

语法制导翻译

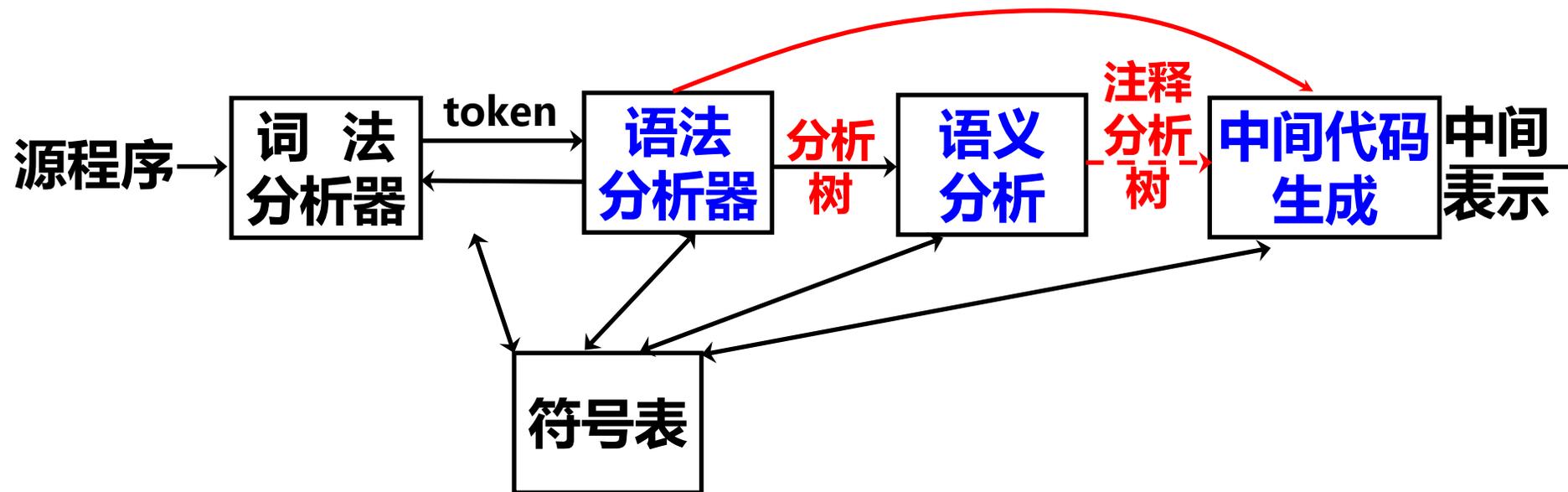
Part1: 语法制导定义

李诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年9月30日



• 语法制导翻译简介

• 语法制导定义

- 属性、属性文法、综合属性及继承属性
- 属性依赖图与计算次序



- **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

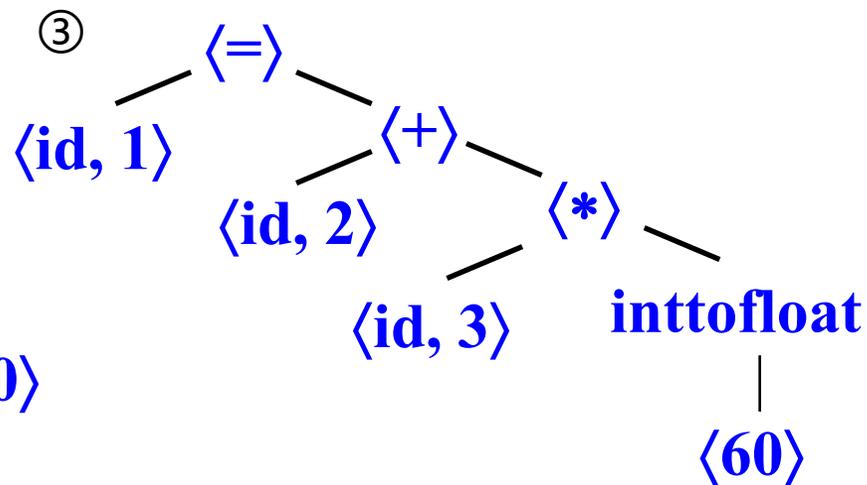


• 中间代码生成：将源程序翻译为中间代码

- 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

① `position = initial + rate * 60`

② `<id, 1> <=> <id, 2> <+> <id, 3> <*> <60>`



④ `t1 = inttofloat(60)`
`t2 = id3 * t1`
`t3 = id2 + t2`
`id1 = t3`



- **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程序片段

```
int arr[2];
```

```
arr[2] = 2024;
```



- **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

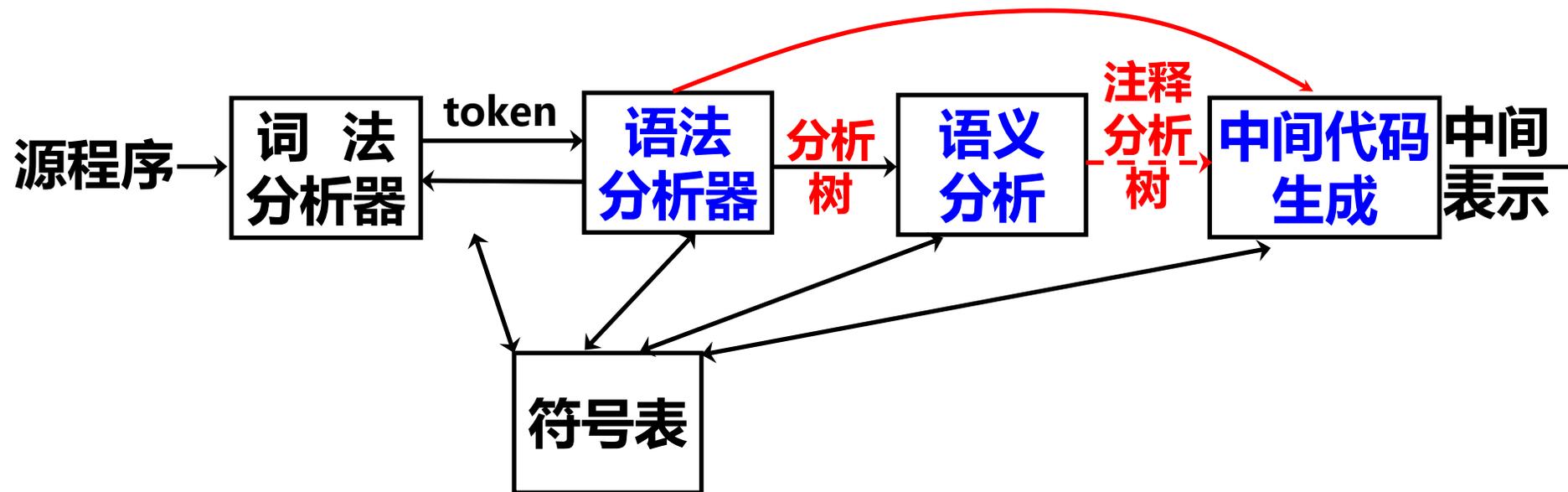


• 如何表示语义信息？

- 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、属性文法、综合属性及继承属性
- 属性依赖图与计算次序



