



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

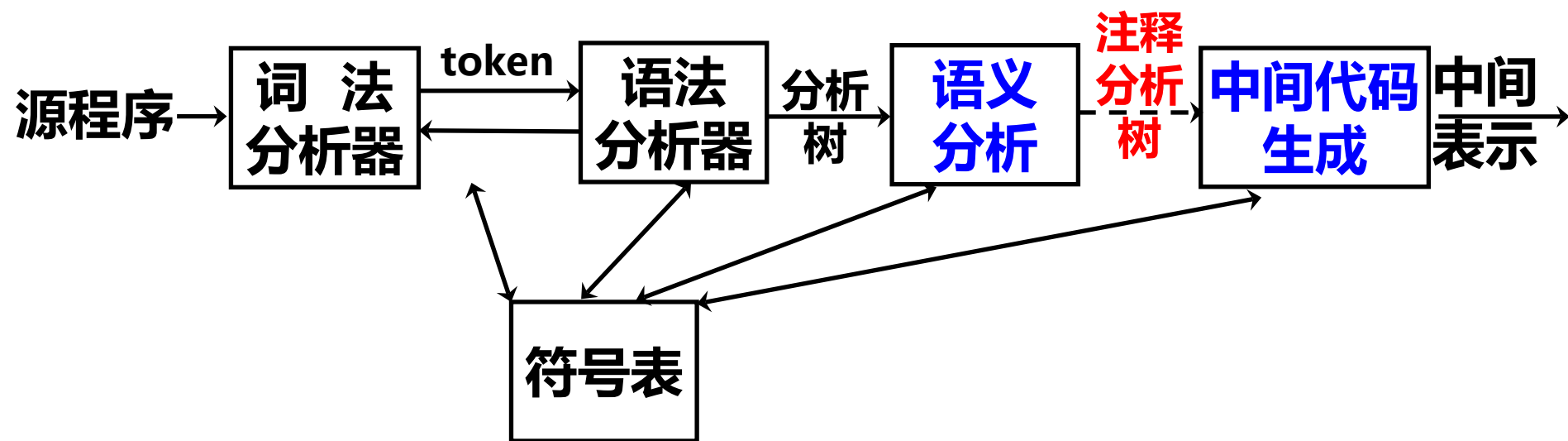


# 《编译原理与技术》 语法制导翻译 I

计算机科学与技术学院

李 诚

2021-10-11



## □语法制导翻译简介

## □语法制导定义

- ❖ 属性、属性依赖图、计算次序
- ❖ S属性的定义、L属性的定义
- ❖ 语法制导定义的应用



□编译程序的目标：将源程序翻译成为**语义等价**的目标程序。

❖源程序与目标程序**具有不同的语法结构**，表达的结果却是相同的。

□语法制导翻译

❖使用**上下文无关文法(CFG)**来引导对语言的翻译，是一种面向文法的翻译技术



## □语义分析：对结构上正确的源程序进行上下文有关性质的审查

❖例：每个算符是否具有语言规范允许的运算对象

一个C程序片断

```
int arr[2], b;
```

```
b = arr * 10;
```



## □语义分析：对结构上正确的源程序进行上下文有关性质的审查

- ❖例：每个算符是否具有语言规范允许的运算对象
- ❖例：数组访问越界

一个C程序片断

```
int a[10];
```

```
a[10] = 5;
```



## □语义分析：对结构上正确的源程序进行上下文有关性质的审查

- ❖例：每个算符是否具有语言规范允许的运算对象
- ❖例：数组访问越界
- ❖例：类型强制转换



□语义分析：对结构上正确的源程序进行上下文有关性质的审查

□中间代码生成：将源程序翻译为中间代码

- ❖ 复杂性介于源语言和机器语言的一种表示形式
- ❖ 便于生成目标代码、机器无关优化、移植

```
position = initial + rate * 60
```

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```



## □如何表示语义信息？

❖为CFG中的文法符号设置**语义属性**，用来表示语法成分对应的语义信息





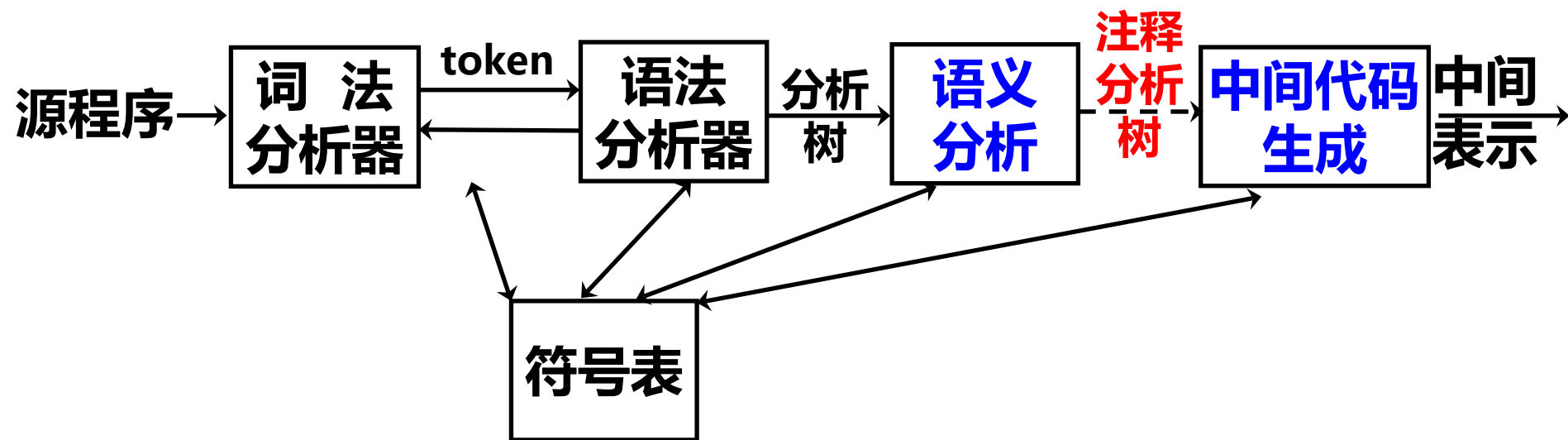
## □如何表示语义信息？

❖为CFG中的文法符号设置**语义属性**，用来表示语法成分对应的语义信息

## □如何计算语义属性？

❖文法符号的语义属性值是用与文法符号所在产生式（**语法规则**）相关联的**语义规则**来计算的

❖对于给定的输入串 $x$ ，构建 $x$ 的语法分析树，并利用与产生式相关联的**语义规则**来计算分析树中各结点对应的语义属性值



## □语法制导翻译简介

## □语法制导定义

- ❖ 属性、属性依赖图、计算次序
- ❖ S属性的定义、L属性的定义
- ❖ 语法制导定义的应用



## □语法制导定义 (Syntax-Directed Definition, SDD)

- ❖ 基础的上下文无关文法
- ❖ 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  $b = f(c_1, c_2, \dots, c_k)$  的语义规则，其中  $f$  是函数  $b$  和  $c_1, c_2, \dots, c_k$  是该产生式文法符号的属性



## □语法制导定义 (Syntax-Directed Definition, SDD)

❖基础的上下文无关文法

❖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

$b = f(c_1, c_2, \dots, c_k)$  的语义规则，其中  $f$  是函数

$b$  和  $c_1, c_2, \dots, c_k$  是该产生式文法符号的属性

❖综合属性(synthesized attribute): 如果  $b$  是  $A$  的属性,  
 $c_1, c_2, \dots, c_k$  是产生式右部文法符号的属性或  $A$  的其它属性

终结符只能有综合属性，属性值无需计算，由词法分析给定



# 综合属性：举例



结尾标记

带副作用  
的规则  
(虚拟属性)

产生式	语义规则
$L \rightarrow E \mathbf{n}$	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

对E加下  
标以区分  
不同的属  
性值

词法分析  
给定



**□思考：如何将语义规则所引起的属性计算与语法分析结合起来？**

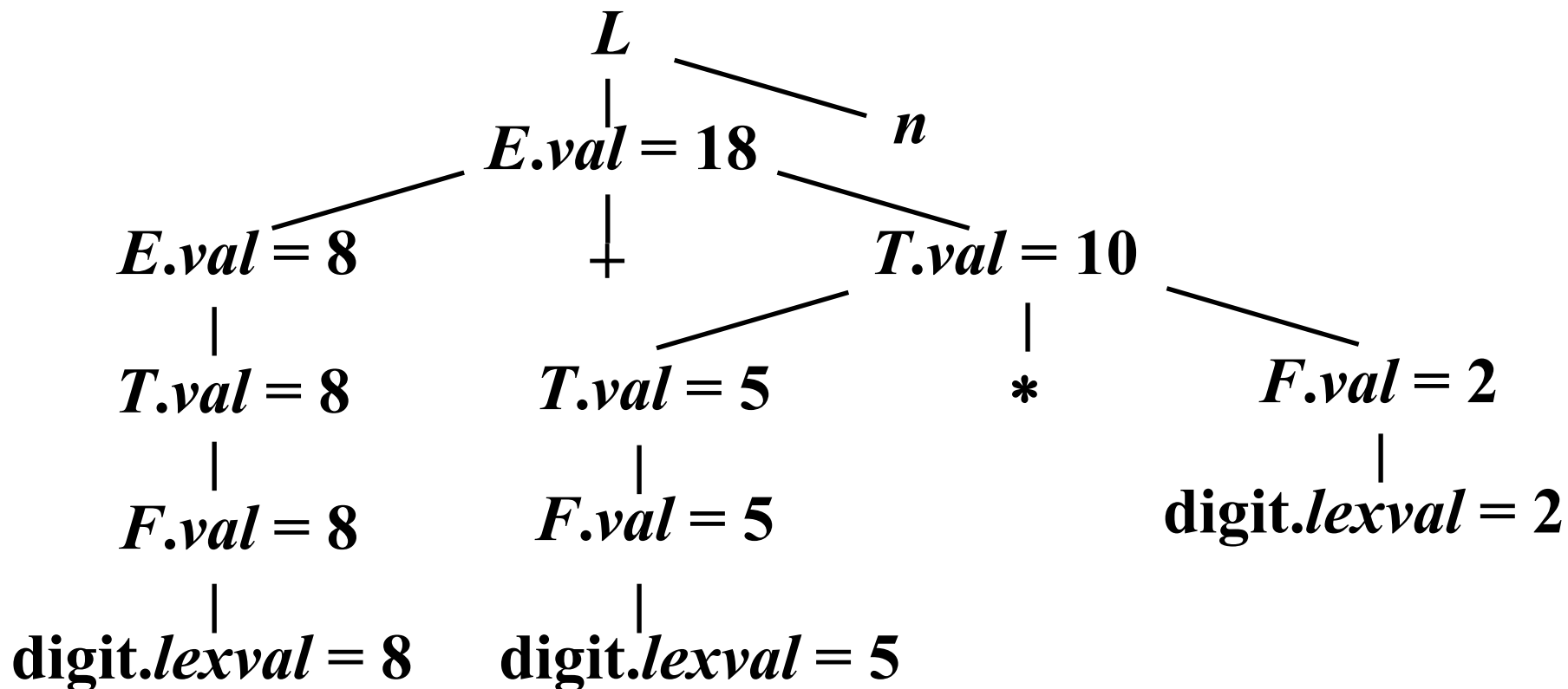


- 思考：如何将语义规则所引起的属性计算与语法分析结合起来？
- 语法分析的过程**显式或隐式**地构造了分析树(parse tree)
- 因此，可以先构造分析树，在分析树上，对每一个节点上的属性值进行求值
  - ❖注释分析树(Annotated parse tree)



