] = disp\ (val[top-3], val[top]\ )$
$N \rightarrow \varepsilon$	$val[top+1] = shrink(val[top-2])$
$B \rightarrow$ text	$val[top] = val[top] \times val[top-1]$

本 章 要 点

- 语义规则的两种描述方法：语法制导的定义和翻译方案
- 设计简单问题的语法制导定义和翻译方案，这是本章的重点和难点
- 语法制导定义和翻译方案的实现
  - *S*属性的自下而上计算（边分析边计算）
  - *L*属性的自上而下计算（边分析边计算）
  - *L*属性的自下而上计算（边分析边计算）
- 不再介绍先分析后计算的方法
  - 不能边分析边计算的情况是存在的，见5.6节

例　题　1

下面是产生字母表Σ = {0, 1, 2}上数字串的一个文法：

$S \rightarrow D\ S\ D \mid 2$

$D \rightarrow 0 \mid 1$

写一个语法制导定义，判断它接受的句子是否为回文数

$S' \rightarrow S$	print(S.*val*)
$S \rightarrow D_1\ S_1\ D_2$	S.*val* = (D_1.*val* == D_2.*val*) and S_1.*val*
$S \rightarrow 2$	S.*val* = *true*
$D \rightarrow 0$	D.*val* = 0
$D \rightarrow 1$	D.*val* = 1

例　题　2

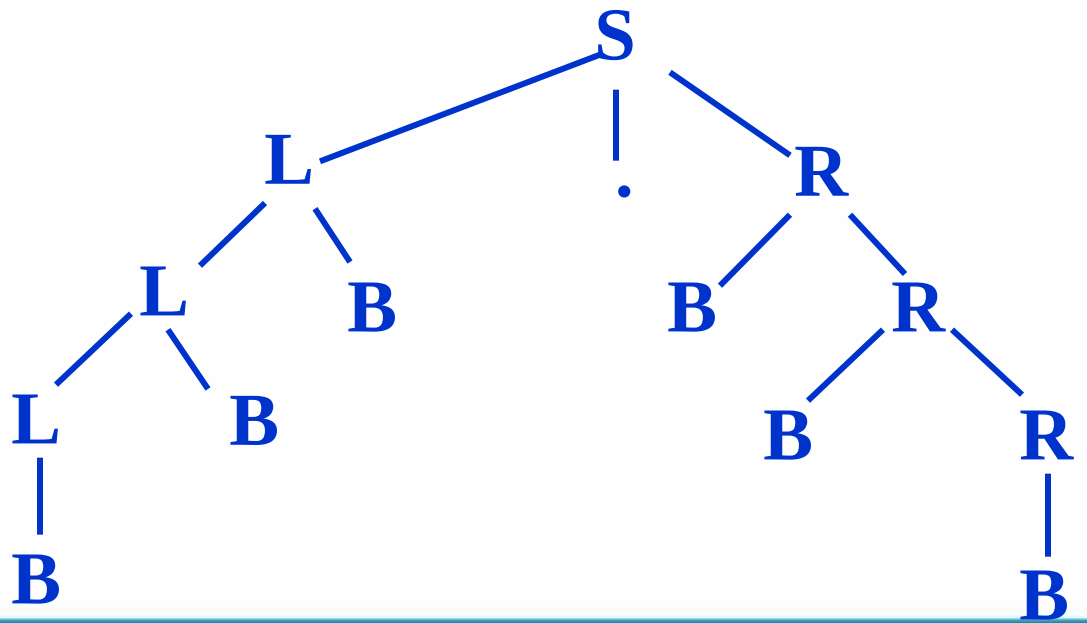
为下面文法写一个语法制导的定义，用S的综合属性*val*给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625（可以修改文法）

$S \rightarrow L . R \mid L$

$L \rightarrow L\ B \mid B$

$R \rightarrow B\ R \mid B$

$B \rightarrow 0 \mid 1$



例　题　2

$S \rightarrow L\ .\ R$	$S.\ val = L.\ val + R.\ val$
$S \rightarrow L$	$S.\ val = L.\ val$
$L \rightarrow L_1\ B$	$L.\ val = L_1.\ val \times 2 + B.\ val$
$L \rightarrow B$	$L.\ val = B.\ val$
$R \rightarrow B\ R_1$	$R.\ val = R_1.\ val\ /\ 2 + B.\ val\ /\ 2$
$R \rightarrow B$	$R.\ val = B.\ val\ /\ 2$
$B \rightarrow 0$	$B.\ val = 0$
$B \rightarrow 1$	$B.\ val = 1$

例　题　3

给出把中缀表达式翻译成没有冗余括号的中缀表达式的语法制导定义。例如, 因为+和*是左结合,

((*a* * (*b* + *c*)) * (*d*))可以重写成*a* * (*b* + *c*) * *d*

例　题　3

第一种方法

$S' \rightarrow E$　　*print* (*E*. *code*)

$E \rightarrow E_1 + T$

　　if *T*. *op* == *plus* then

　　　　E.*code* =E_1.*code*||“+”||“(”||*T*.*code*||“)”

　　else

　　　　E. *code* = E_1. *code* || “+” || *T*. *code*;

　　E. *op* = *plus*

$E \rightarrow T$　　*E*. *code* = *T*. *code*; *E*. *op* = *T*. *op*

例　题　3

$T \rightarrow T_1 * F$

if (*F*. *op* == *plus*) or (*F*. *op* == *times*) then

if T_1. *op* == *plus* then

T. *code* = “(” || T_1. *code* || “)” || “*” || “(” || *F*. *code* || “)”

else

T. *code* = T_1. *code* || “*” || “(” || *F*. *code* || “)”

else if T_1. *op* = *plus* then

T. *code* = “(” || T_1. *code* || “)” || “*” || *F*. *code*

else

T. *code* = T_1. *code* || “*” || *F*. *code*;

T. *op* = *times*

例　题　3

$T \rightarrow F$

$T.code = F.code;\ \ T.op = F.op$

$F \rightarrow id$

$F.code = id.lexeme;\ \ F.op = id$

$F \rightarrow (\ E\ )$

$F.code = E.code;\ \ F.op = E.op$