• 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

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

带副作用的规则
(虚拟属性)

对E加下标以区分不同的属性值

词法分析给定



属性的计算



- **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？**

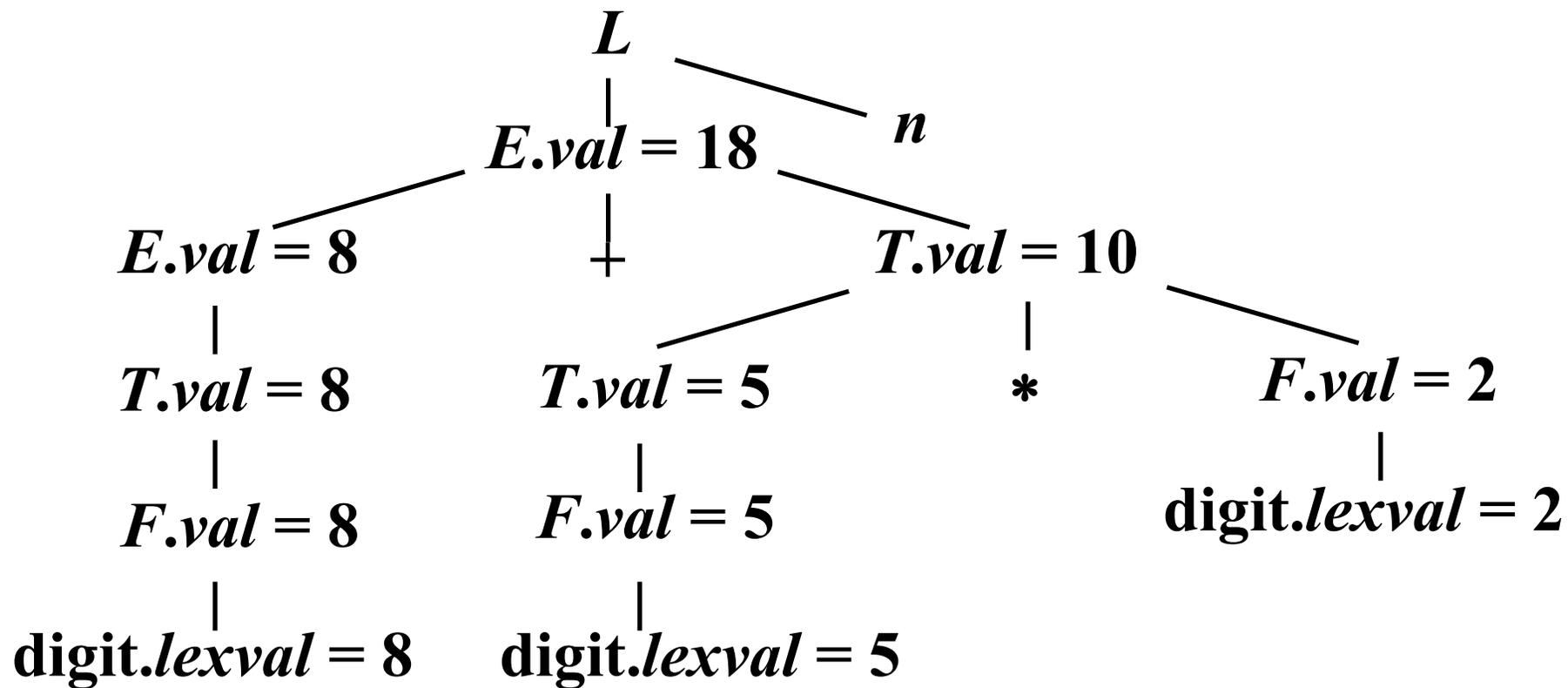


- **思考：如何将语义规则所引起的属性计算与语法分析结合起来？**
- **语法分析的过程显式或隐式地构造了分析树(parse tree)**
- **因此，可以先构造分析树，在分析树上，对每一个节点上的属性值进行求值**
 - 注释分析树(Annotated parse tree)