□定义：结点的属性值都标注出来的分析树

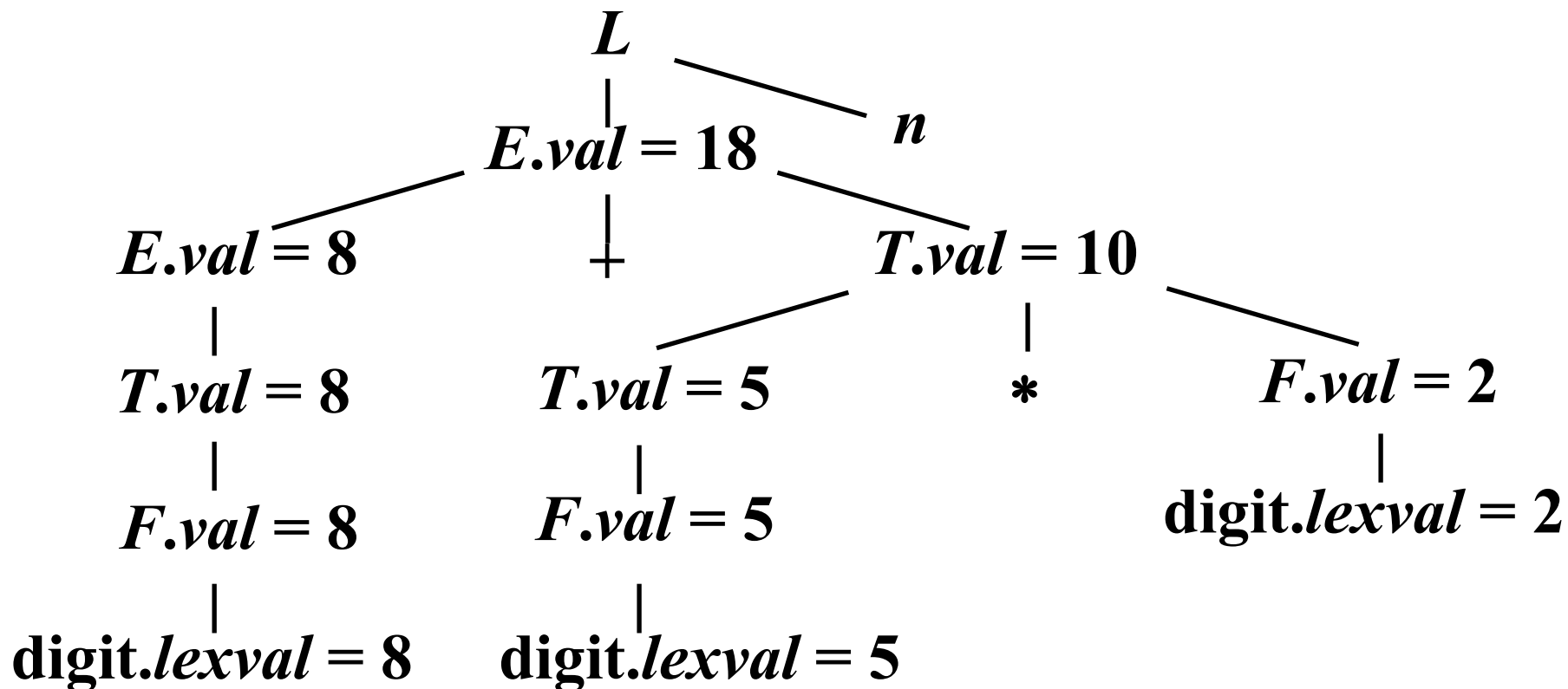
$8+5*2$   $n$ 的注释分析树





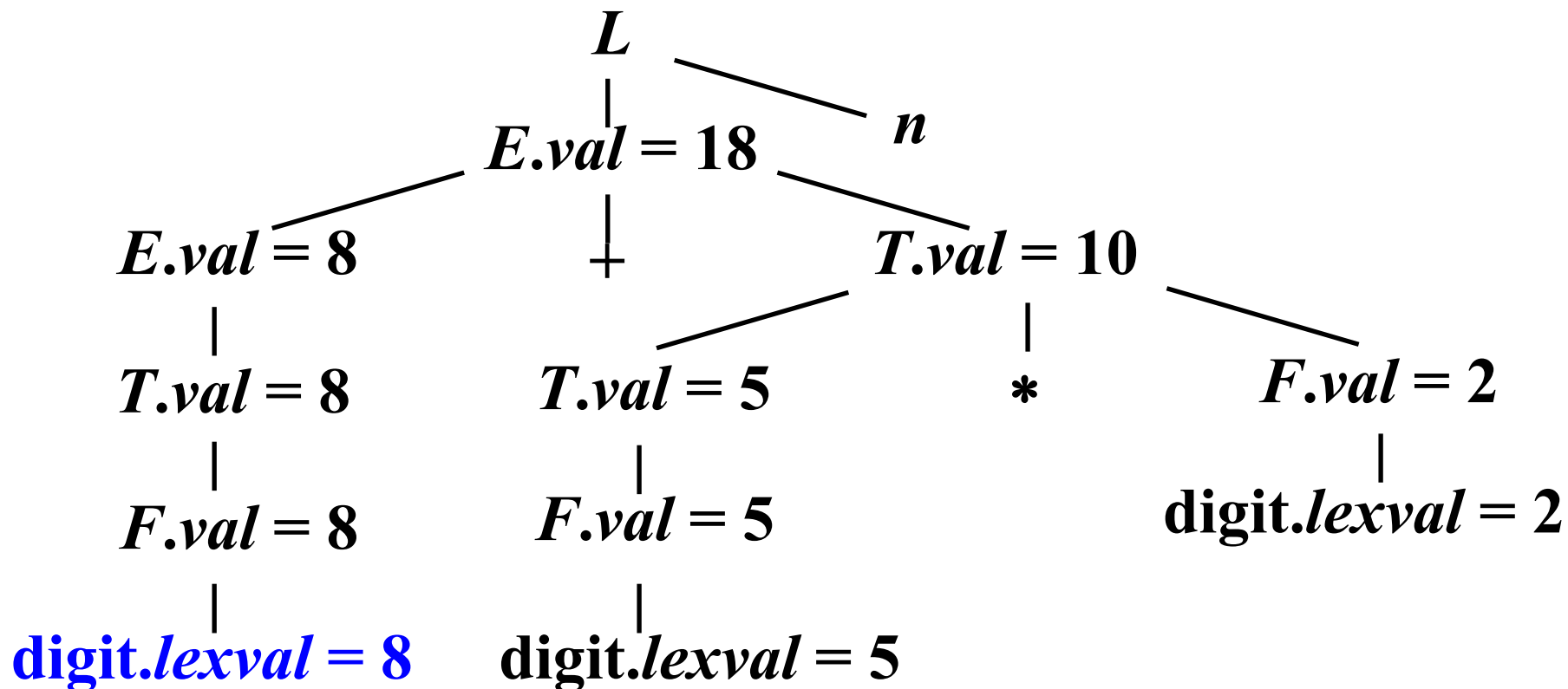


□各结点综合属性的计算可以自底向上地完成



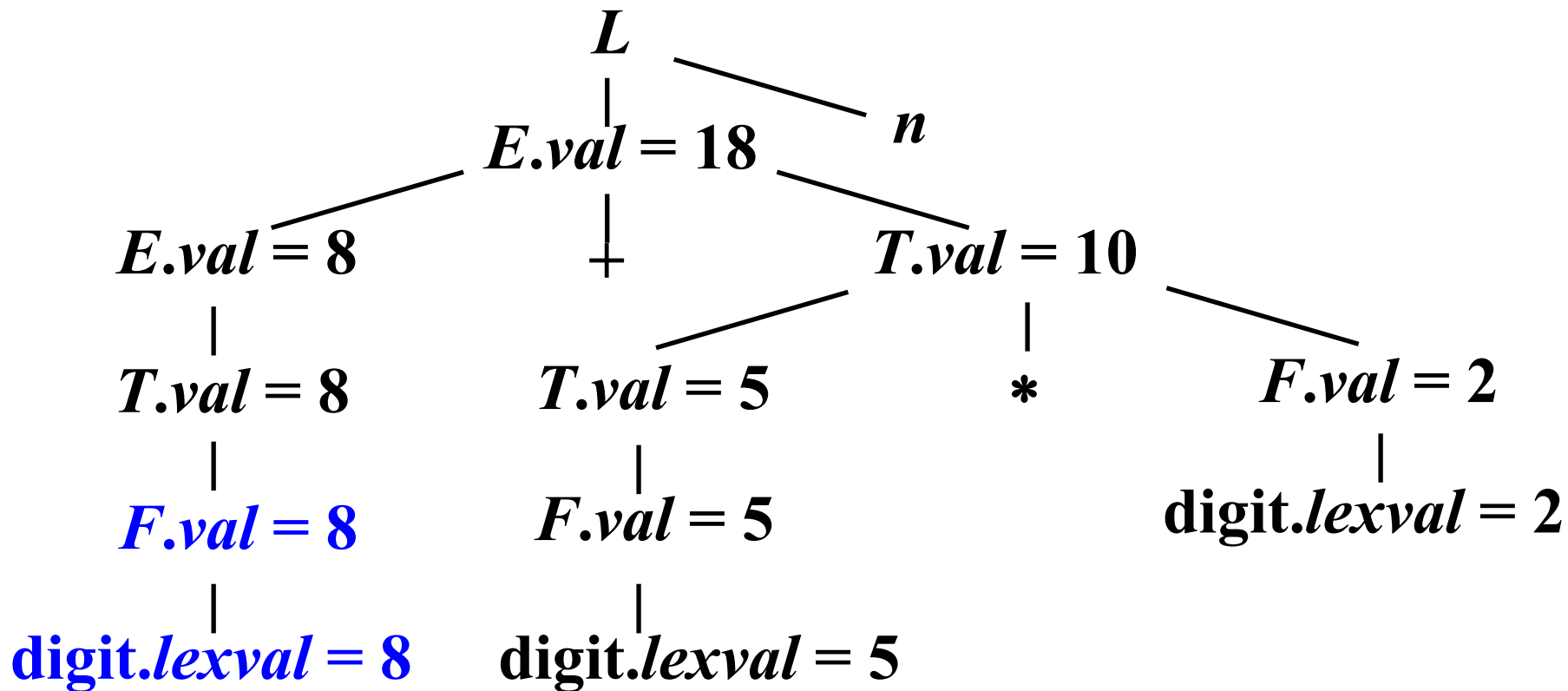


□各结点综合属性的计算可以自底向上地完成



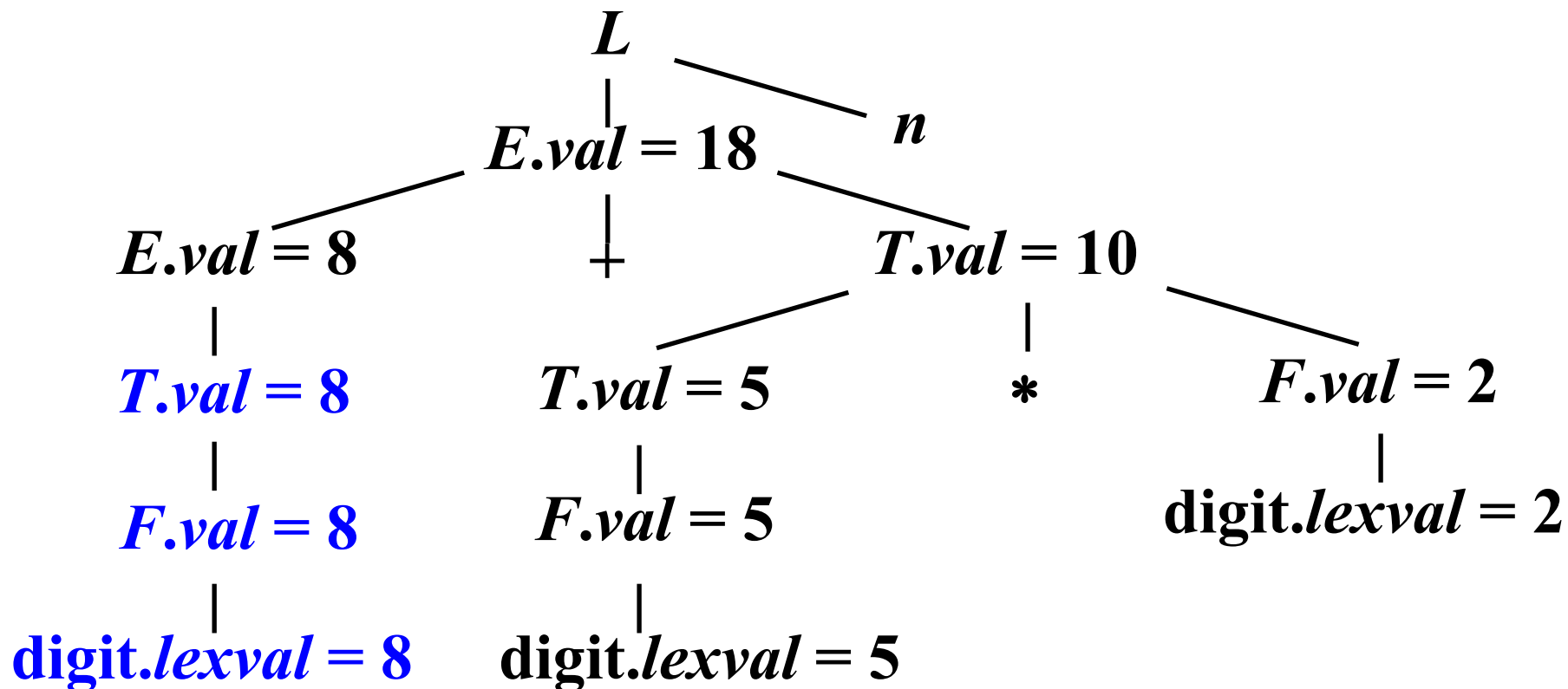


□ 各结点综合属性的计算可以自底向上地完成



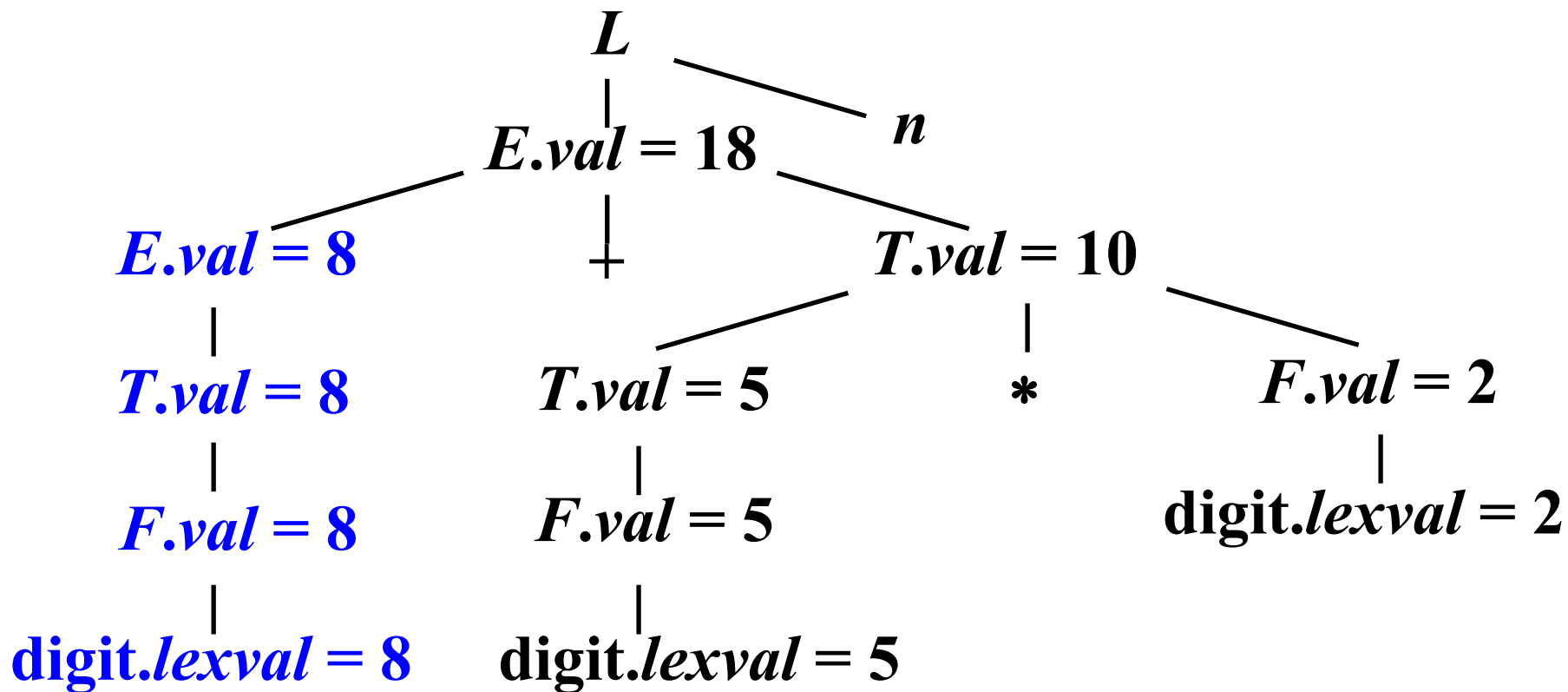


□ 各结点综合属性的计算可以自底向上地完成



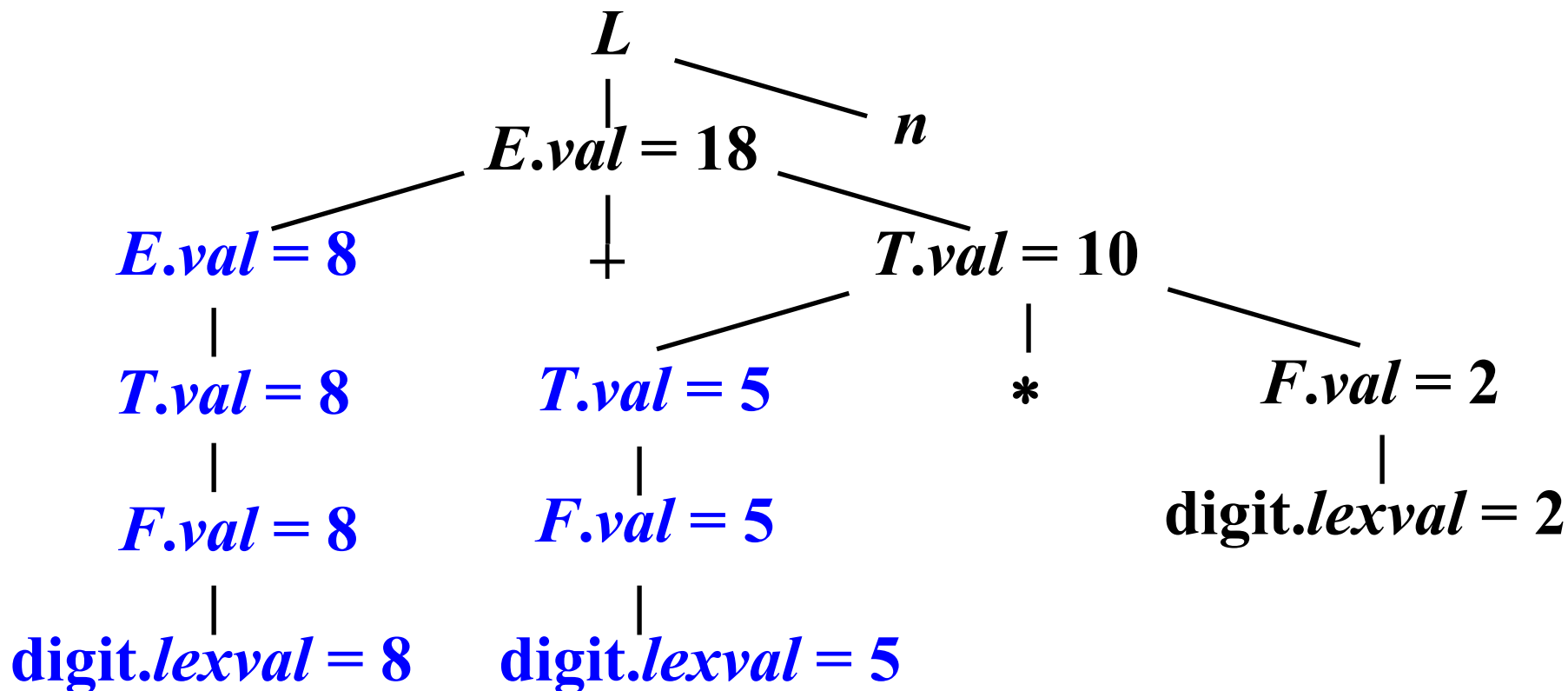


□各结点综合属性的计算可以自底向上地完成



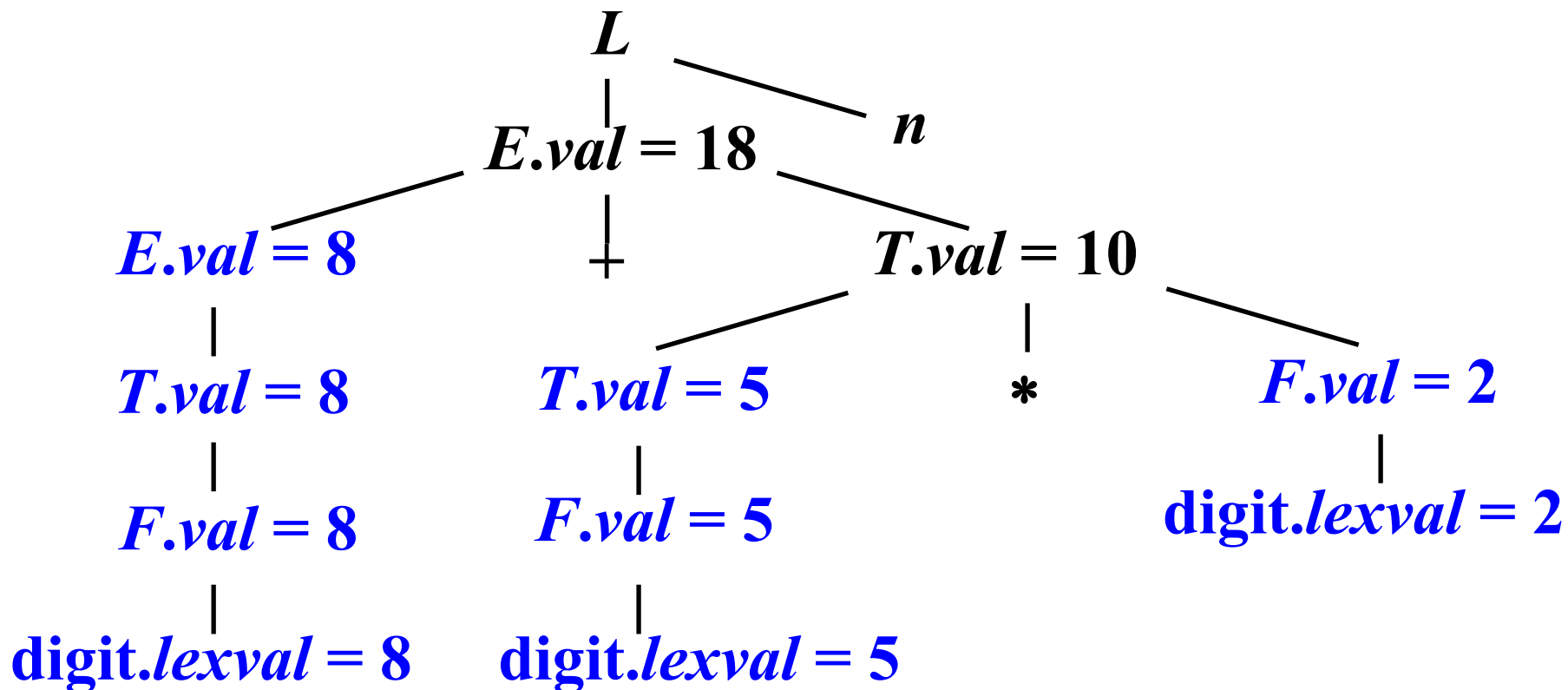


□ 各结点综合属性的计算可以自底向上地完成



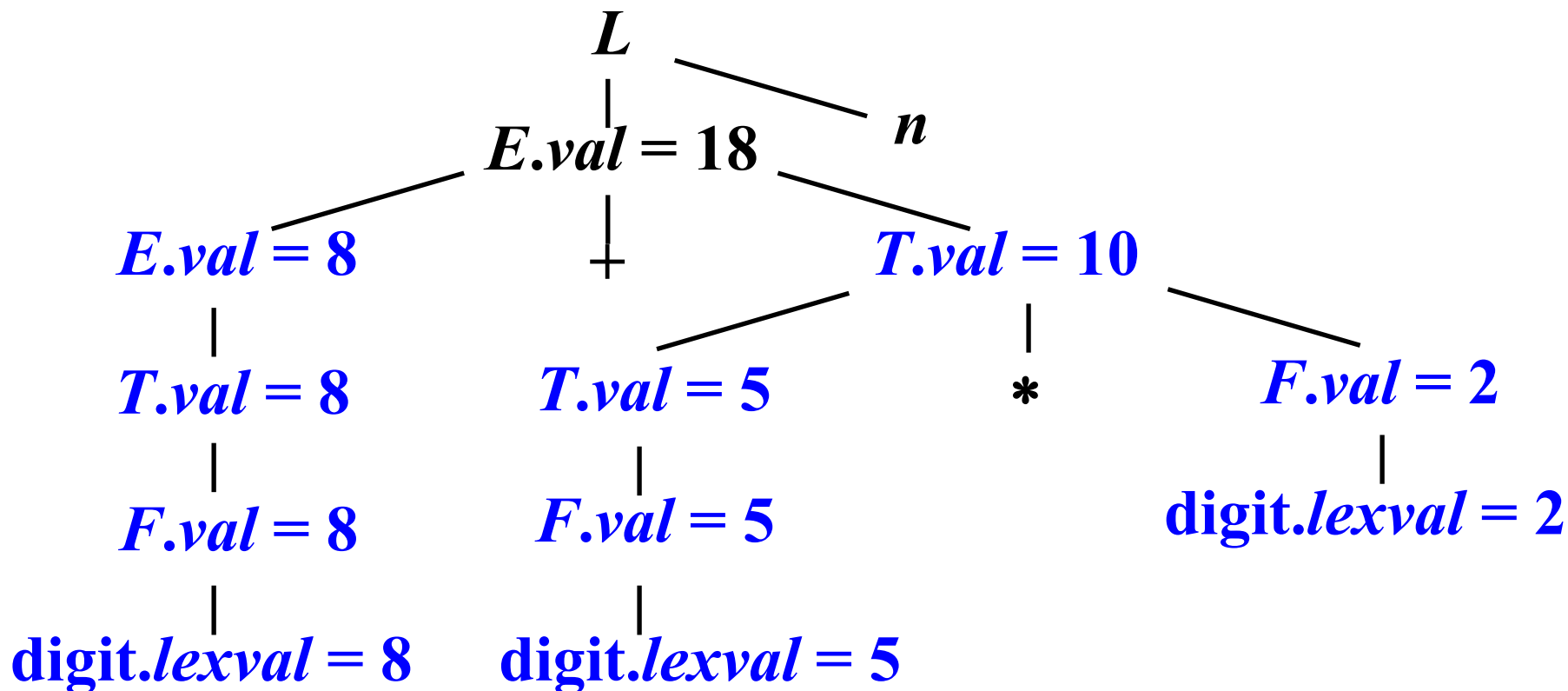


□ 各结点综合属性的计算可以自底向上地完成





□ 各结点综合属性的计算可以自底向上地完成

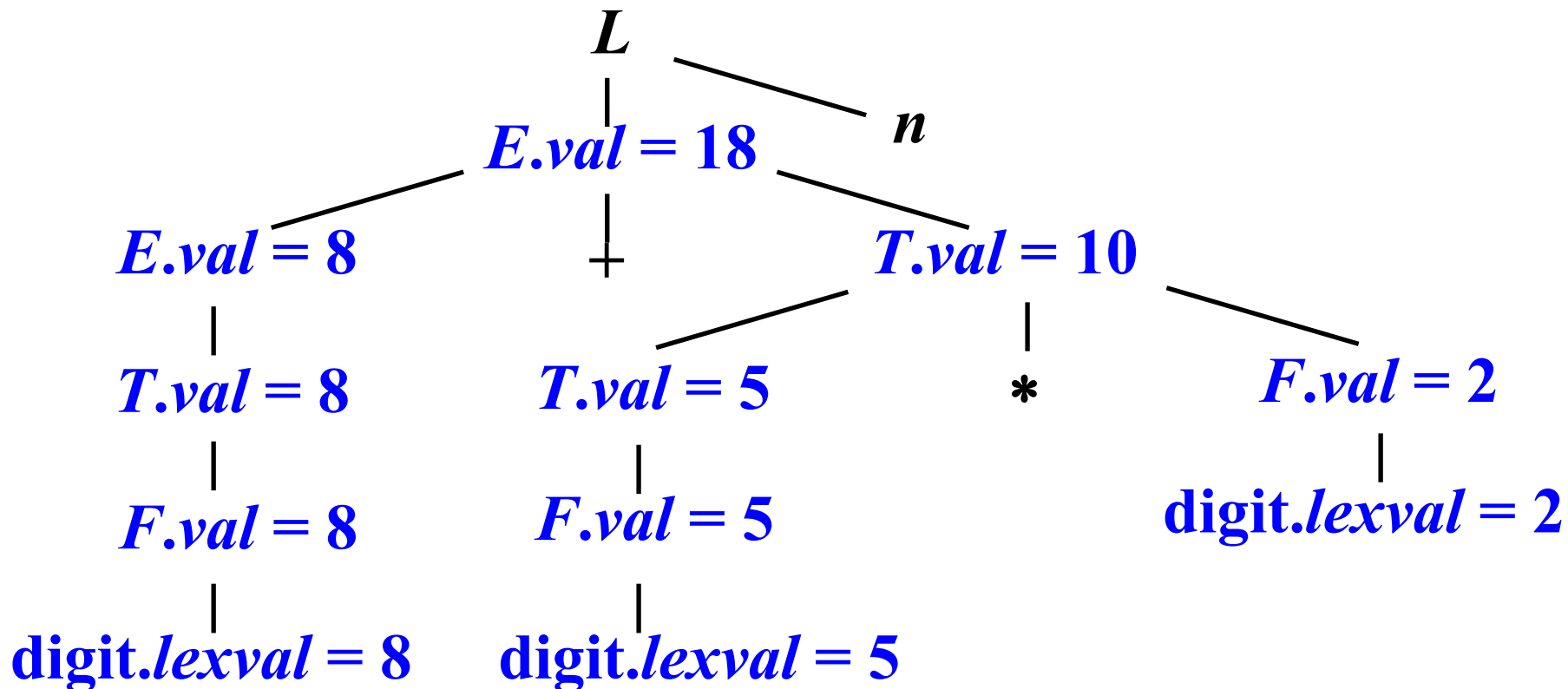






□ 各结点综合属性的计算可以自底向上地完成

综合属性的计算可以和LR分析器一起自然地实现。





# 例1



□ 已知如下文法:

$E \rightarrow E - T \mid T$

$T \rightarrow \text{num} \mid \text{num.num}$

❖ 写一个语法制导定义，来确定减法表达式的类型



□已知如下文法:

$E \rightarrow E - T \mid T$

$T \rightarrow \text{num} \mid \text{num.num}$

✧写一个语法制导定义，来确定减法表达式的类型

□设E和T有综合属性type，num的综合属性为integer，  
num.num的综合属性为float

产生式	语义规则
$E \rightarrow E_1 - T$	$E.type = \text{if } (E_1.type == T.type) \text{ } T.type$ $\text{else float}$
$E \rightarrow T$	$E.type = T.type$
$T \rightarrow \text{num}$	$T.type = \text{integer}$
$T \rightarrow \text{num.num}$	$T.type = \text{float}$



## □考虑消除左递归的算术表达式文法

产生式	语义规则
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

消除左递归

产生式
$T \rightarrow FT'$
$T' \rightarrow *FT'_1$
$T' \rightarrow \varepsilon$