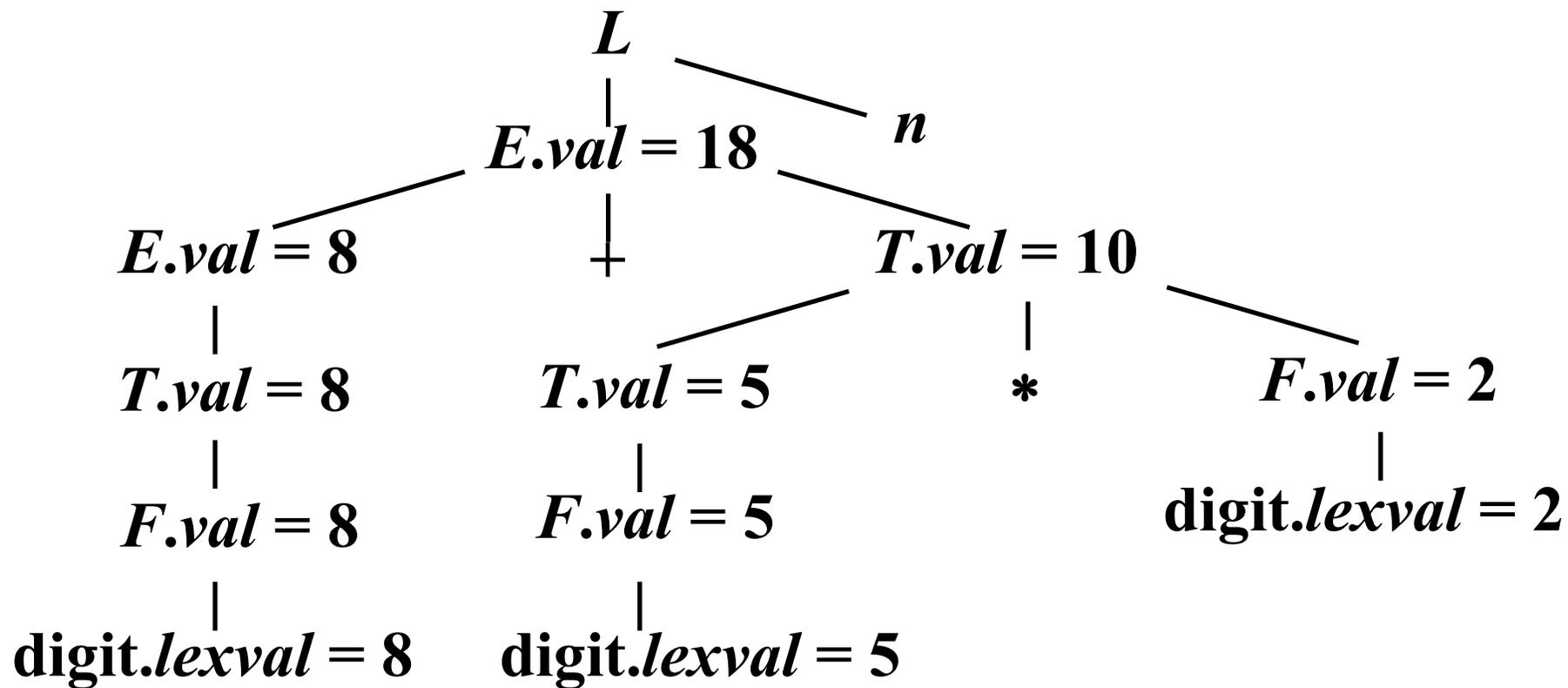
- 定义：结点的属性值都标注出来的分析树

$8+5*2$ 的注释分析树



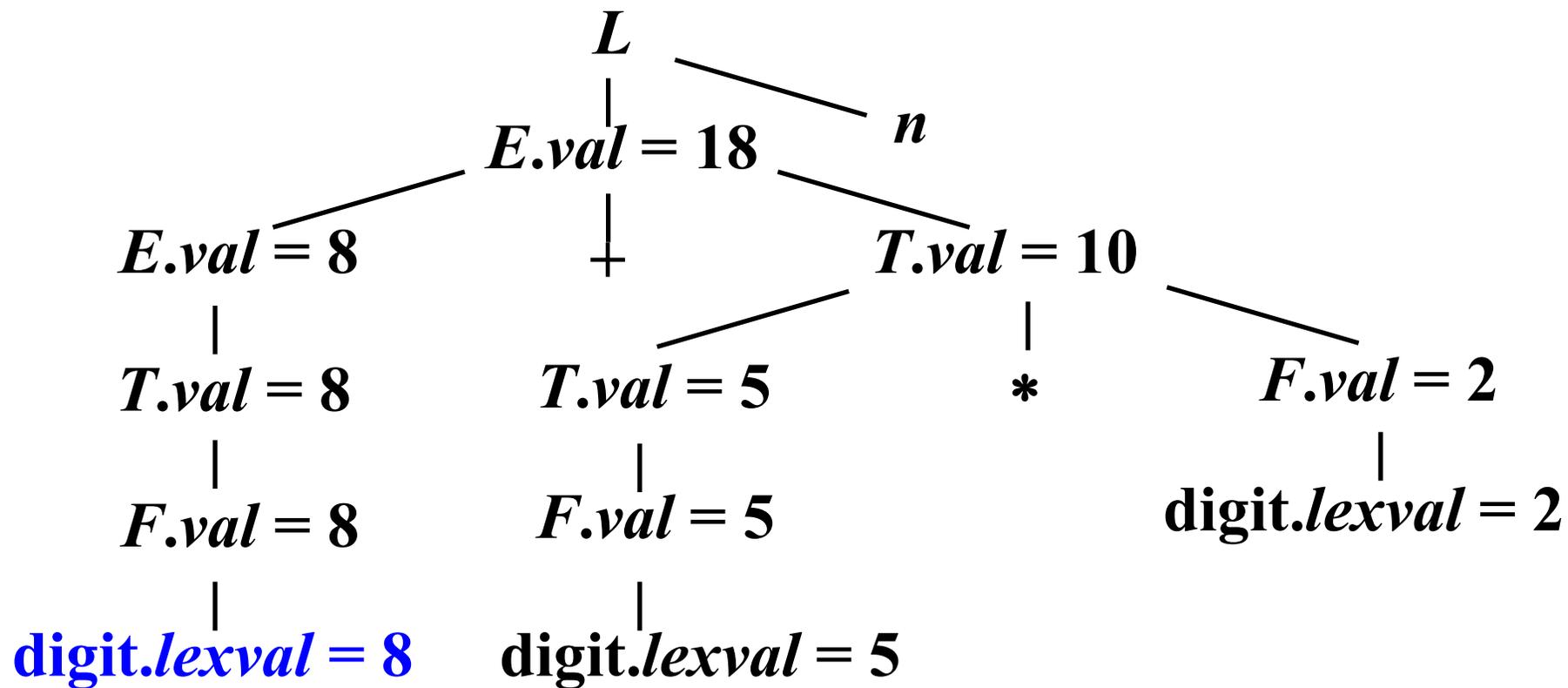


- 各结点综合属性的计算可以自底向上地完成



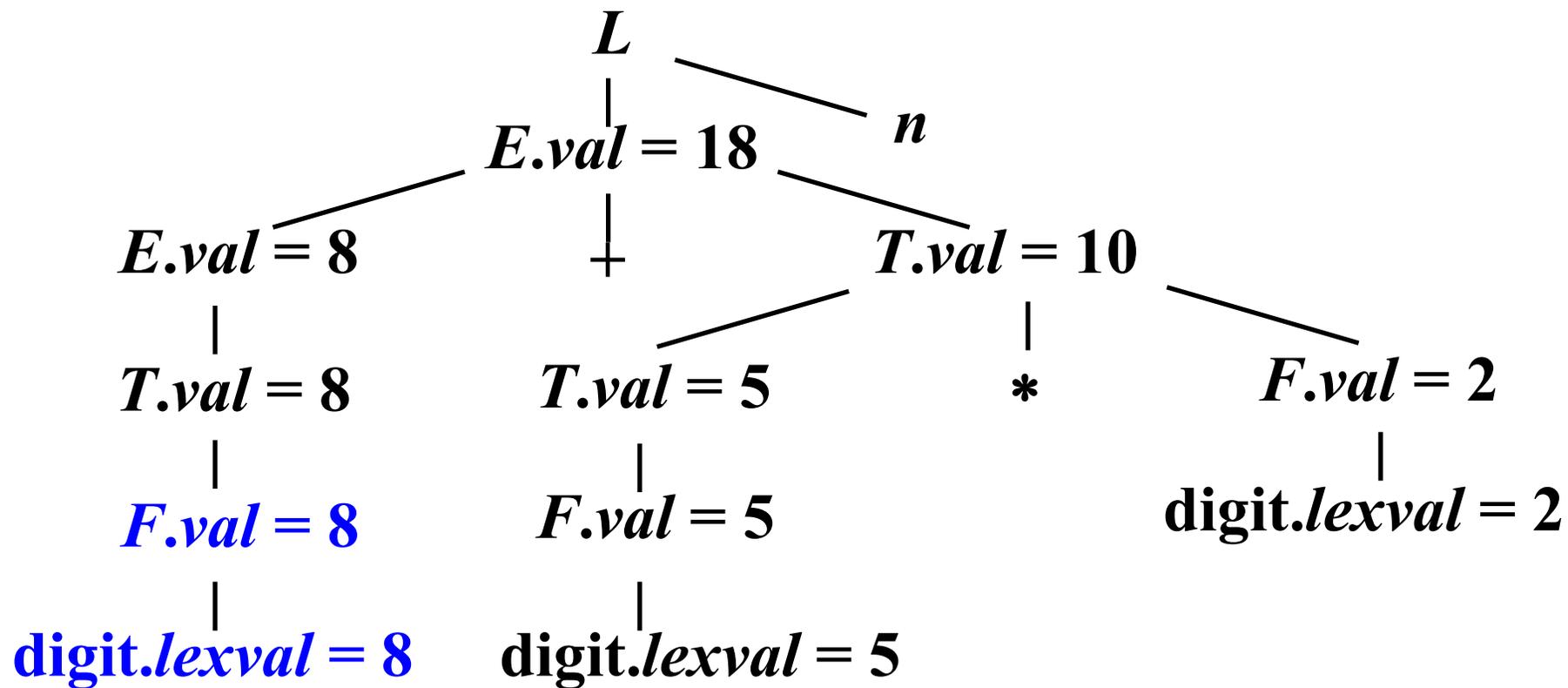


- 各结点综合属性的计算可以自底向上地完成



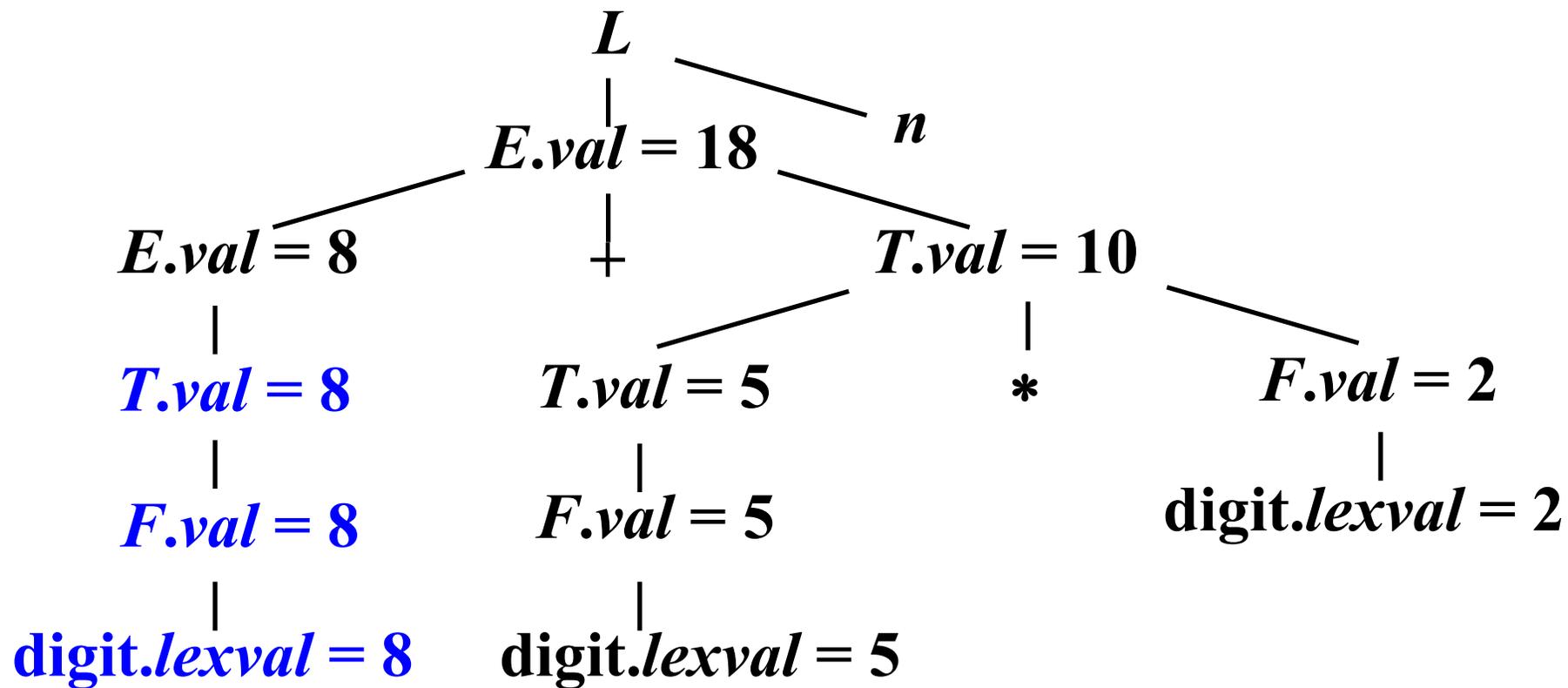