$F \rightarrow \text{digit}$

## □思考是否可以直接依赖综合属性val来计算算术表达式的值?



## □考虑消除左递归的算术表达式文法

产生式	语义规则
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

消除左递归

产生式
$T \rightarrow FT'$
$T' \rightarrow *FT'_1$
$T' \rightarrow \varepsilon$
$F \rightarrow \text{digit}$

□T对应的项中，第一个运算分量是F，而运算符和第二个运算分量在T'中



## □语法制导定义 (Syntax-Directed Definition, SDD)

- ❖基础的上下文无关文法
- ❖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 $b=f(c_1, c_2, \dots, c_k)$ 的语义规则，其中 $f$ 是函数 $b$ 和 $c_1, c_2, \dots, c_k$ 是该产生式文法符号的属性
- ❖综合属性(synthesized attribute): 如果 $b$ 是 $A$ 的属性， $c_1, c_2, \dots, c_k$ 是产生式右部文法符号的属性或 $A$ 的其它属性
- ❖继承属性(inherited attribute): 如果 $b$ 是右部某文法符号 $X$ 的属性， $c_1, c_2, \dots, c_k$ 是 $A$ 和产生式右部文法符号的属性



□考虑消除左递归的算术表达式文法

□为 $T'$ 引入继承属性 $inh$

❖该属性继承了对应的\*号的左运算分量

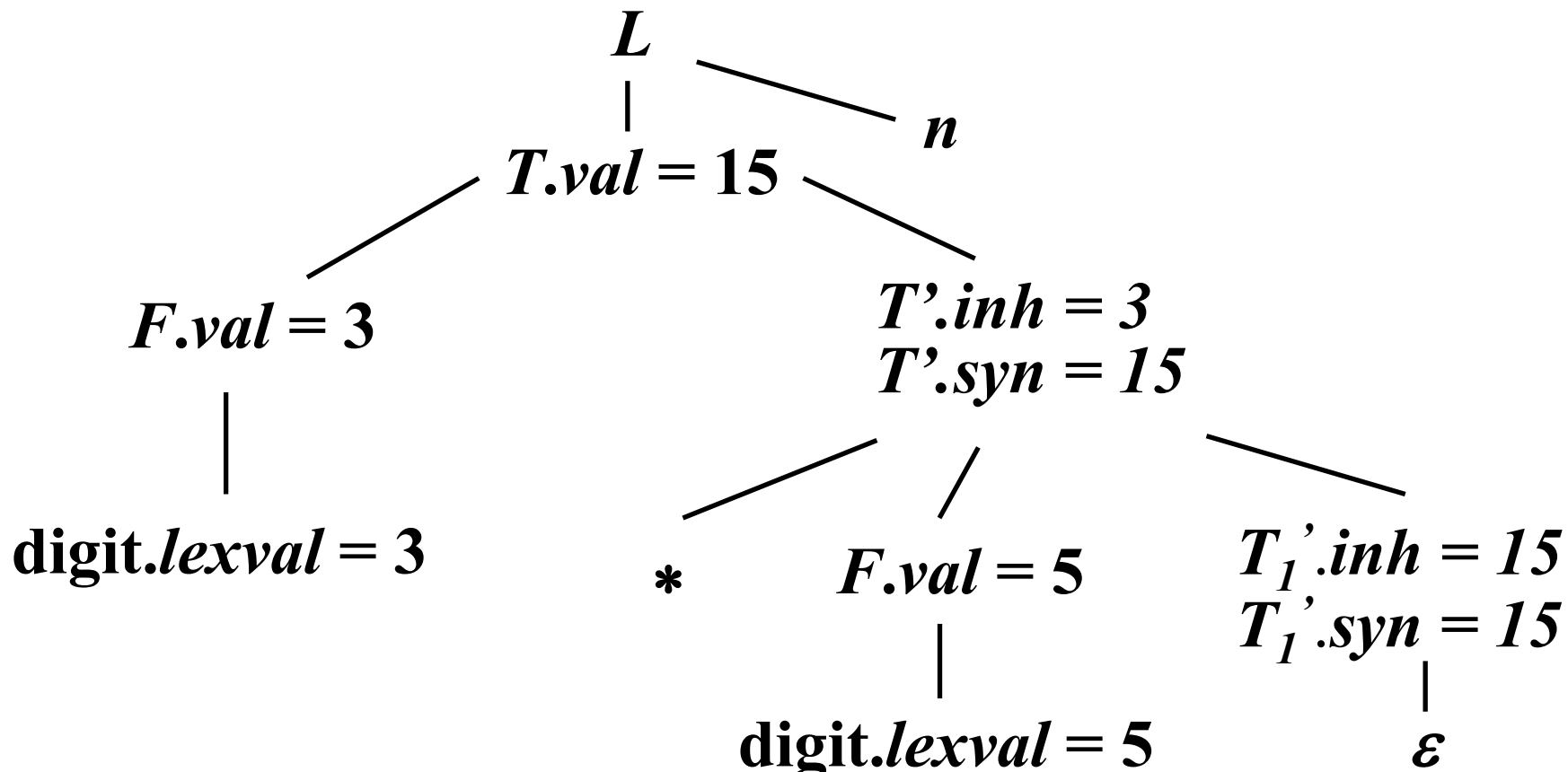
产生式	语义规则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

红色代表  
通过计算  
分量的值  
传递开来

蓝色代表  
最终结果  
返回



## □3\*5的注释分析树

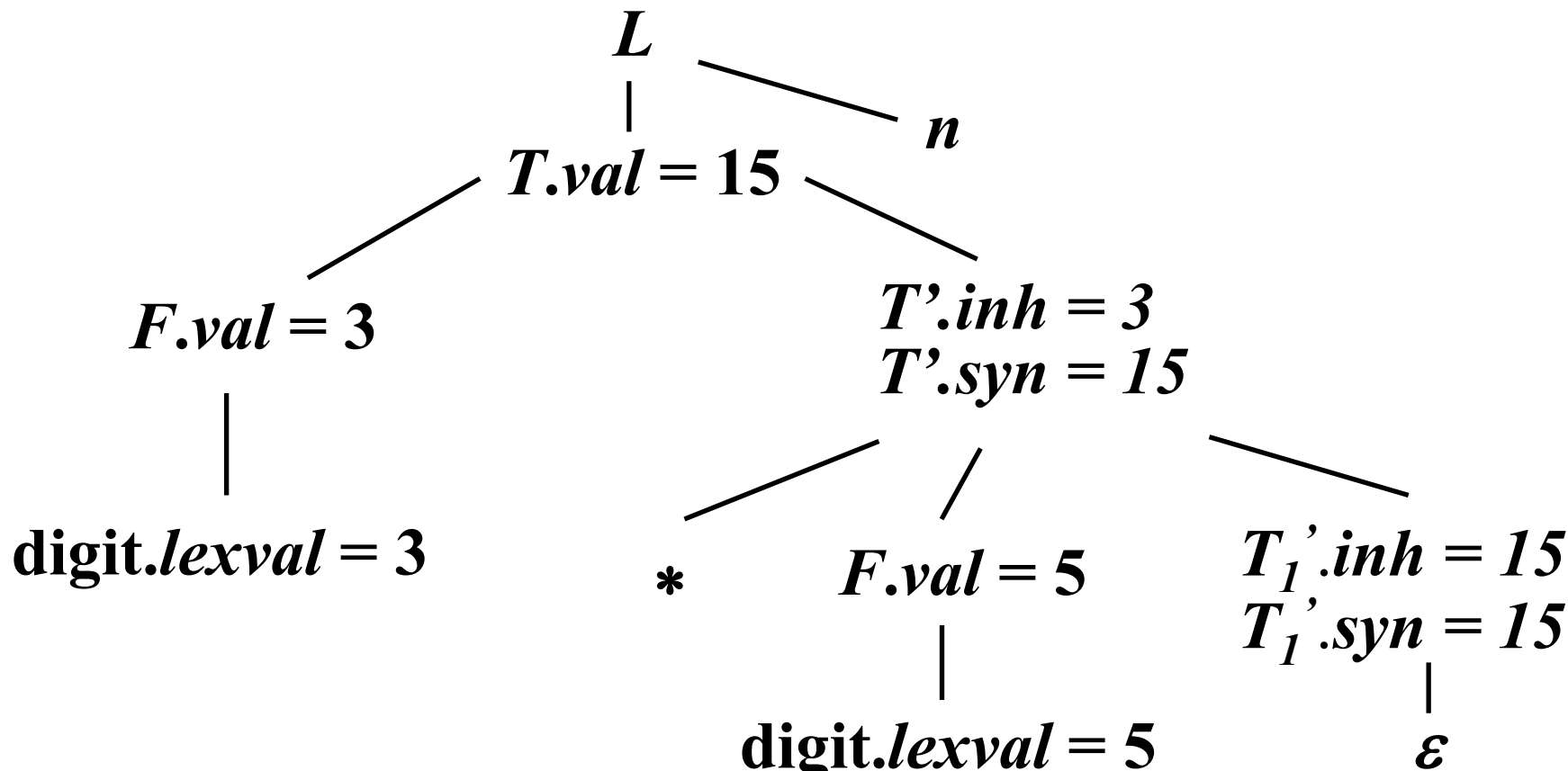






## □ $3*5$ 的注释分析树

□ 显然自底向上的计算方式是不合适的





## □ SDD为CFG中的文法符号设置语义属性。

❖ 对于给定的输入串 $x$ ，应用**语义规则**计算分析树中各结点对应的属性值

## □ 按照什么顺序计算属性值？

❖ 语义规则建立了属性之间的依赖关系，在对语法分析树节点的一个属性求值之前，必须首先求出这个属性值所依赖的所有属性值

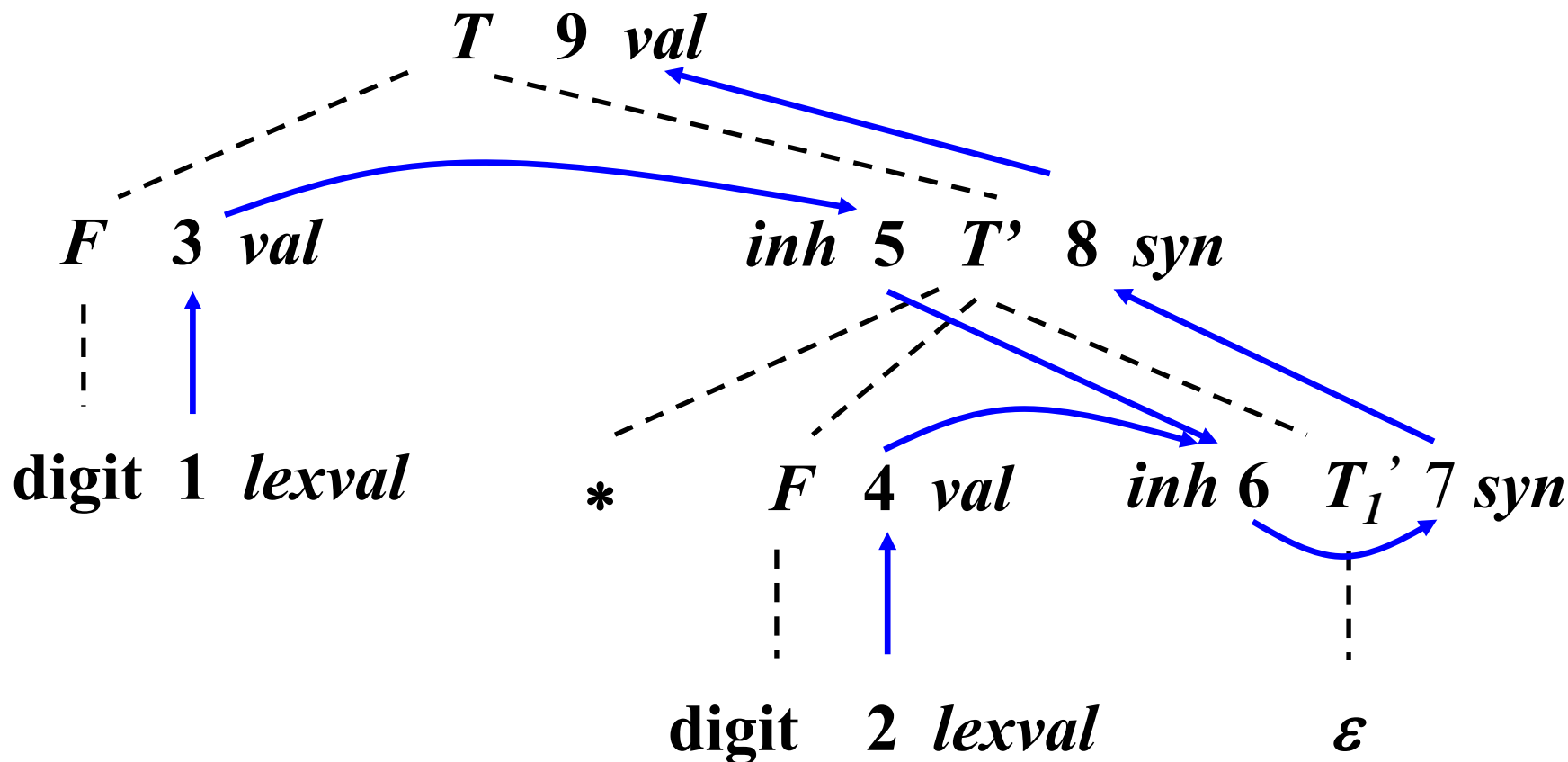


## □ 依赖图(dependency graph)是一个描述了分析树中结点属性间依赖关系的有向图

- ❖ 属性值为点(vertex): 分析树中每个标号为 $X$ 的结点的每个属性 $a$ 都对应着依赖图中的一个结点
- ❖ 属性依赖关系为边(edge): 如果属性 $X.a$ 的值依赖于属性 $Y.b$ 的值, 则依赖图中有一条从 $Y.b$ 的结点指向 $X.a$ 的结点的有向边