- 各结点综合属性的计算可以自底向上地完成



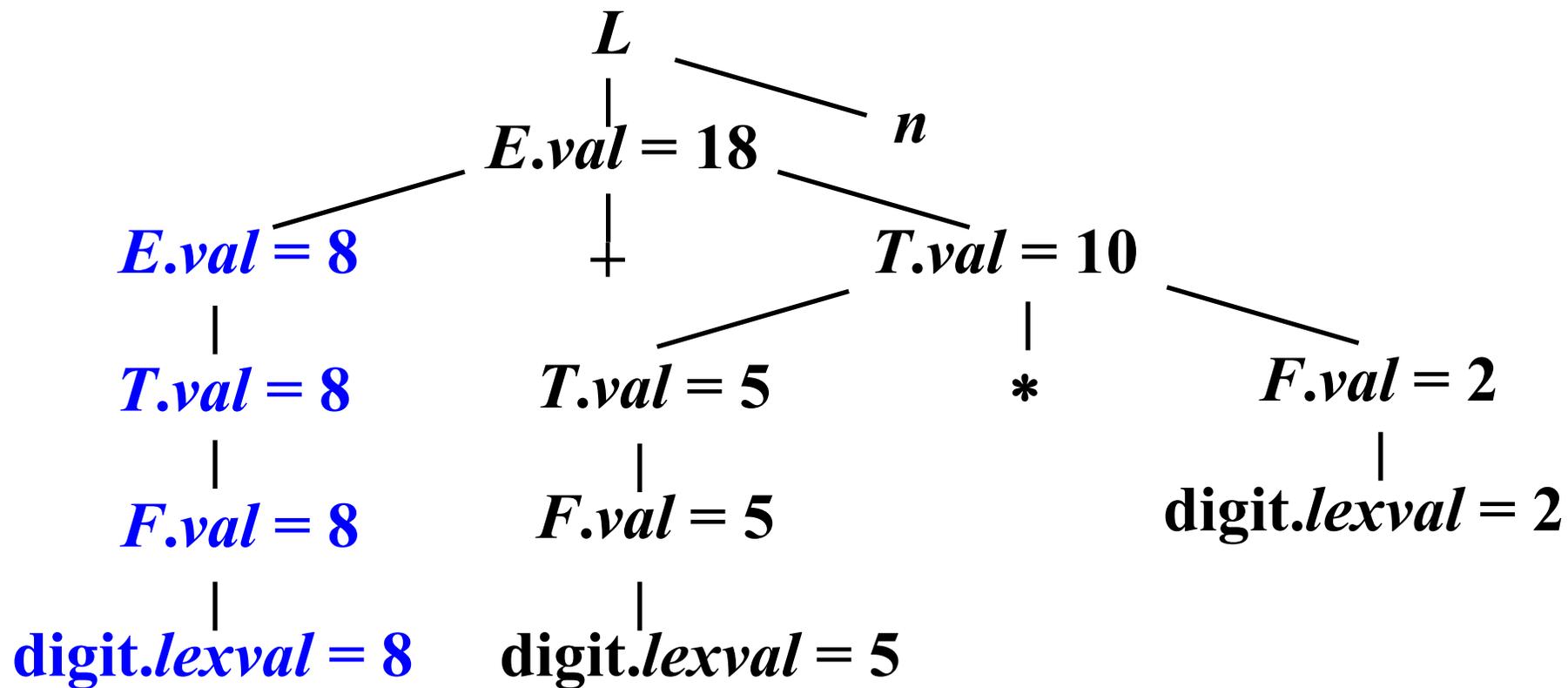


- 各结点综合属性的计算可以自底向上地完成



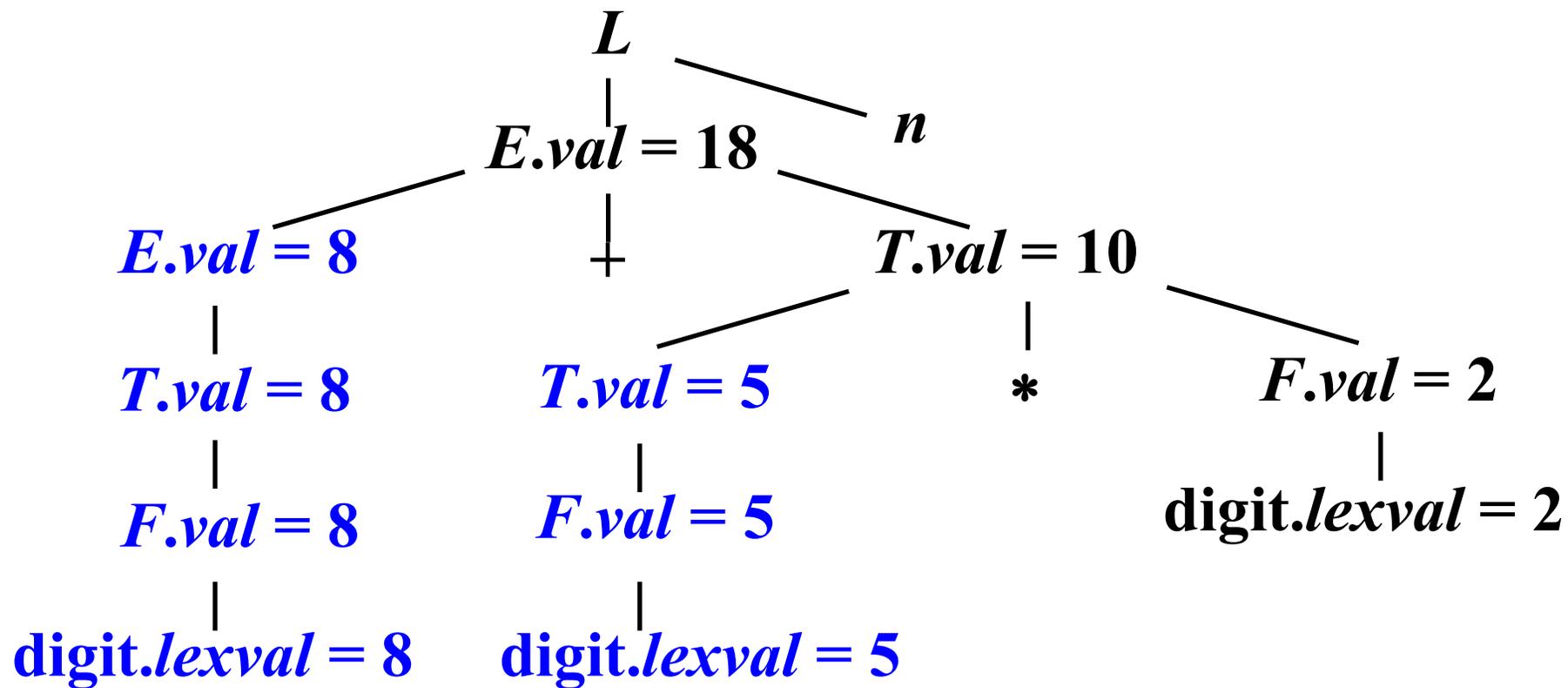


- 各结点综合属性的计算可以自底向上地完成



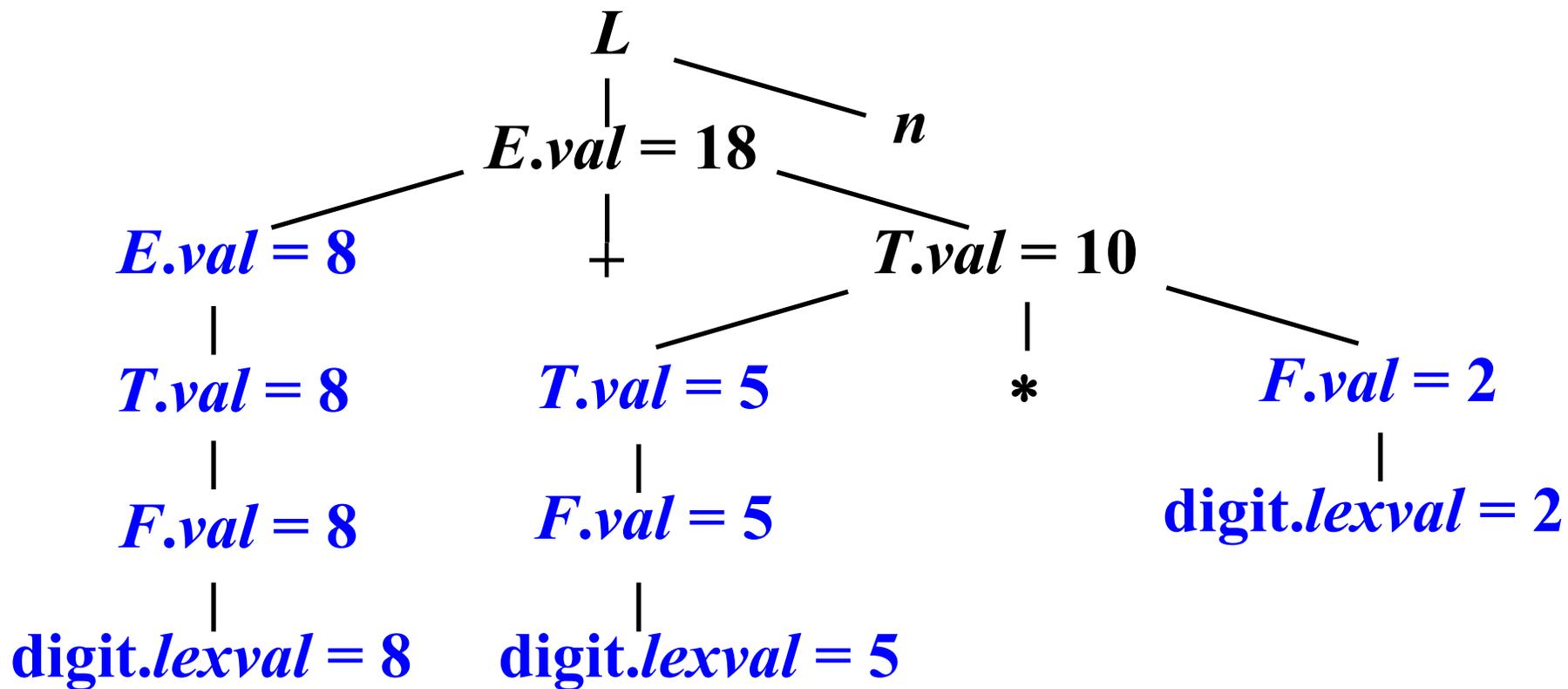


- 各结点综合属性的计算可以自底向上地完成