## □以消除左递归的算术表达式文法和 $3*5$ 为例





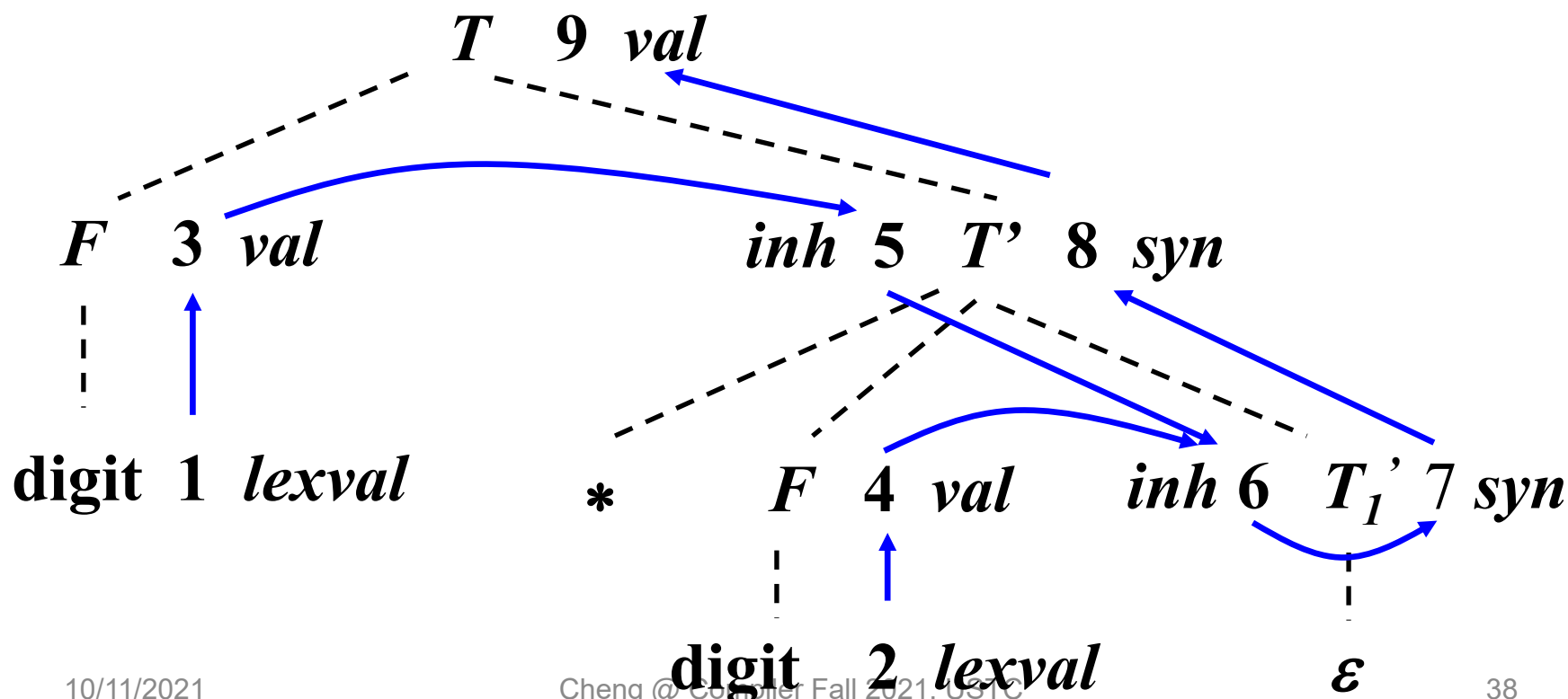
## □可行的求值顺序是满足下列条件的结点序列

$N_1, N_2, \dots, N_k$ :

- ❖如果依赖图中有一条从结点 $N_i$ 到 $N_j$ 的边( $N_i \rightarrow N_j$ ), 那么, 在节点序列中,  $N_i$ 排在 $N_j$ 前面
- ❖该排序称为这个图的**拓扑排序(topological sort)**



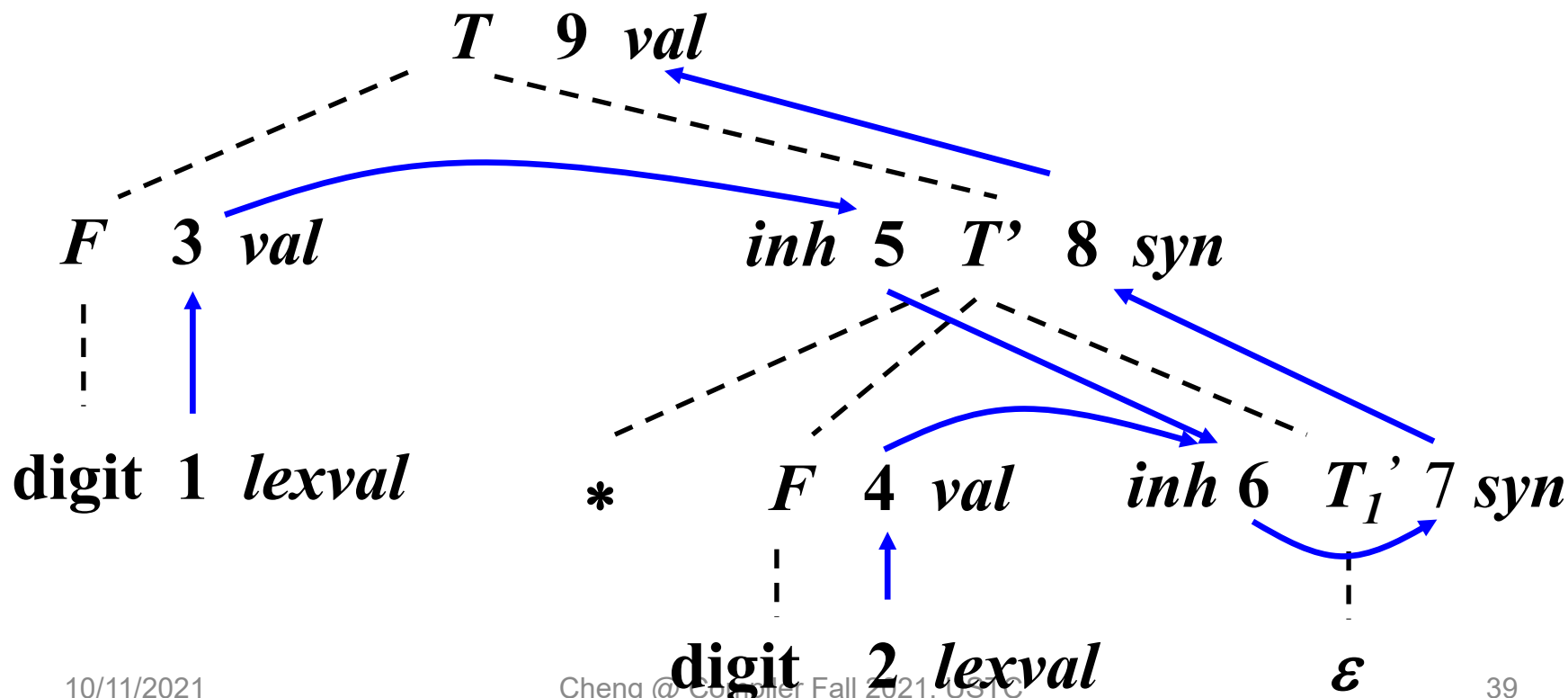
- 构造输入的分析树，构造属性依赖图，对结点进行拓扑排序，按拓扑排序的次序计算属性





□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

❖可行排序一：1, 2, 3, 4, 5, 6, 7, 8, 9

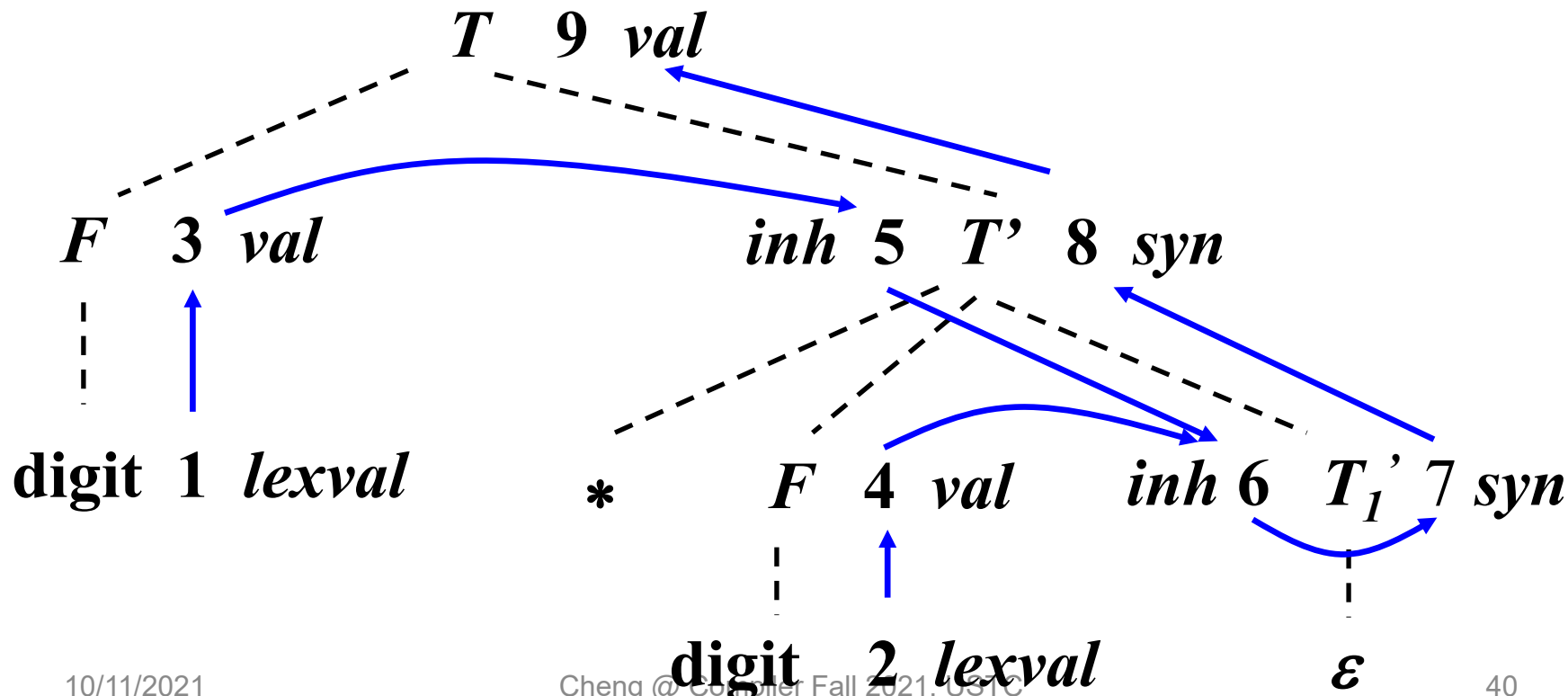




□构造输入的分析树，构造属性依赖图，对结点进行拓扑排序，按拓扑排序的次序计算属性

❖可行排序一：1, 2, 3, 4, 5, 6, 7, 8, 9

❖可行排序二：2, 4, 1, 3, 5, 6, 7, 8, 9







□依赖于拓扑排序

□思考：在有向图中，什么时候拓扑排序不存在？



□依赖于拓扑排序

□思考：在有向图中，什么时候拓扑排序不存在？

❖当图中出现环的时候

❖SDD的属性之间存在循环依赖关系



□依赖于拓扑排序

□思考：在有向图中，什么时候拓扑排序不存在？

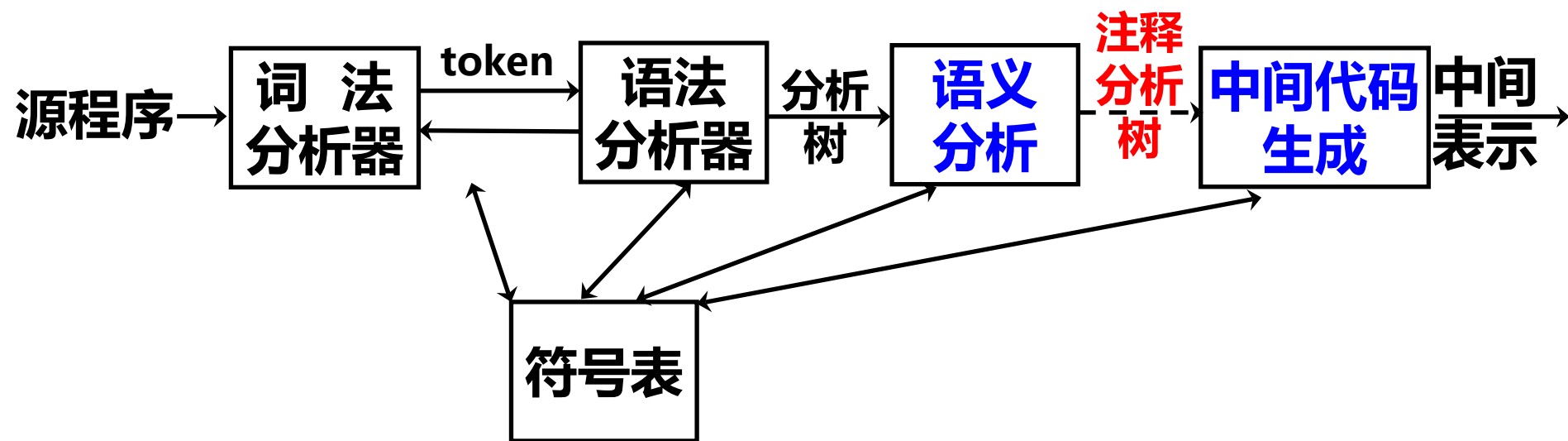
❖当图中出现环的时候

❖SDD的属性之间存在循环依赖关系

□解决方案：

❖使用某些特定类型的依赖图不存在环的SDD

➤S属性的SDD和L属性的SDD



## □语法制导翻译简介

## □语法制导定义

- ❖ 属性、属性、依赖图、计算次序
- ❖ S属性的定义、L属性的定义
- ❖ 语法指导定义的应用



## □ 仅仅使用综合属性的语法制导定义称为S属性的SDD，或S-属性定义、S-SDD

- ❖ 如果一个SDD是S属性的，可以按照语法分析树节点的任何自底向上顺序来计算它的各个属性值
- ❖ S-属性定义可在**自底向上**的语法分析过程中实现
  - 例如：**LR分析器**



□ 仅仅使用综合属性的语法制导定义称为S属性的SDD，或S-属性定义、S-SDD

产生式	语义规则
$L \rightarrow E \mathbf{n}$	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val = \mathbf{E_1}.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

□下面是产生字母表 $\Sigma = \{0, 1, 2\}$ 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

□下面是产生字母表 $\Sigma = \{0, 1, 2\}$ 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 = (D1.val == D2.val) and S1.val`

$S \rightarrow 2$

`S.val = true`

$D \rightarrow 0$

`D.val = 0`

$D \rightarrow 1$

`D.val = 1`



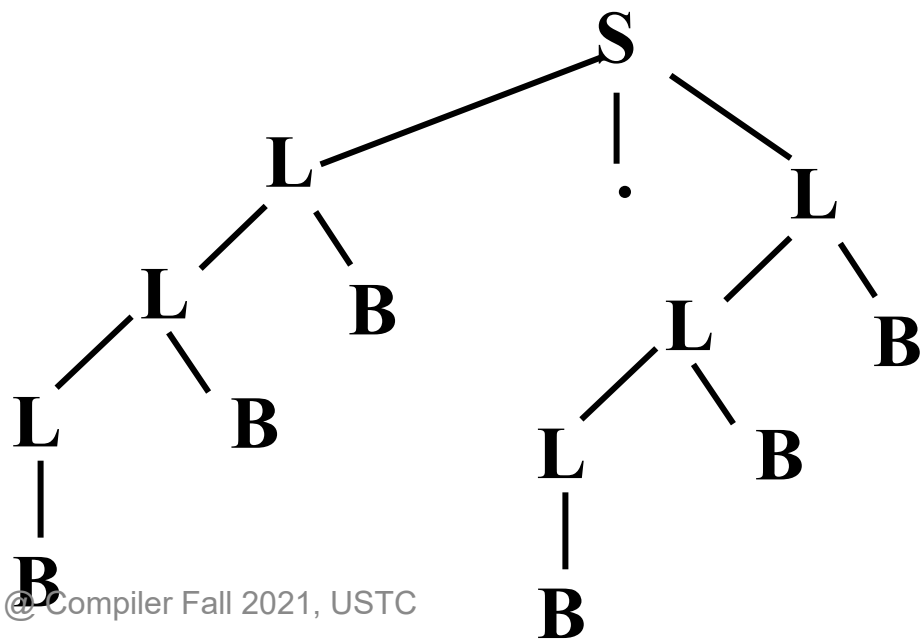
□为下面文法写一个语法制导的定义，用S的综合属性 $val$ 给出下面文法中S产生的二进制数的值。例如，输入101.101时， $S.val = 5.625$ （可以修改文法）

若按 $2^2 + 0 + 2^0 + 2^{-1} + 0 + 2^{-3}$ 来计算，该文法对小数点左边部分的计算不利，因为需要继承属性来确定每个B离开小数点的距离

$S \rightarrow L . L \mid L$

$L \rightarrow L B \mid B$

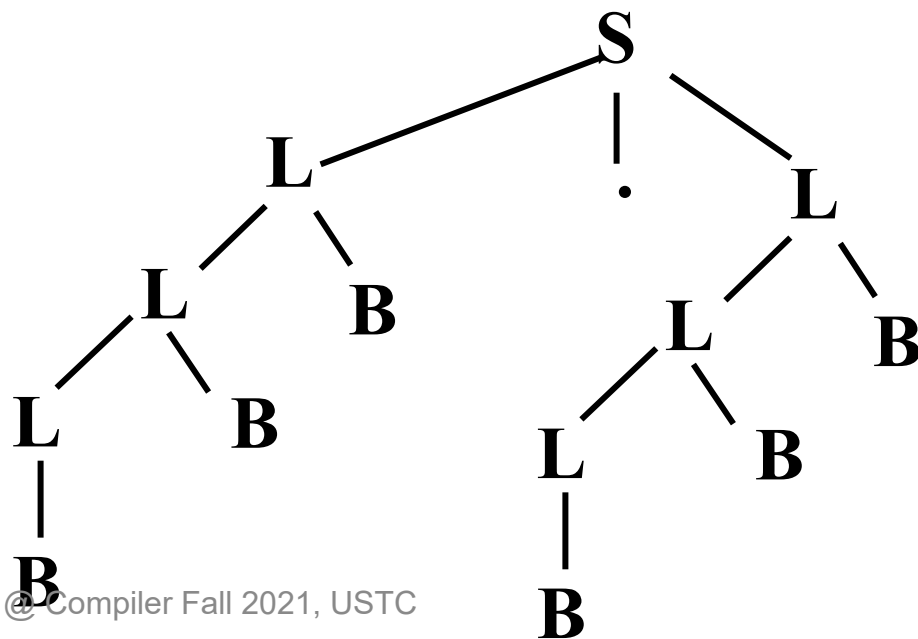
$B \rightarrow 0 \mid 1$





**若小数点左边按 $(1 \times 2 + 0) \times 2 + 1$ 计算。该办法不能直接用于小数点右边，需改成 $((1 \times 2 + 0) \times 2 + 1)/2^3$ ，这时需要综合属性来统计B的个数**

**B → 0 | 1**



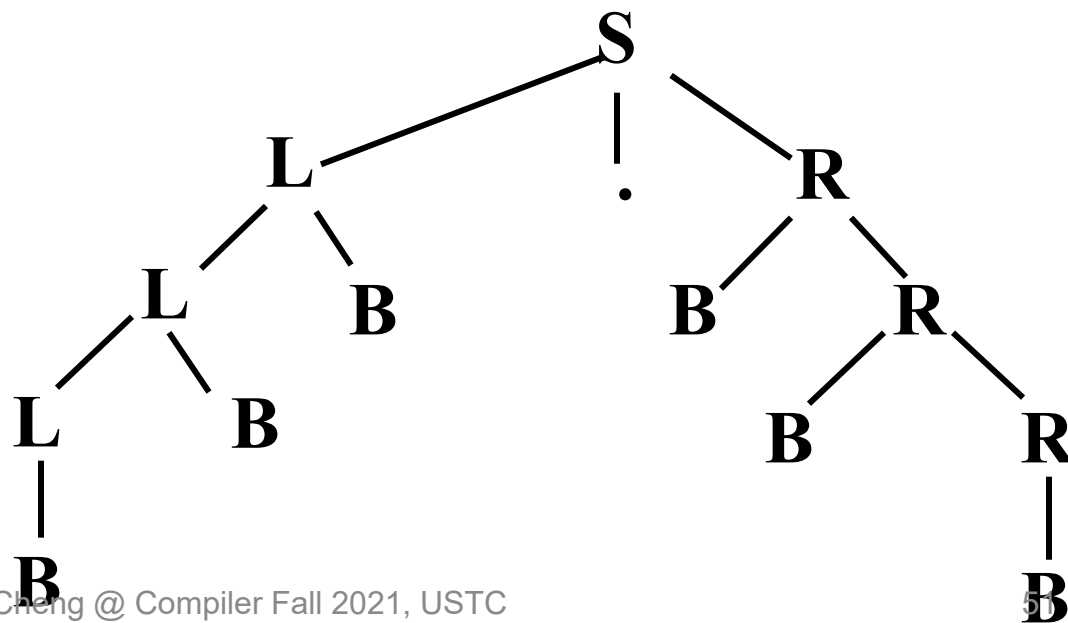
□为下面文法写一个语法制导的定义，用S的综合属性 $val$ 给出下面文法中S产生的二进制数的值。例如，输入101.101时， $S.val = 5.625$ （可以修改文法）