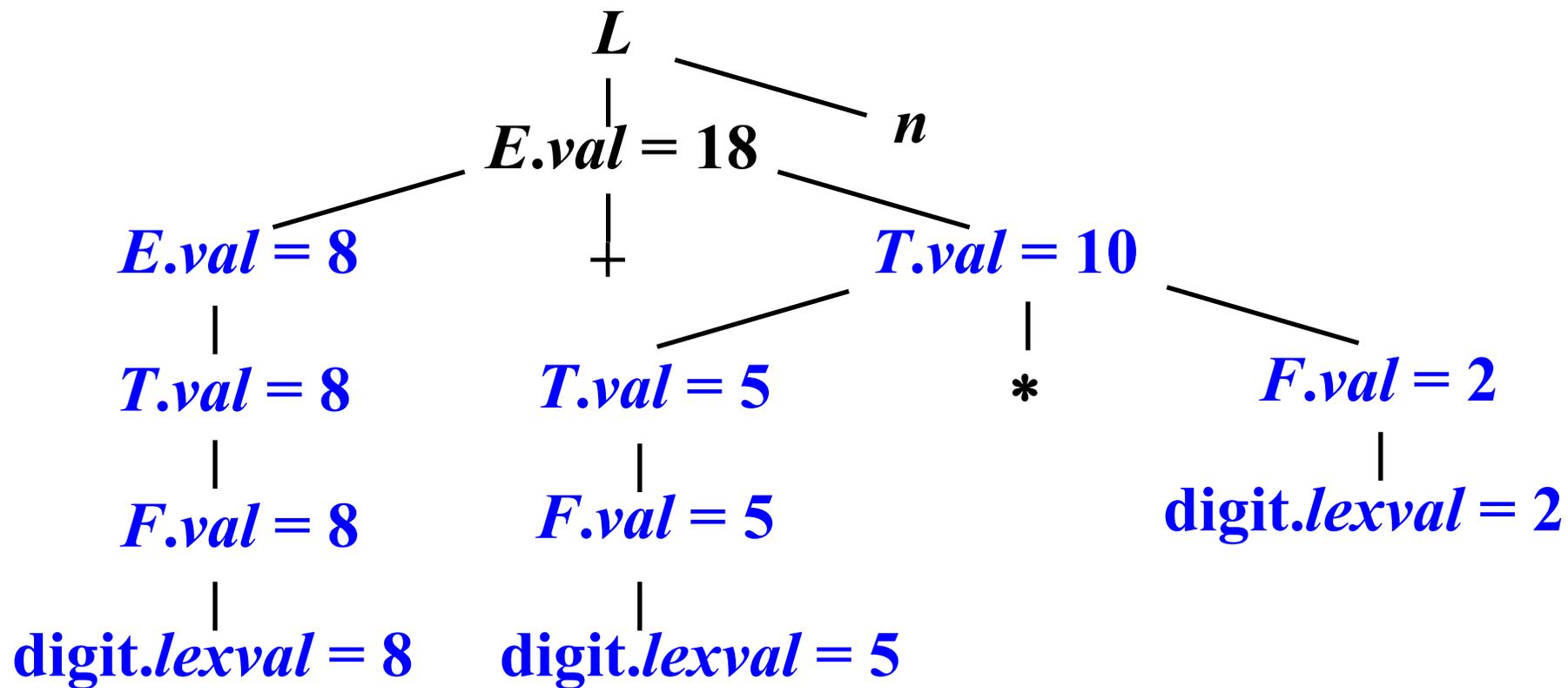


- 各结点综合属性的计算可以自底向上地完成





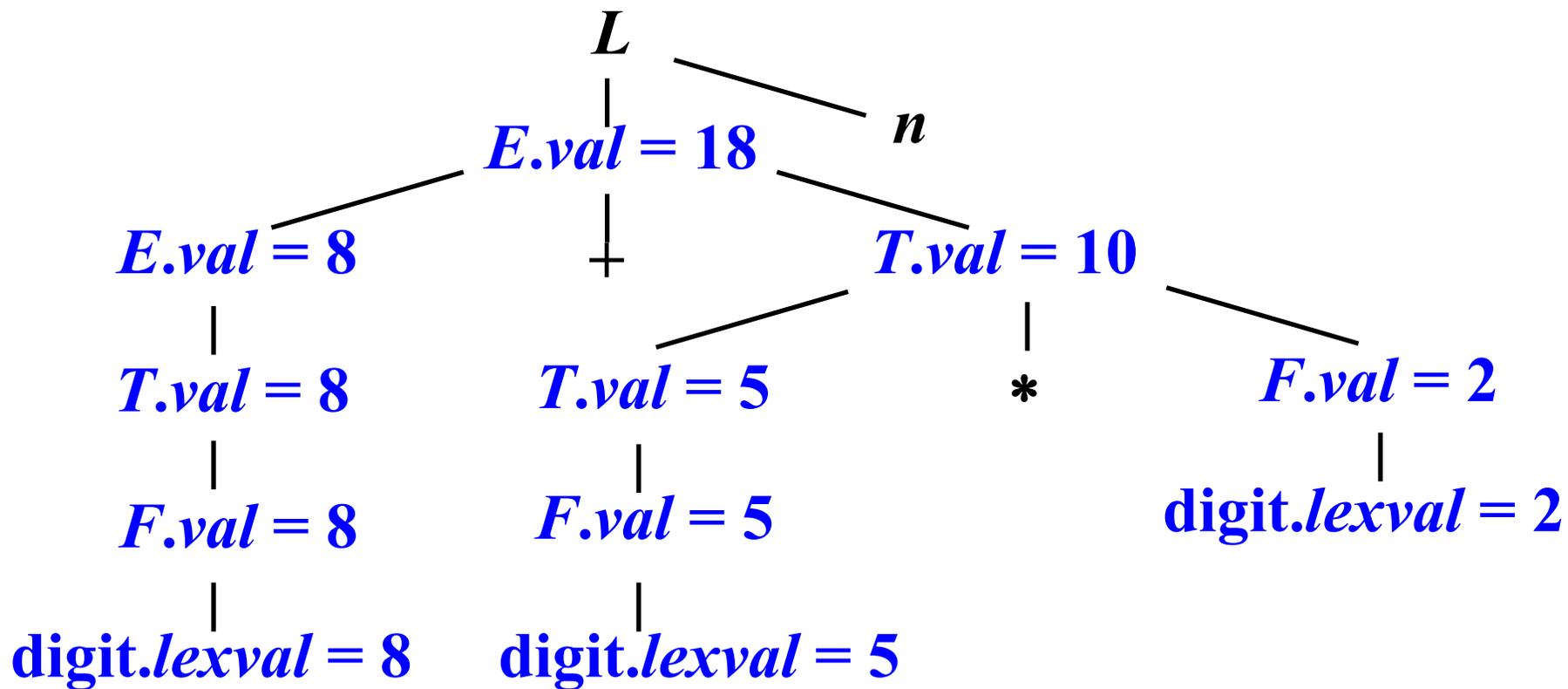
- 各结点综合属性的计算可以自底向上地完成





- 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



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