更清楚的办法是将文法改成下面的形式

$$S \rightarrow L . R \mid L$$

$$L \rightarrow L B \mid B$$

$$R \rightarrow B R \mid B$$

$$B \rightarrow 0 \mid 1$$


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



## □L-属性定义(也称为L属性的SDD或L-SDD)的 直观含义:

- ❖ 在一个产生式所关联的各属性之间，依赖图的边  
可以从左到右，但不能从右到左(因此称为L属性的，L是Left的首字母)
- ❖ 可以在LR分析器或者LL分析器中实现
- ❖ 更加一般化



□任意产生式 $A \rightarrow X_1 X_2 \dots X_n$ ，其右部符号 $X_i (1 \leq i \leq n)$ 的继承属性仅依赖于下列属性：

❖  $A$ 的继承属性

➤如果依赖 $A$ 的综合属性，由于 $A$ 的综合属性可能依赖 $X_i$ 的属性，包括 $X_i$ 的综合属性和继承属性，因此可能形成环路

❖产生式中 $X_i$ 左边的符号 $X_1, X_2, \dots, X_{i-1}$ 的属性

❖ $X_i$ 本身的属性，但 $X_i$ 的全部属性不能在依赖图中形成环路



## 例： $L$ -SDD

	产生式	语义规则
(1)	$T \rightarrow F T'$	$\underline{T'.inh} = F.val$ $\underline{T.val} = T'.syn$
(2)	$T' \rightarrow * F T_1'$	$\underline{T_1'.inh} = T'.inh \times F.val$ $\underline{T'.syn} = T_1'.syn$
(3)	$T' \rightarrow \varepsilon$	$\underline{T'.syn} = T'.inh$
(4)	$F \rightarrow \text{digit}$	$\underline{F.val} = \text{digit.lexval}$

继承属性

综合属性



## 非L属性的SDD

➤ 例

产生式	语义规则
(1) $A \rightarrow LM$	$L.i = l(A.i)$ $M.i = m(L.s)$ $A.s = f(M.s)$
(2) $A \rightarrow QR$	$R.i = r(A.i)$ $Q.i = q(R.s)$ $A.s = f(Q.s)$

继承属性

综合属性





# L属性的定义：举例



**int id, id, id  
标识符声明**

产生式	语义规则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow \text{int}$	$T.type = \text{integer}$
$T \rightarrow \text{real}$	$T.type = \text{real}$
$L \rightarrow L_1, \text{id}$	$L_1.in = L.in;$ $addType(id.entry, L.in)$
$L \rightarrow \text{id}$	$addType(id.entry, L.in)$

□  $type$  –  $T$ 的综合属性

□  $in$  –  $L$ 的继承属性，把声明的类型传递给标识符列表

□  $addType$  – 把类型加到符号表中的标识符条目里(副作用)



## □ 一个没有副作用的SDD称为属性文法

- ❖ 属性文法增加了语义规则描述的复杂度
- ❖ 如：符号表必须作为属性传递
- ❖ 为了简单起见，我们可以把符号表作为全局变量，通过副作用函数读取或者添加标识符



## □ 一个没有副作用的SDD称为属性文法

- ❖ 属性文法增加了语义规则描述的复杂度
- ❖ 如：符号表必须作为属性传递
- ❖ 为了简单起见，我们可以把符号表作为全局变量，通过副作用函数读取或者添加标识符

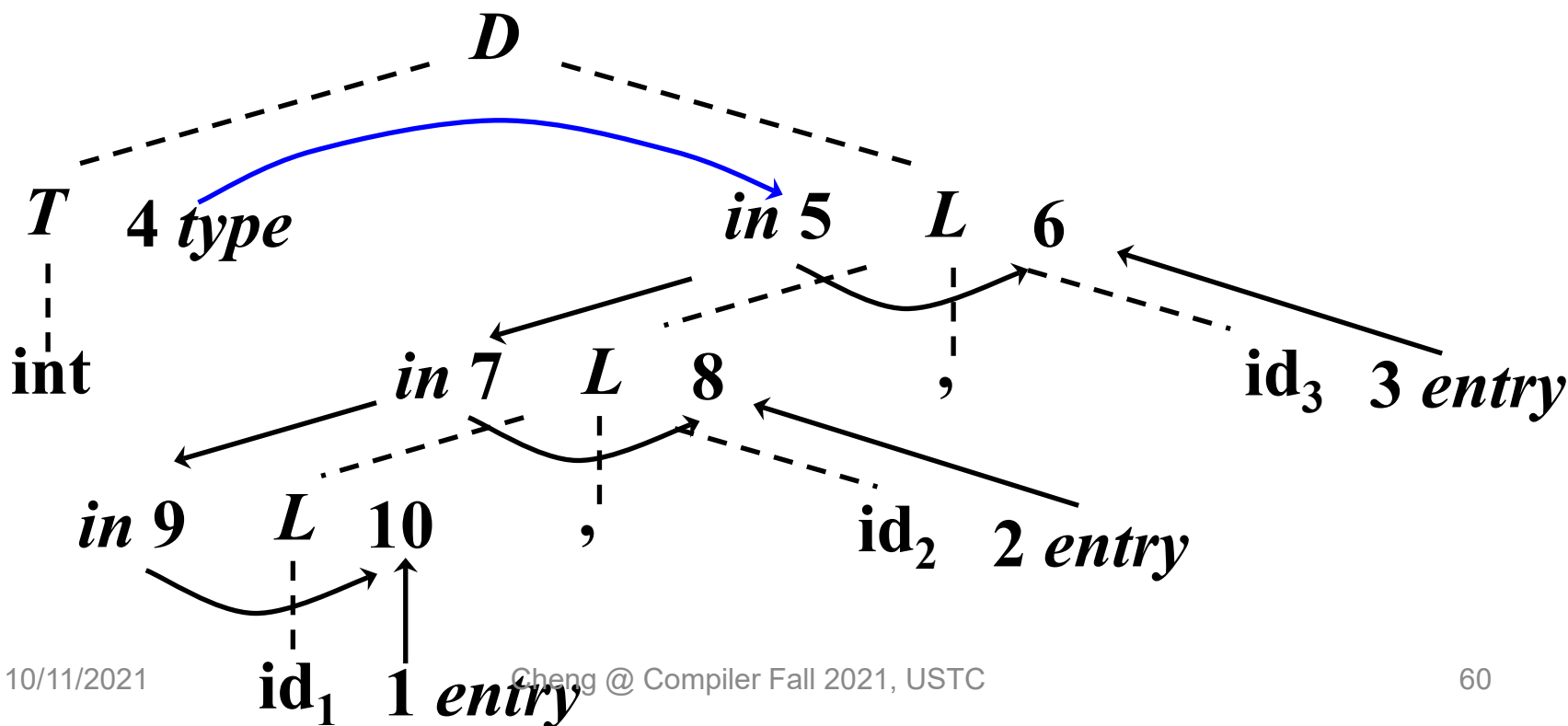
## □ 受控的副作用

- ❖ 不会对属性求值产生约束，即可以按照任何拓扑顺序求值，不会影响最终结果
- ❖ 或者对求值过程添加约束



□例 int id<sub>1</sub>, id<sub>2</sub>, id<sub>3</sub>的分析树（虚线）的依赖图（实线）

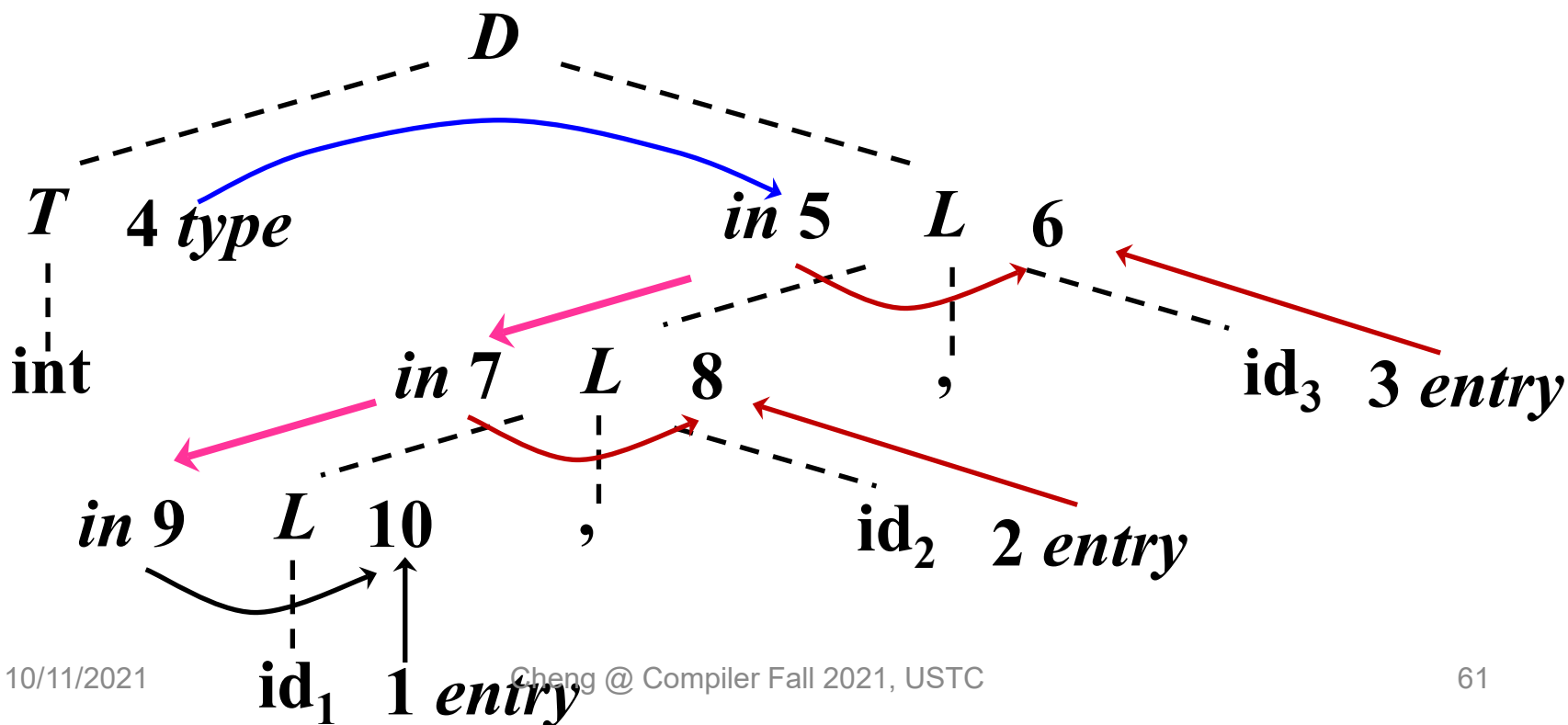
$$D \rightarrow TL \quad L.in = T.type$$





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

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

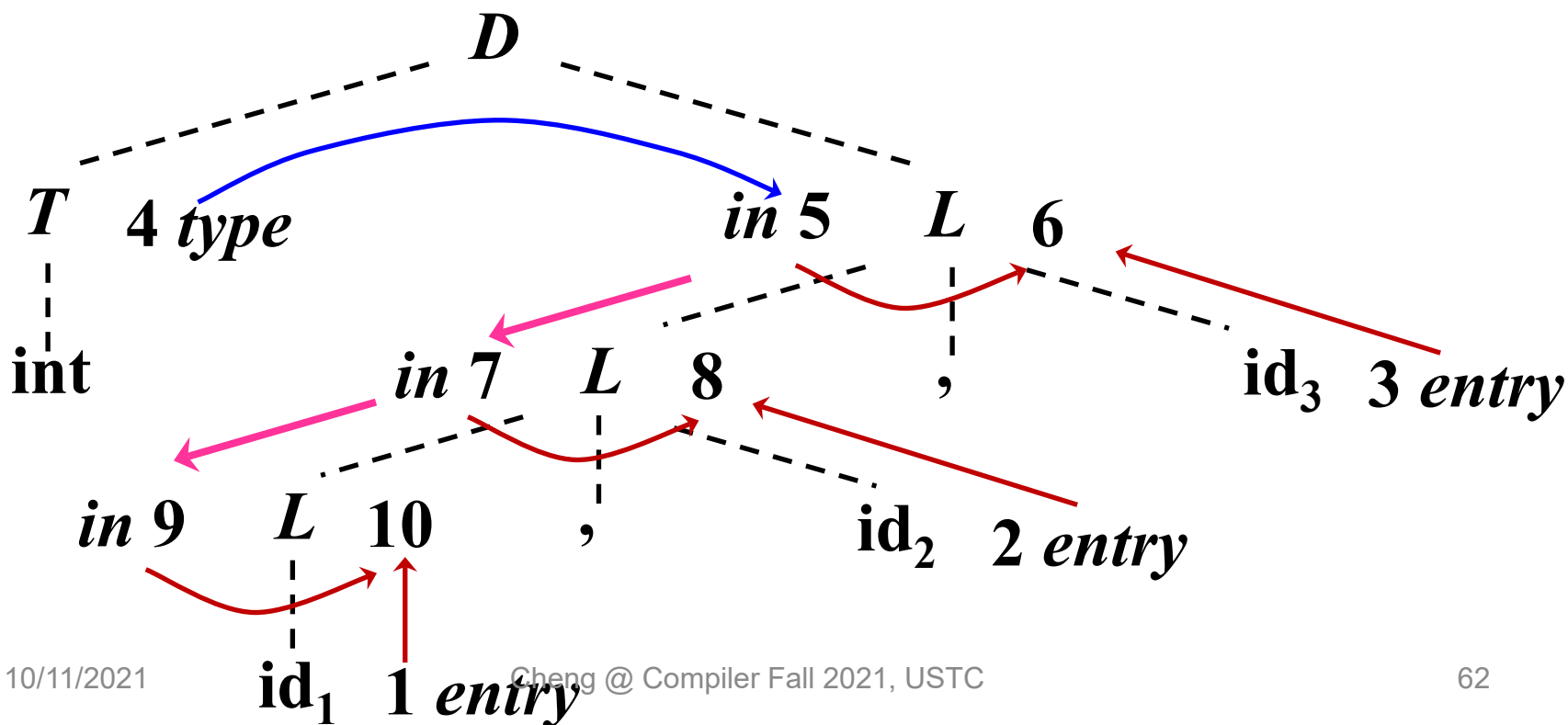


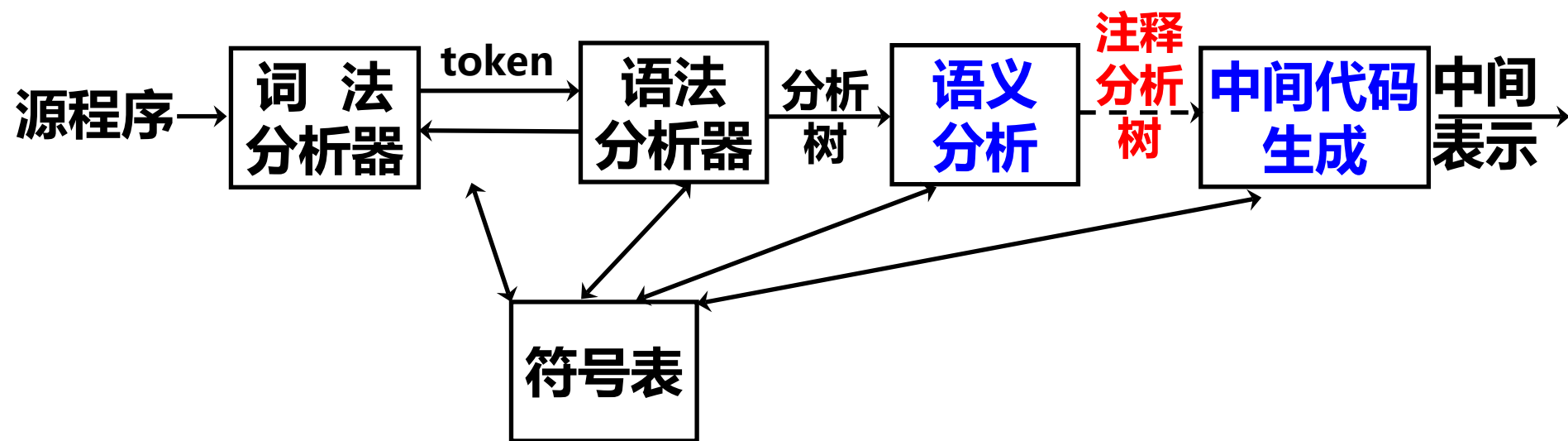


□例 int  $id_1, id_2, id_3$ 的分析树（虚线）的依赖图（实线）

$L \rightarrow id$

$addType(id.entry, L.in)$





## □语法制导翻译简介

## □语法制导定义

- ❖ 属性、属性依赖图、计算次序
- ❖ S属性的定义、L属性的定义
- ❖ 语法制导定义的应用



## □ 抽象语法树的构造

❖ S属性定义的方法

❖ L属性定义的方法

## □ 类型检查(下一章)

## □ 中间代码生成(下一章)

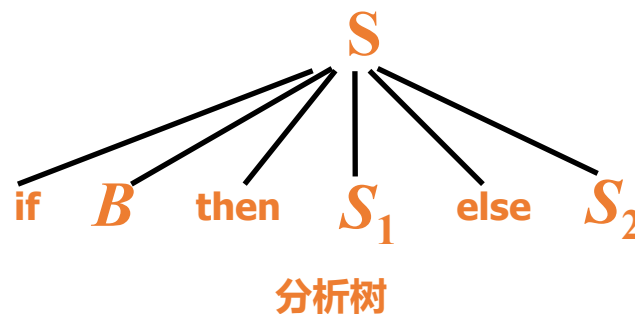
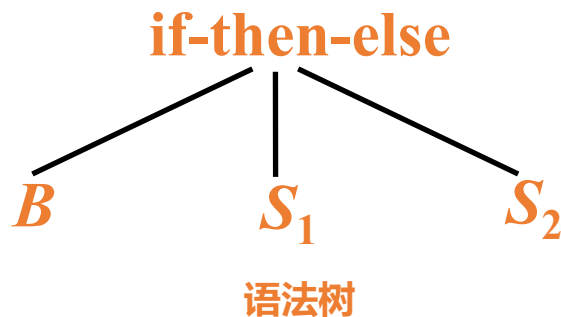