- 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
$E \rightarrow E_1 - T$	$E.type = \text{if } (E_1.type == T.type) T.type$ 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
$T \rightarrow FT'$
$T' \rightarrow *FT'_1$
$T' \rightarrow \varepsilon$
$F \rightarrow \text{digit}$

- **答案是否定的**，因为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

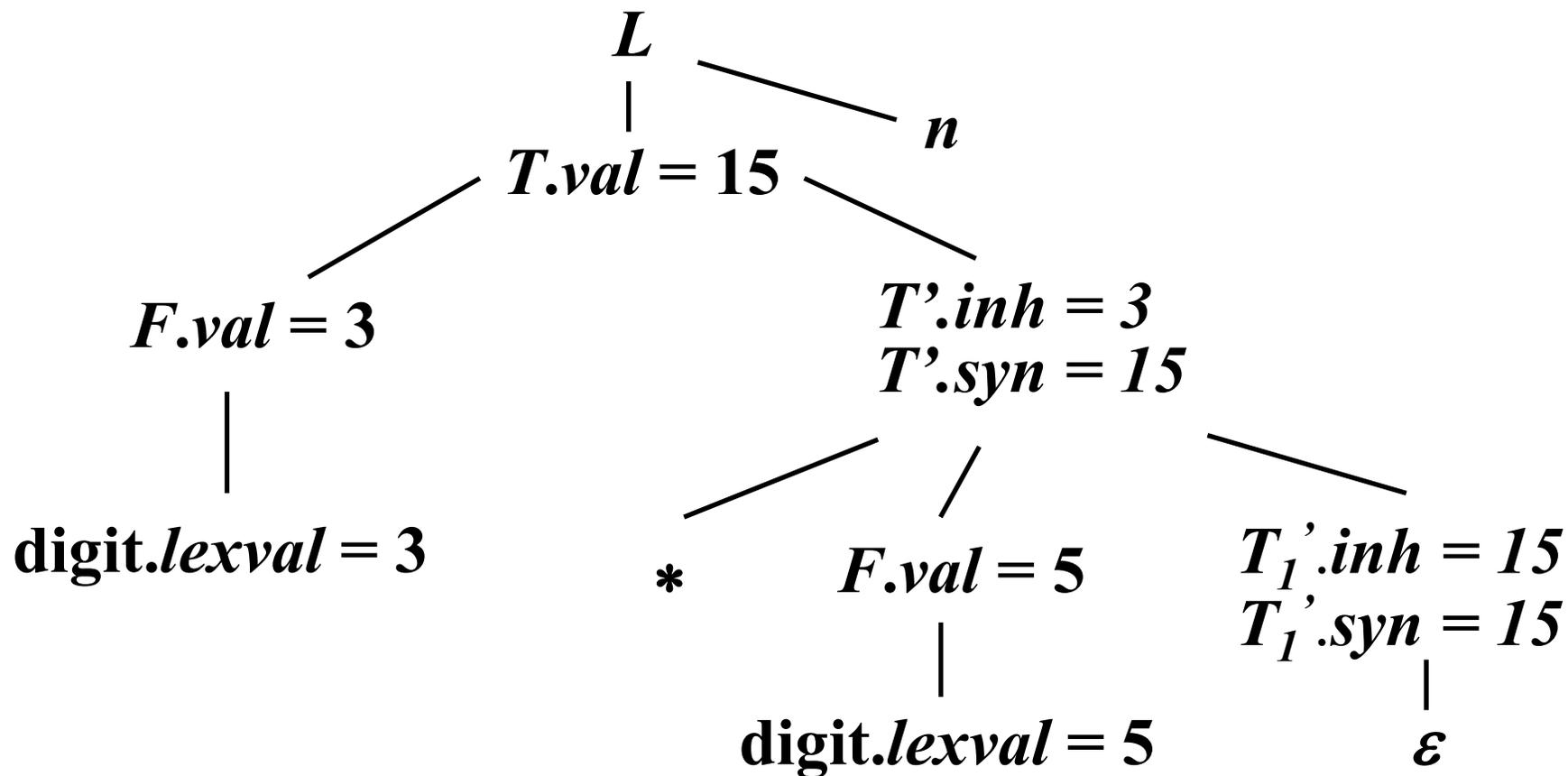
产生式	语义规则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