## □抽象语法树(Abstract syntax tree, AST)

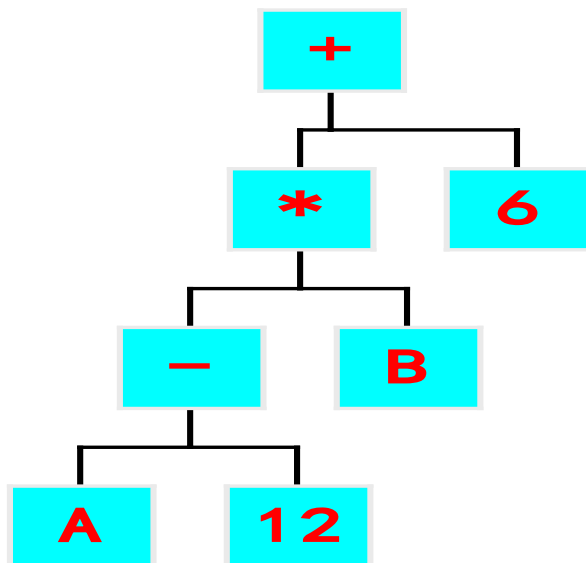
- ❖ 简称语法树，是分析树的浓缩表示：**算符和关键字**是作为内部结点。
- ❖ 语法制导翻译可基于分析树，也可基于语法树

$S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$





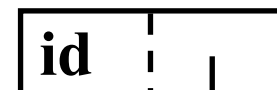
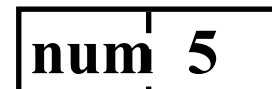
□例：表达式  $(A - 12) * B + 6$  的语法树。





## □对基本运算对象结点

- ❖ 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）

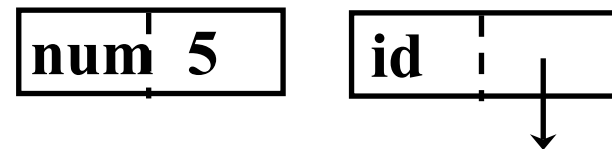


指向符号表中id的条目



## □对基本运算对象结点

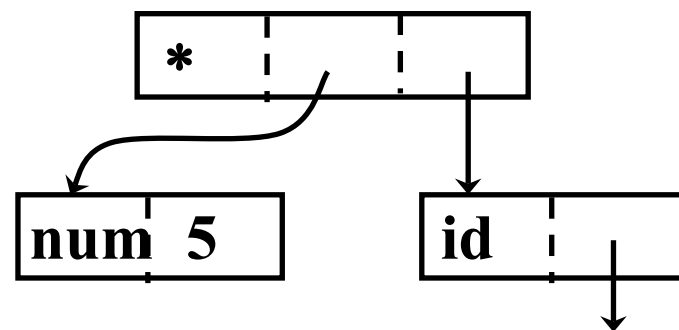
- ❖ 一个域存放运算对象类别
- ❖ 另一个域存放其值（也可用其他域保存其他属性或者指向该属性值的指针）



指向符号表中id的条目

## □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。





## □mknode (op, left, right)

❖ 建立一个运算符结点，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

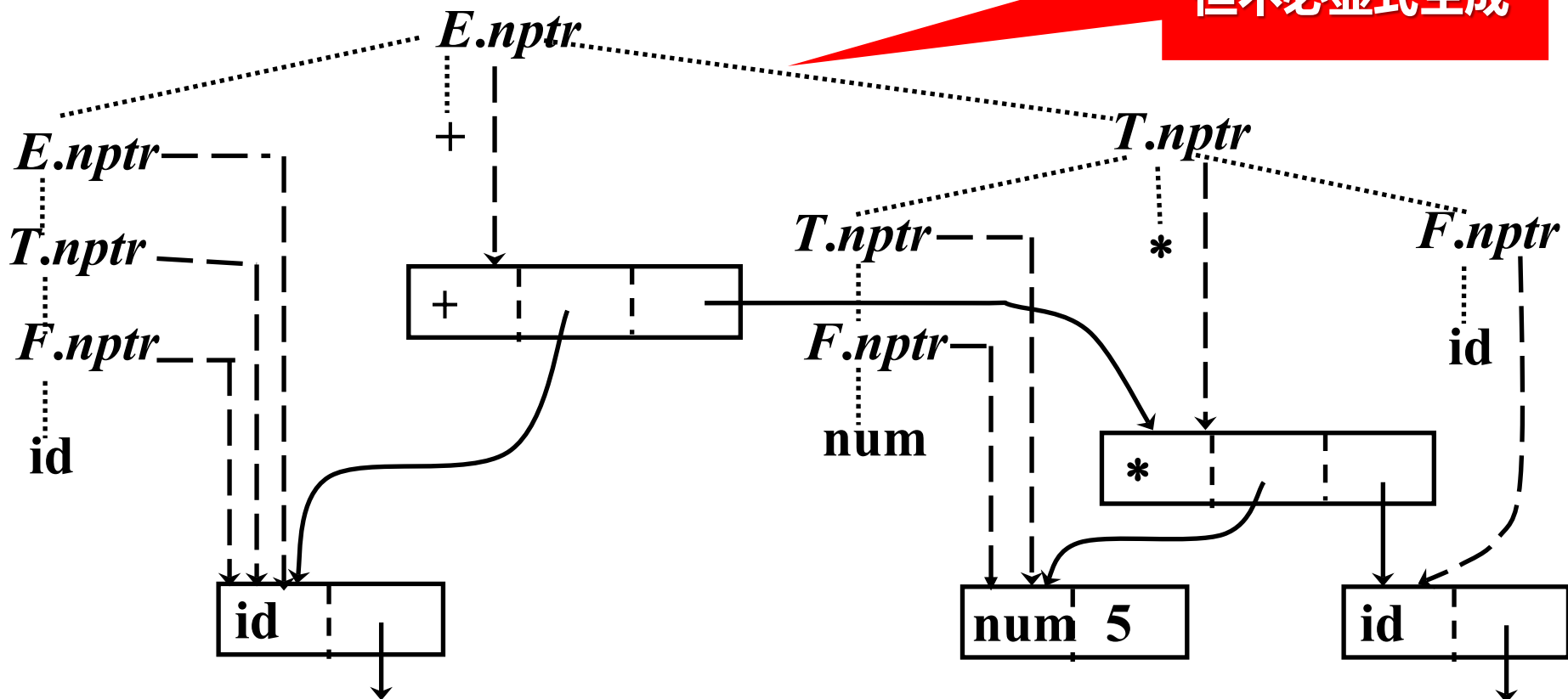
❖ *nptr*综合属性：文法符号对应的抽象语法树结点

产生式	语义规则
$E \rightarrow E_1 + T$	$E.nptr = mkNode( '+', E_1.nptr, T.nptr )$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode( '*', T_1.nptr, F.nptr )$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow id$	$F.nptr = mkLeaf( id, id.entry )$
$F \rightarrow num$	$F.nptr = mkLeaf( num, num.val )$



## $a+5*b$ 的语法树的构造

不带箭头的虚线  
代表语法分析树,  
但不必显式生成

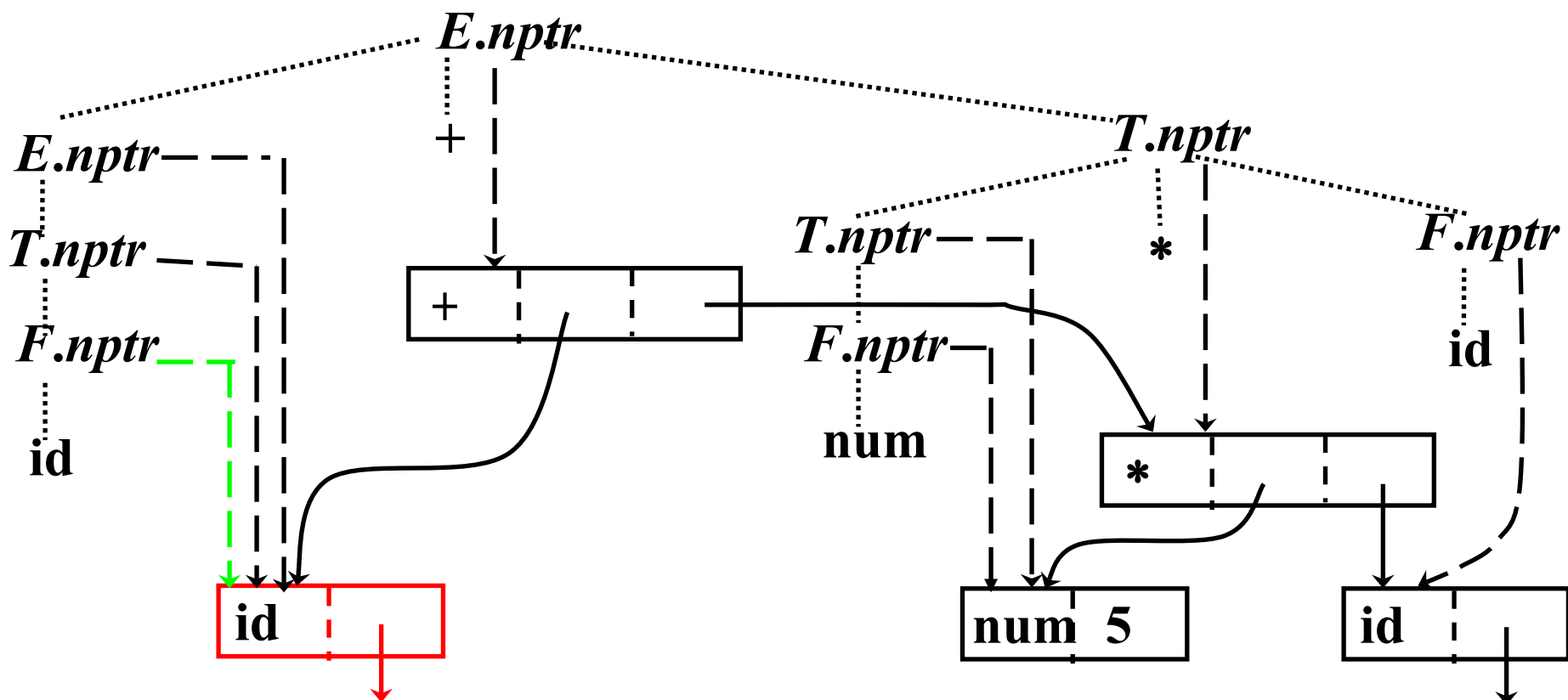


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造



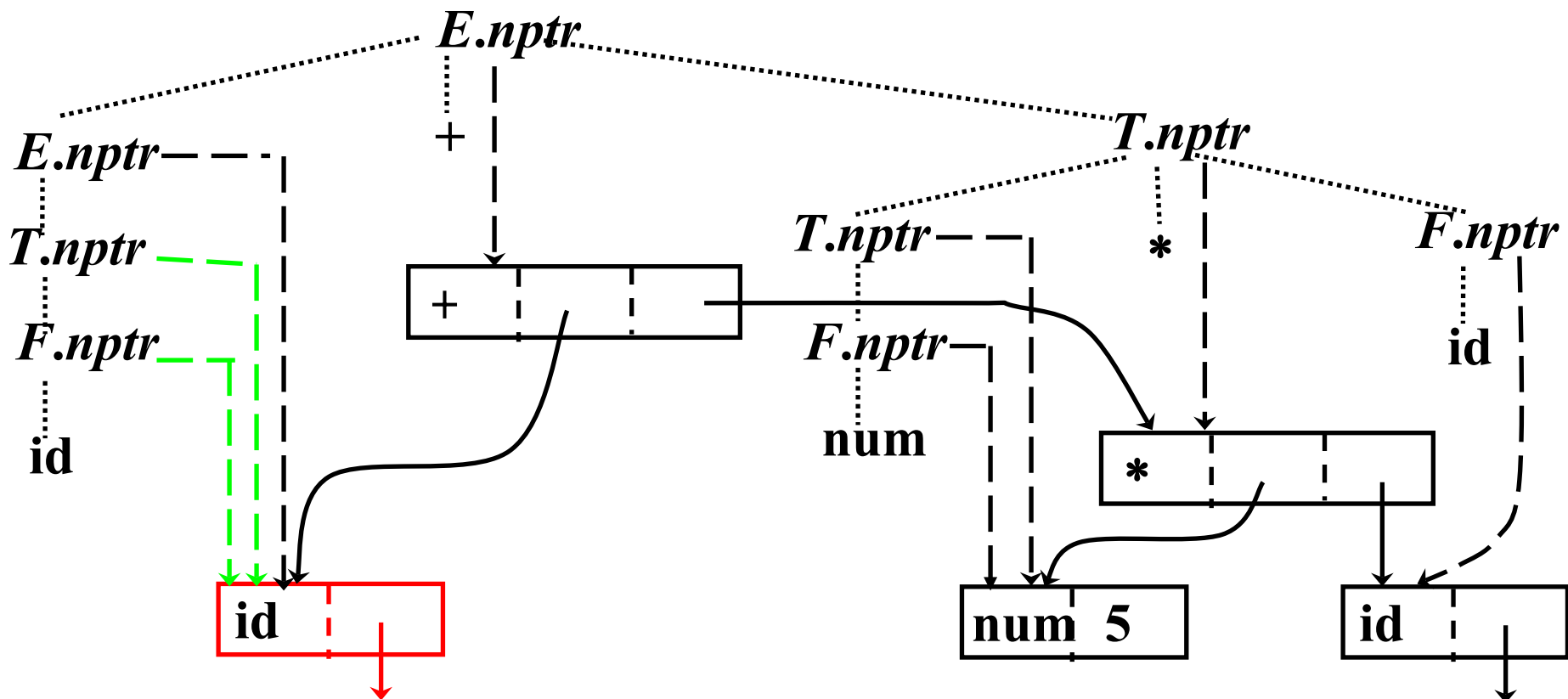
指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口





## $a+5*b$ 的语法树的构造

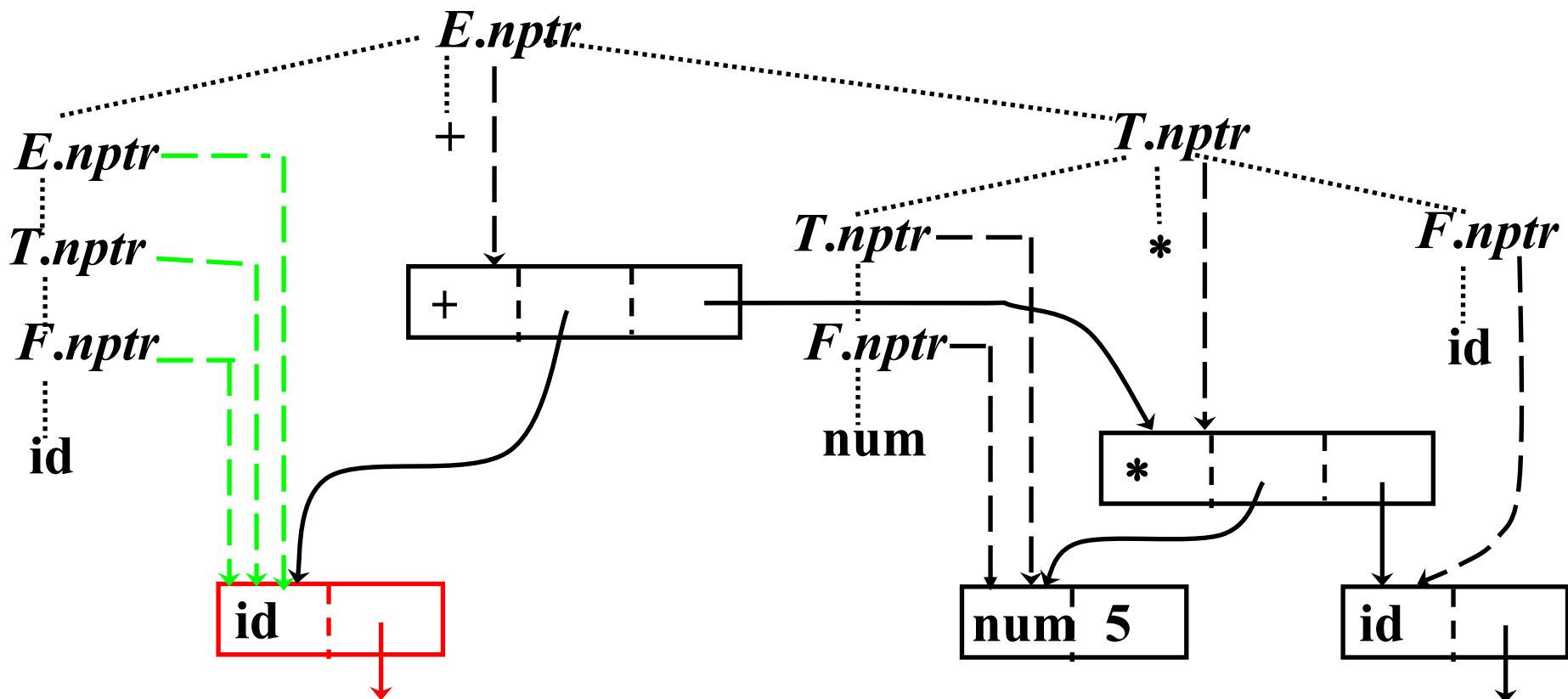


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造

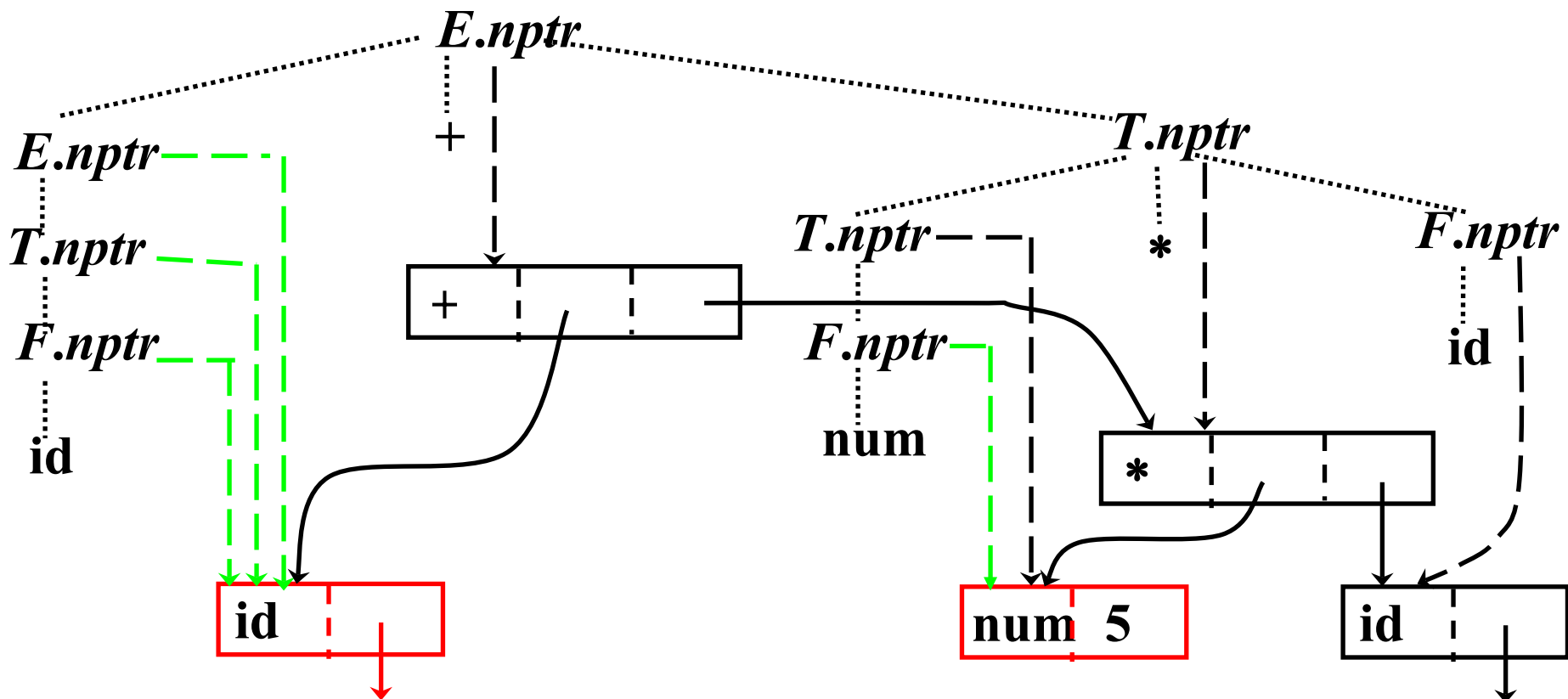


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造

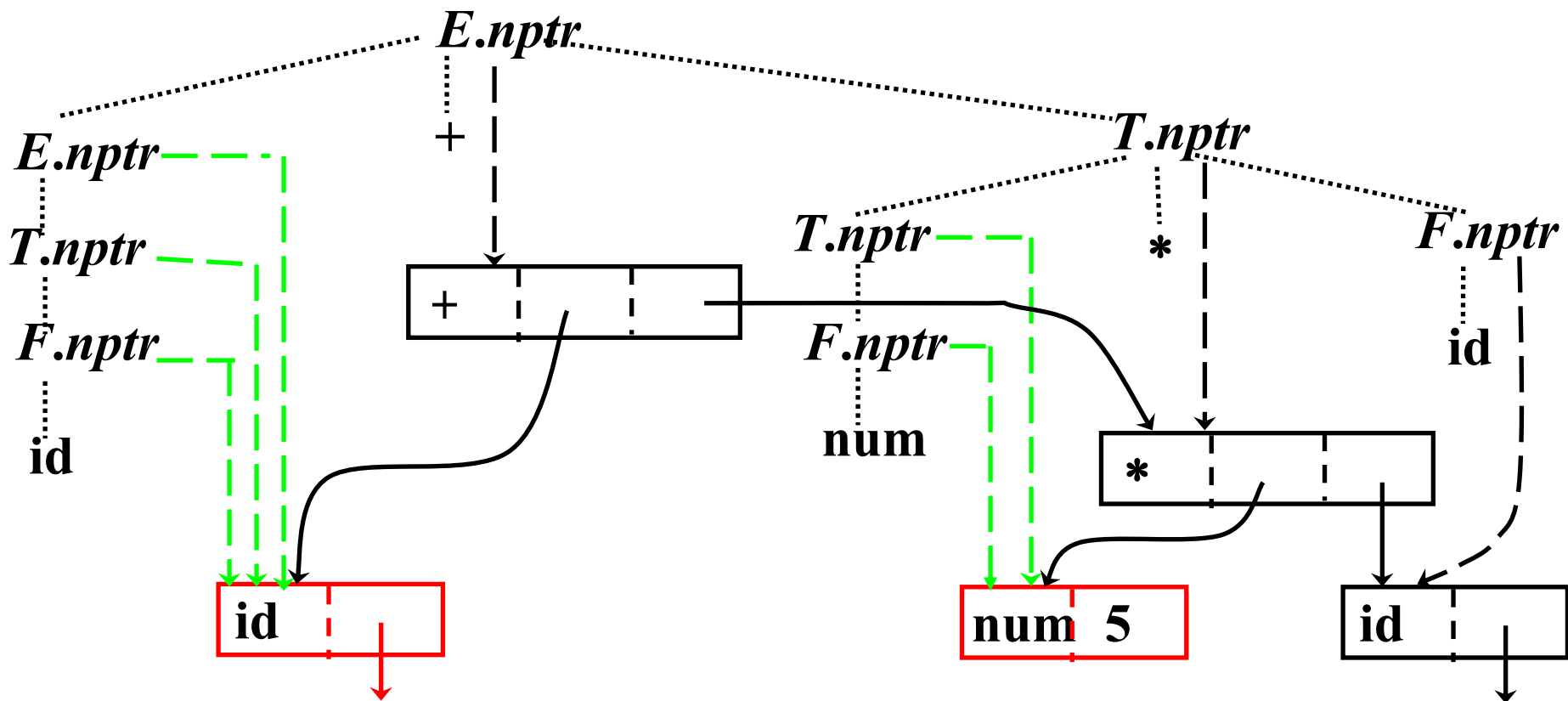


指向符号表中  $a$  的入口

指向符号表中  $b$  的入口



## $a+5*b$ 的语法树的构造

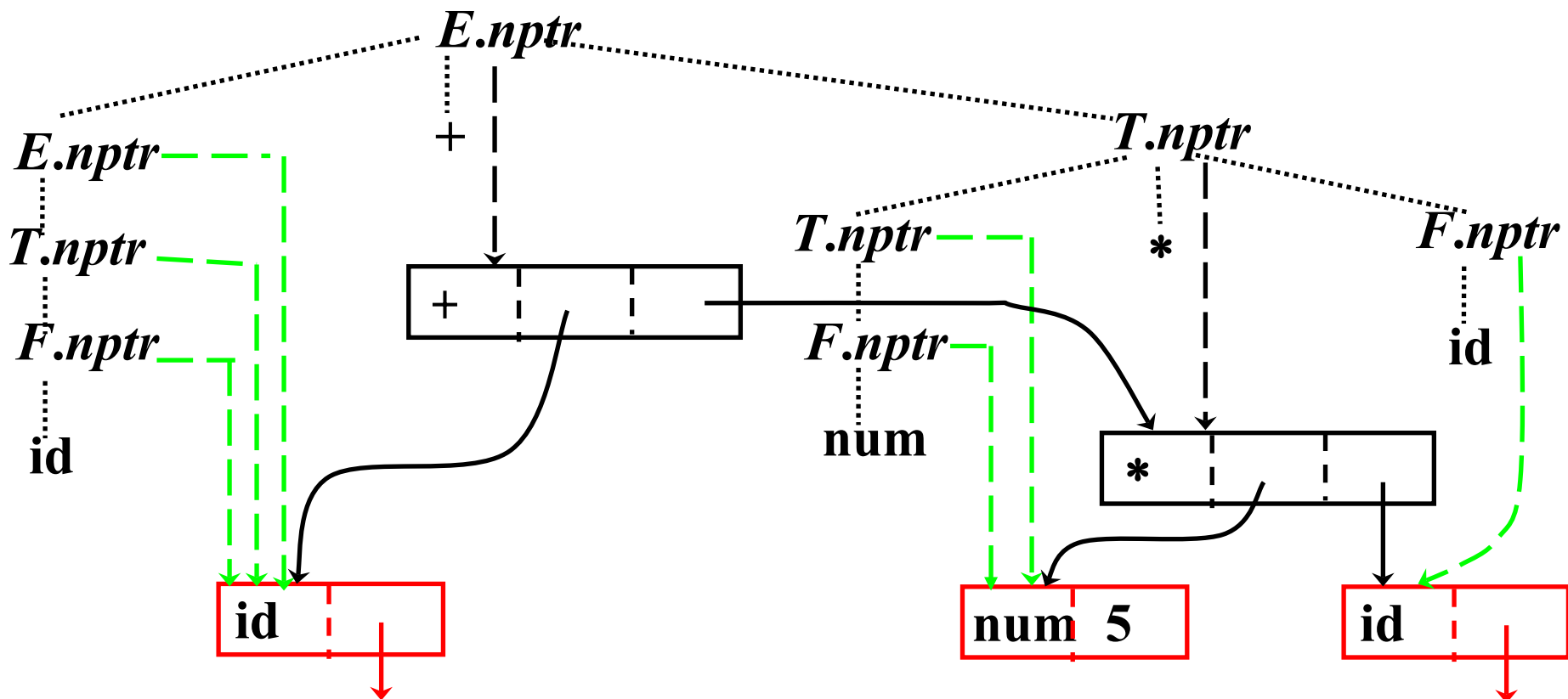


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造

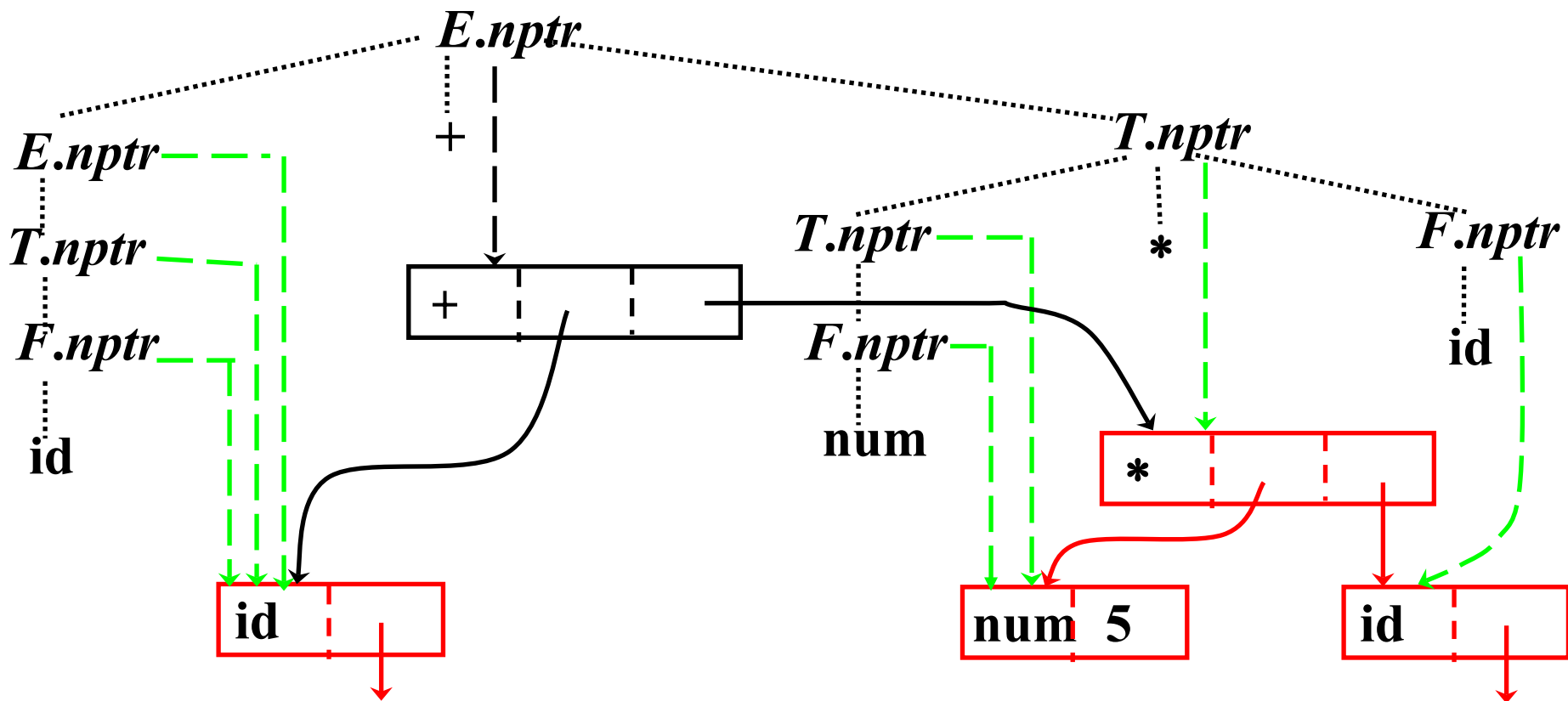


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造

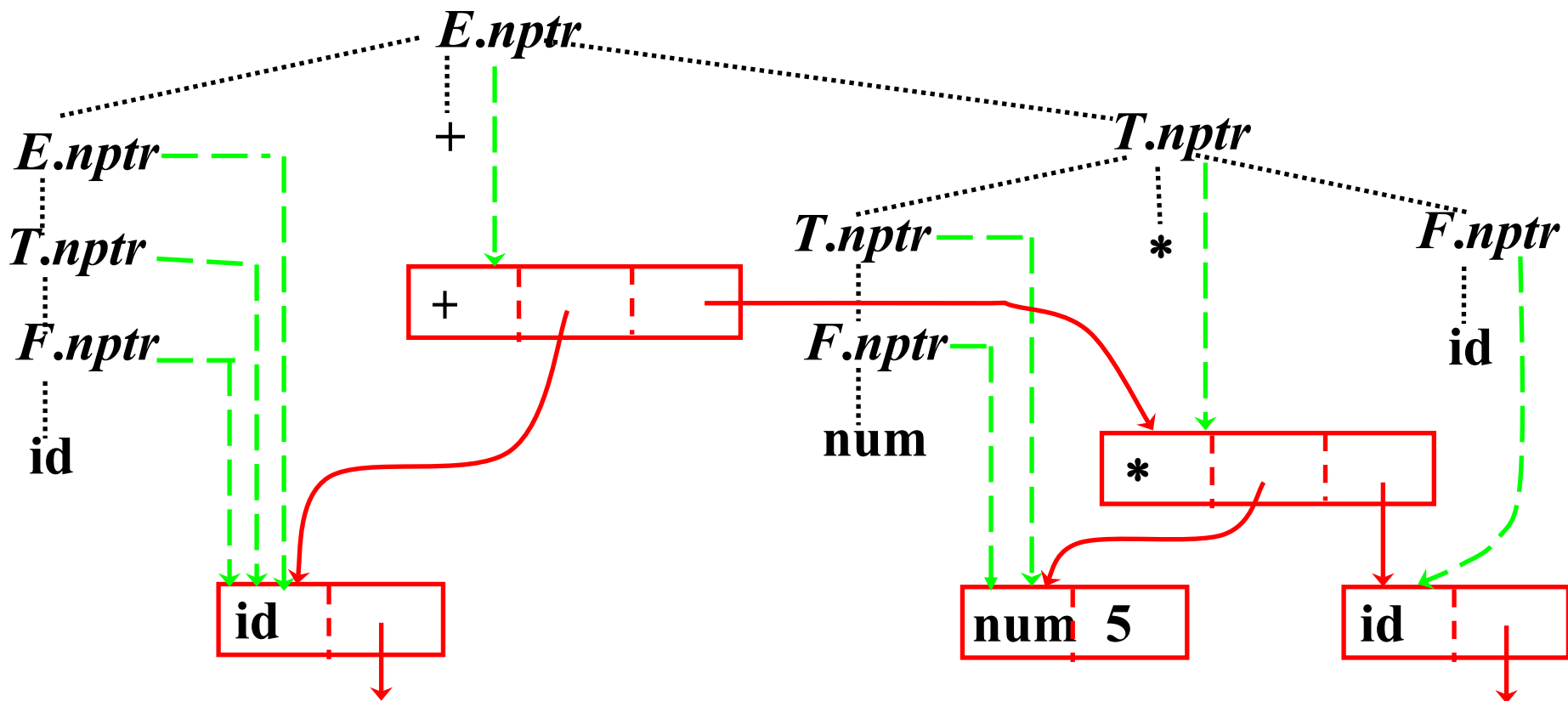


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



## $a+5*b$ 的语法树的构造

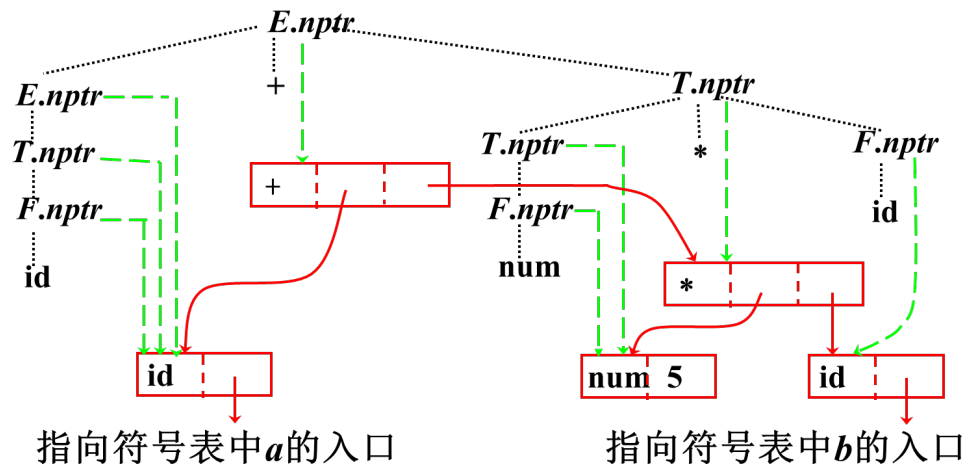


指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口



- 1) **p1:=mkleaf(id,entry a);**
- 2) **p2:=mkleaf(num, 5);**
- 3) **p3:=mkleaf(id, entry b)**
- 4) **p4:=mknode('\*', p2, p3)**
- 5) **p5:=mknode('+', p1, p4)**



$p_1, p_2, \dots, p_5$ 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。





- 每个属性都是综合属性
- 在依赖图中，总是通过子结点的属性值来计算父结点的属性值。



□每个属性都是综合属性

□在依赖图中，总是通过子结点的属性值来计算父结点的属性值。

❖特别适合与自底向上的语法分析过程一起计算

➤在用产生式归约时，即在构造分析树的中间节点时，计算相关属性（此时其子结点的属性必然已经计算完）



□每个属性都是综合属性

□在依赖图中，总是通过子结点的属性值来计算父结点的属性值。

❖特别适合与自底向上的语法分析过程一起计算

➤在用产生式归约时，即在构造分析树的中间节点时，计算相关属性（此时其子结点的属性必然已经计算完）

❖也可以与自顶向下的语法分析过程一起计算

➤递归下降分析中，可以在过程A()的最后一步计算A的属性(此时，A调用的其他子结点过程已处理完)



## □抽象语法树的构造

❖S属性定义的方法

❖L属性定义的方法

## □类型检查(下一章)

## □中间代码生成(下一章)



## □考虑以下左递归文法

产生式	语义规则
$E \rightarrow E_1 + T$	$E.nptr = mkNode( '+', E_1.nptr, T.nptr )$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode( '*', T_1.nptr, F.nptr )$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow id$	$F.nptr = mkLeaf( id, id.entry )$
$F \rightarrow num$	$F.nptr = mkLeaf( num, num.val )$



## □首先消除左递归

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

$T + T + T + \dots$

$E \rightarrow TR$

$R \rightarrow +TR_1$

$R \rightarrow \varepsilon$

$T \rightarrow FW$

$W \rightarrow *FW_1$

$W \rightarrow \varepsilon$

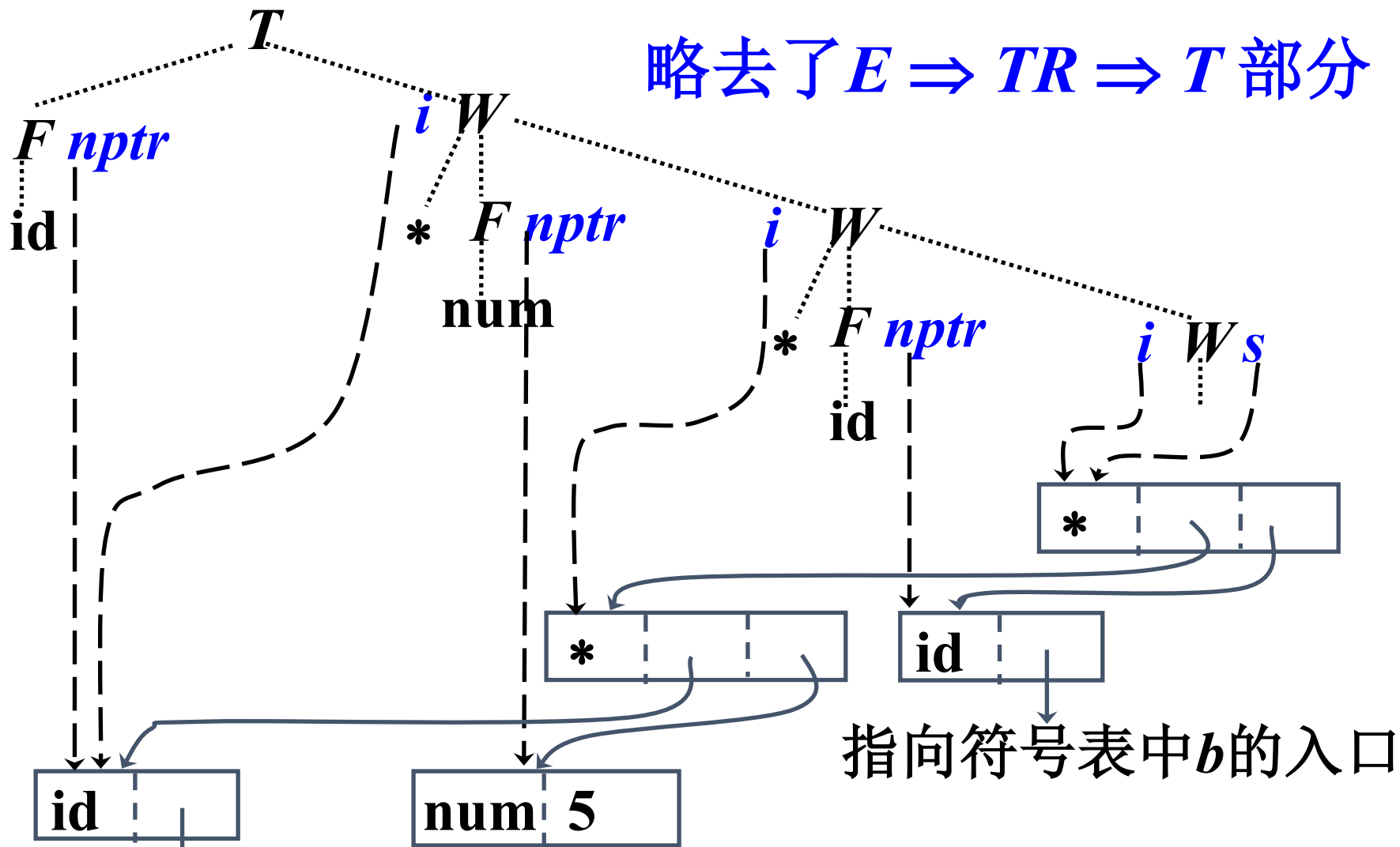
$F$  产生式部分不再给出



产生式	语义规则
$T \rightarrow FW$	$W.i = F.nptr$ $T.nptr = W.s$
$W \rightarrow *FW_1$	$W_1.i = mkNode ('*', W.i, F.nptr)$ $W.s = W_1.s$
$W \rightarrow \varepsilon$	$W.s = W.i$
$F \rightarrow id$	$F.nptr = mkLeaf(id, id.entry)$
$F \rightarrow num$	$F.nptr = mkLeaf(num, num.val)$



略去了  $E \Rightarrow TR \Rightarrow T$  部分



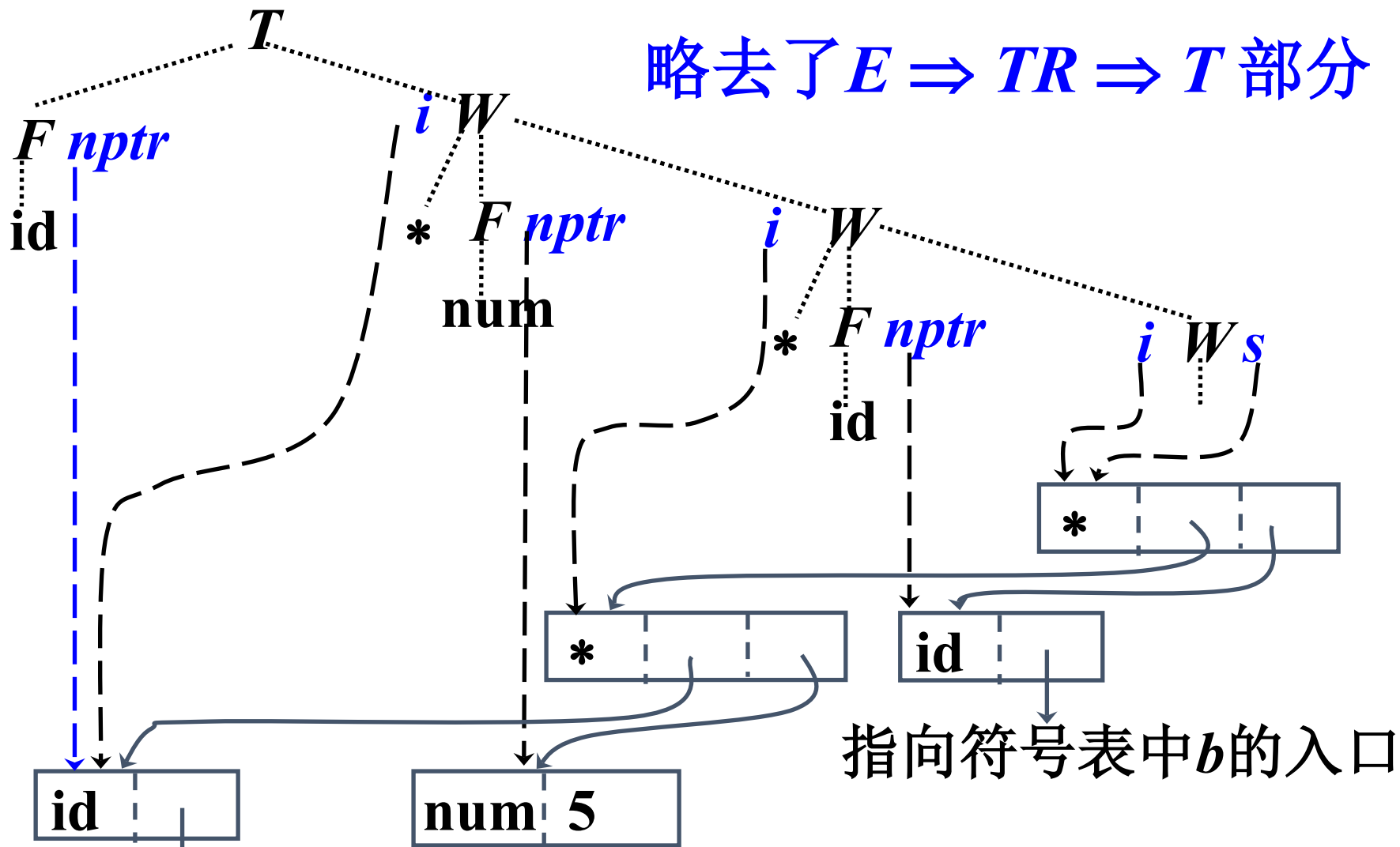
指向符号表中  $b$  的入口

指向符号表中  $a$  的入口





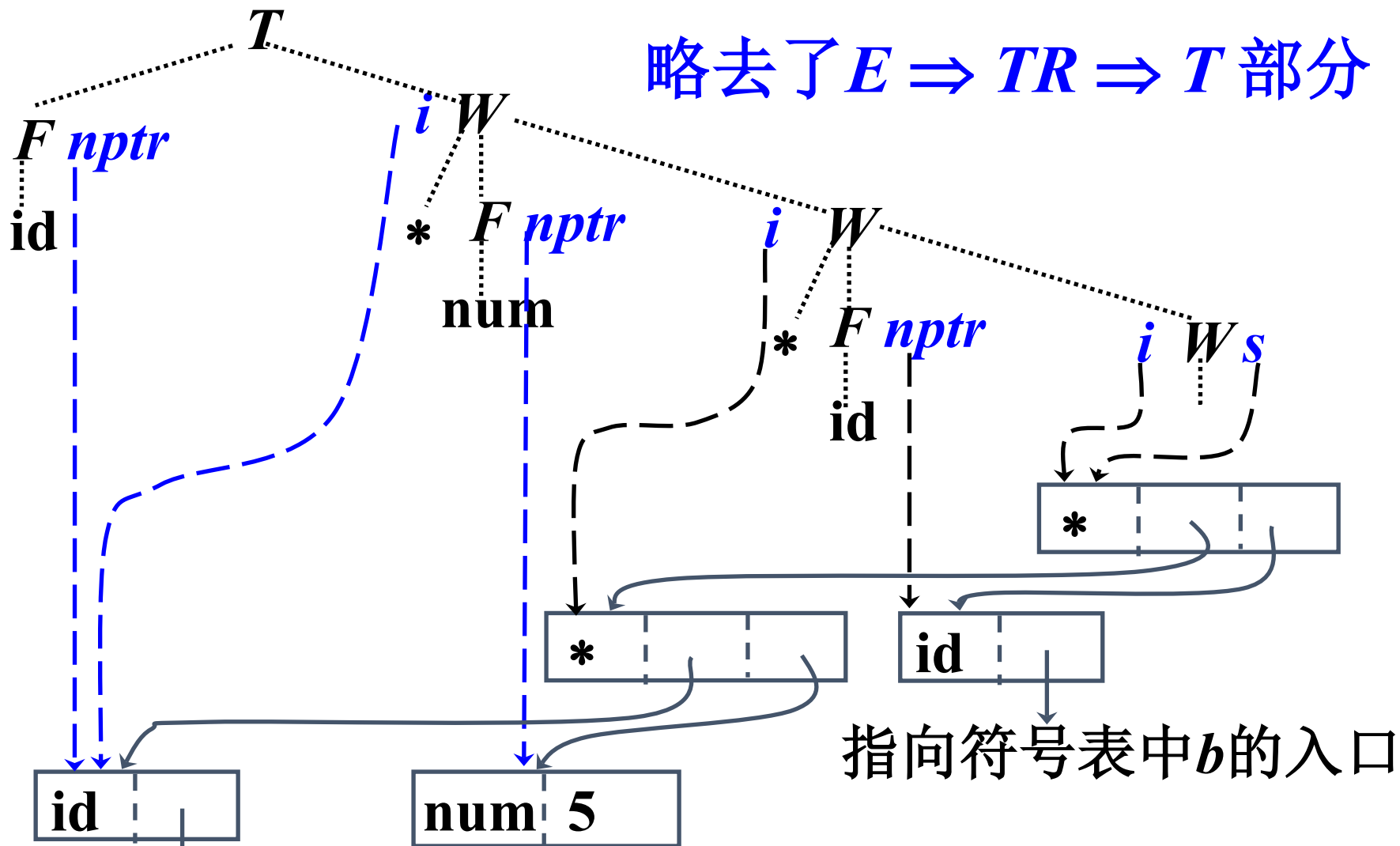
略去了  $E \Rightarrow TR \Rightarrow T$  部分







略去了  $E \Rightarrow TR \Rightarrow T$  部分



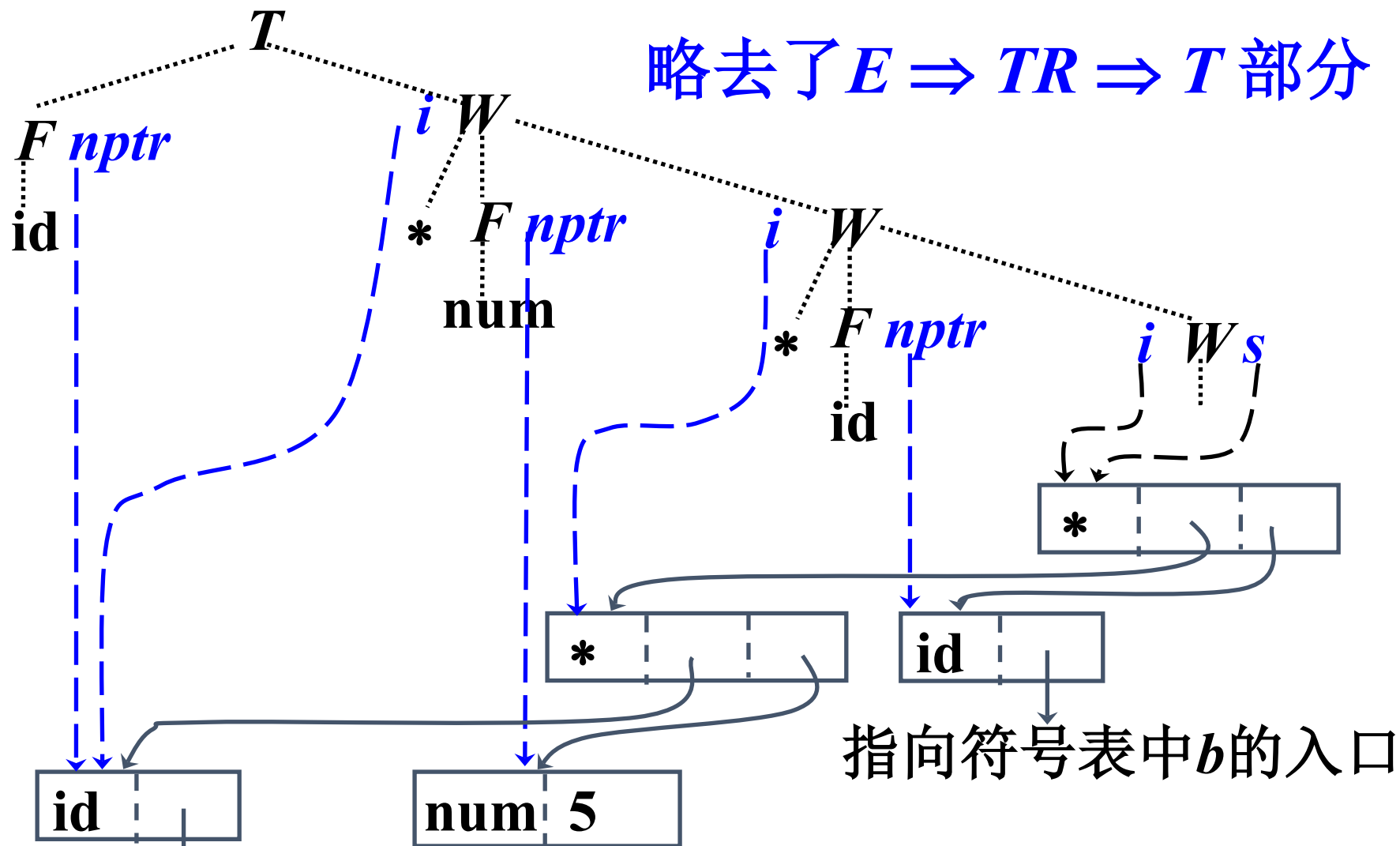
指向符号表中 $a$ 的入口

指向符号表中 $b$ 的入口





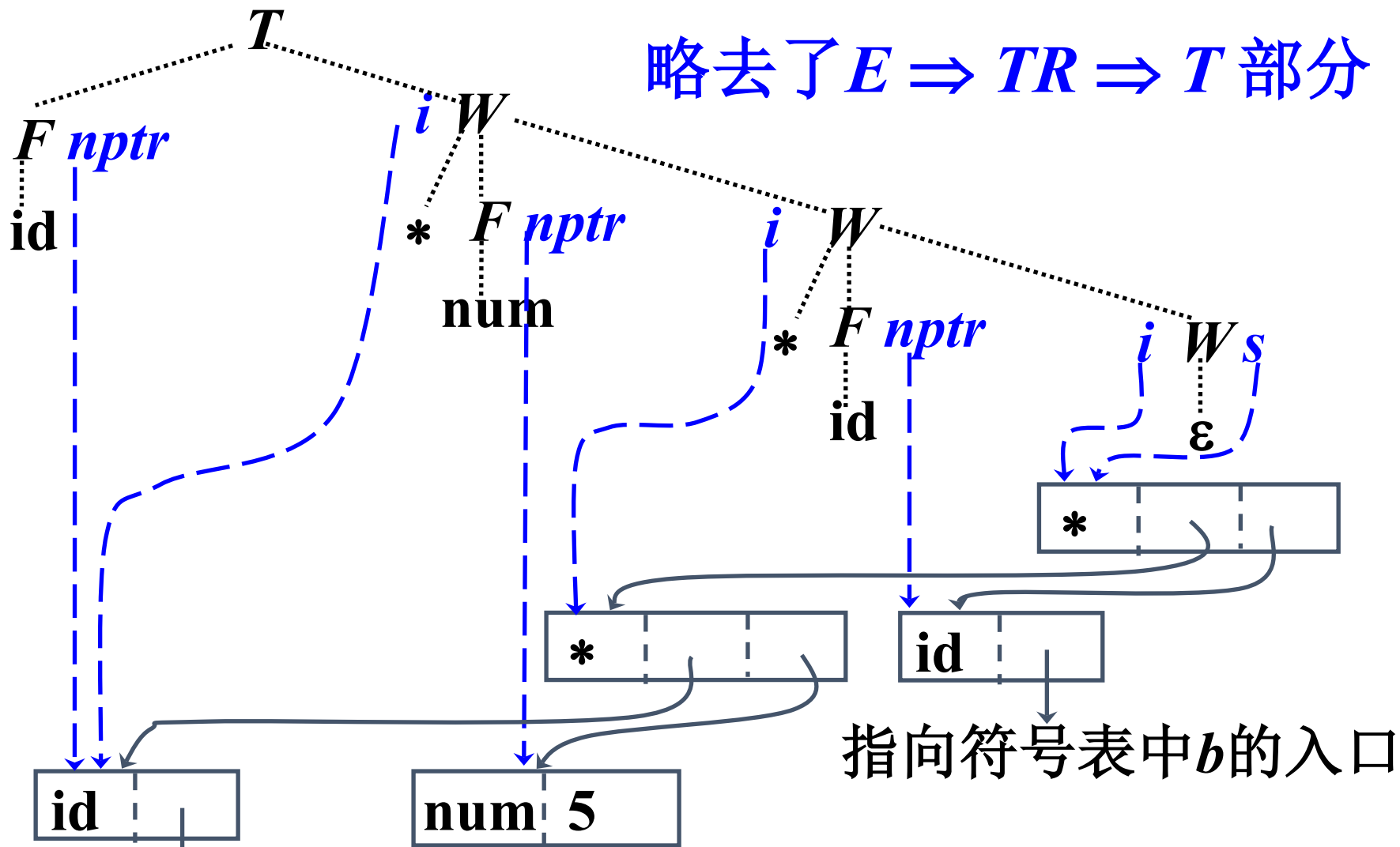
略去了  $E \Rightarrow TR \Rightarrow T$  部分



指向符号表中  $a$  的入口



略去了  $E \Rightarrow TR \Rightarrow T$  部分



指向符号表中  $a$  的入口



□最后抽象语法树都是一样的

□但是，分析树却是不同的

❖S-SDD的分析树与抽象语法树比较接近

❖L-SDD的分析树与抽象语法树结构不同



# 《编译原理与技术》

## 语法制导翻译 I

**I don't spend my time pontificating about high-concept things; I spend my time solving engineering and manufacturing problems.**

**—— *Elon Musk***