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

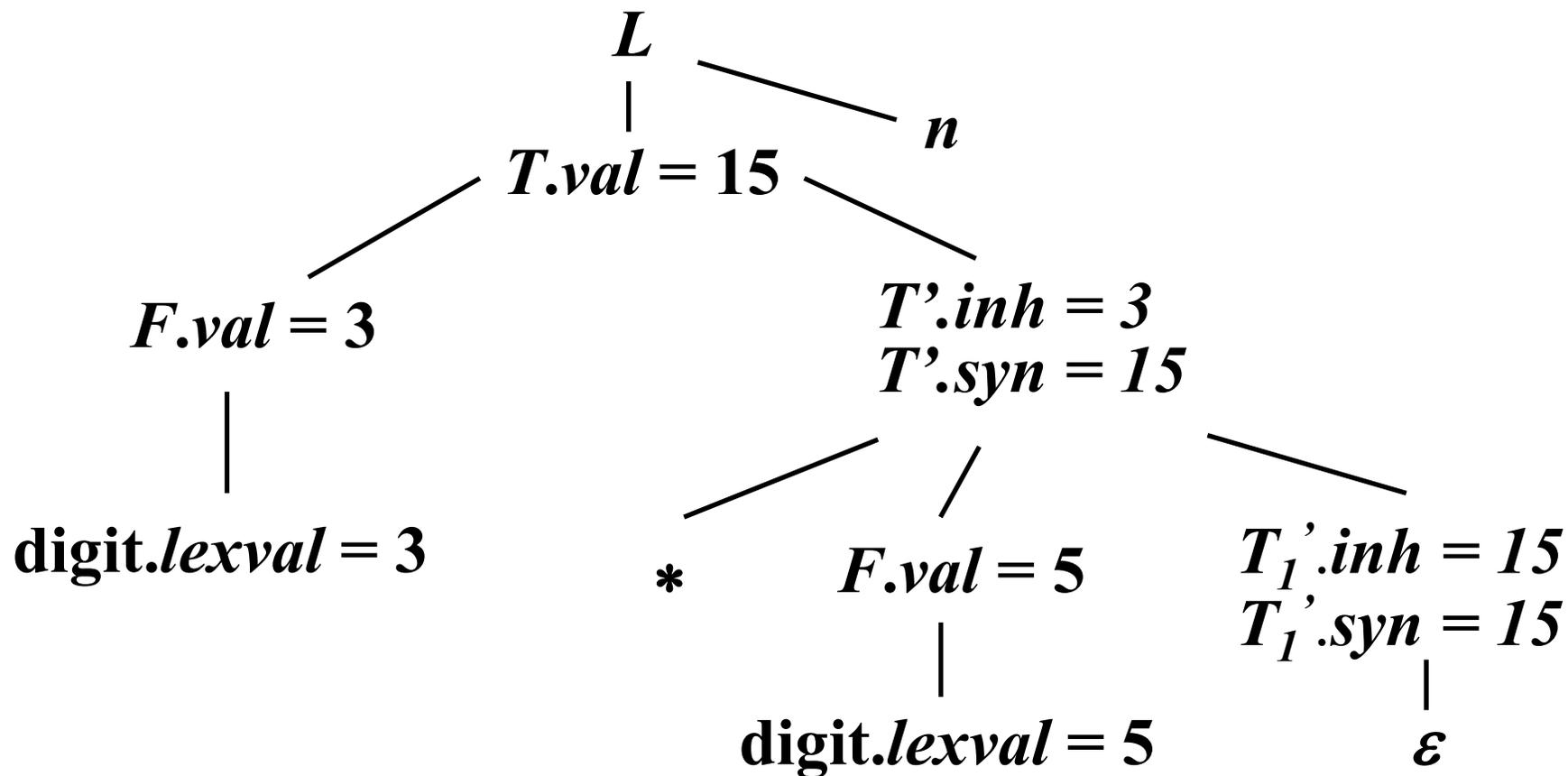


• 3*5的注释分析树





- 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

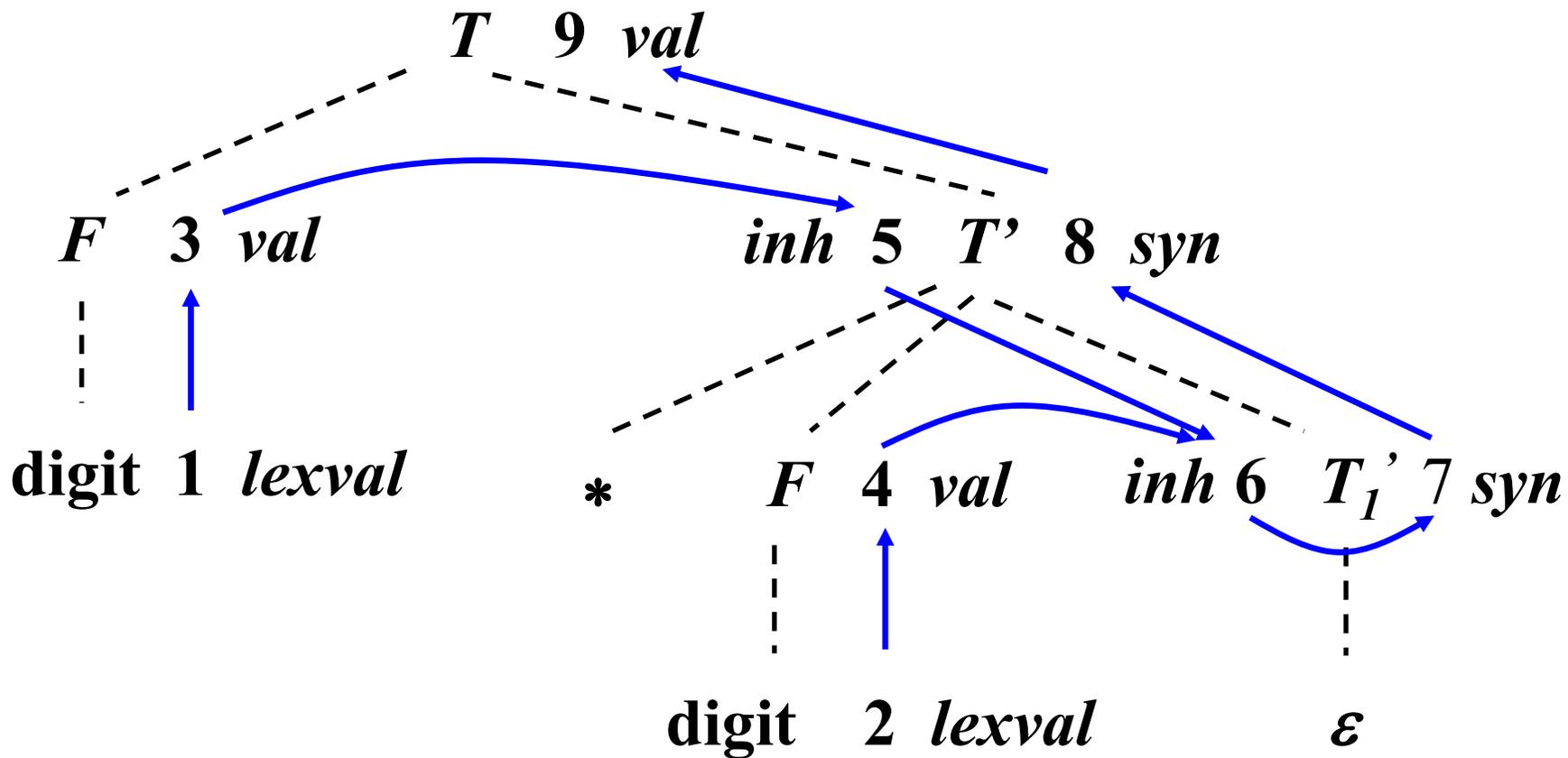
- **按照什么顺序计算属性值？**
 - 语义规则建立了属性之间的依赖关系，在对语法分析树节点的一个属性求值之前，必须首先求出这个属性值所依赖的所有属性值



- **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





属性的计算次序



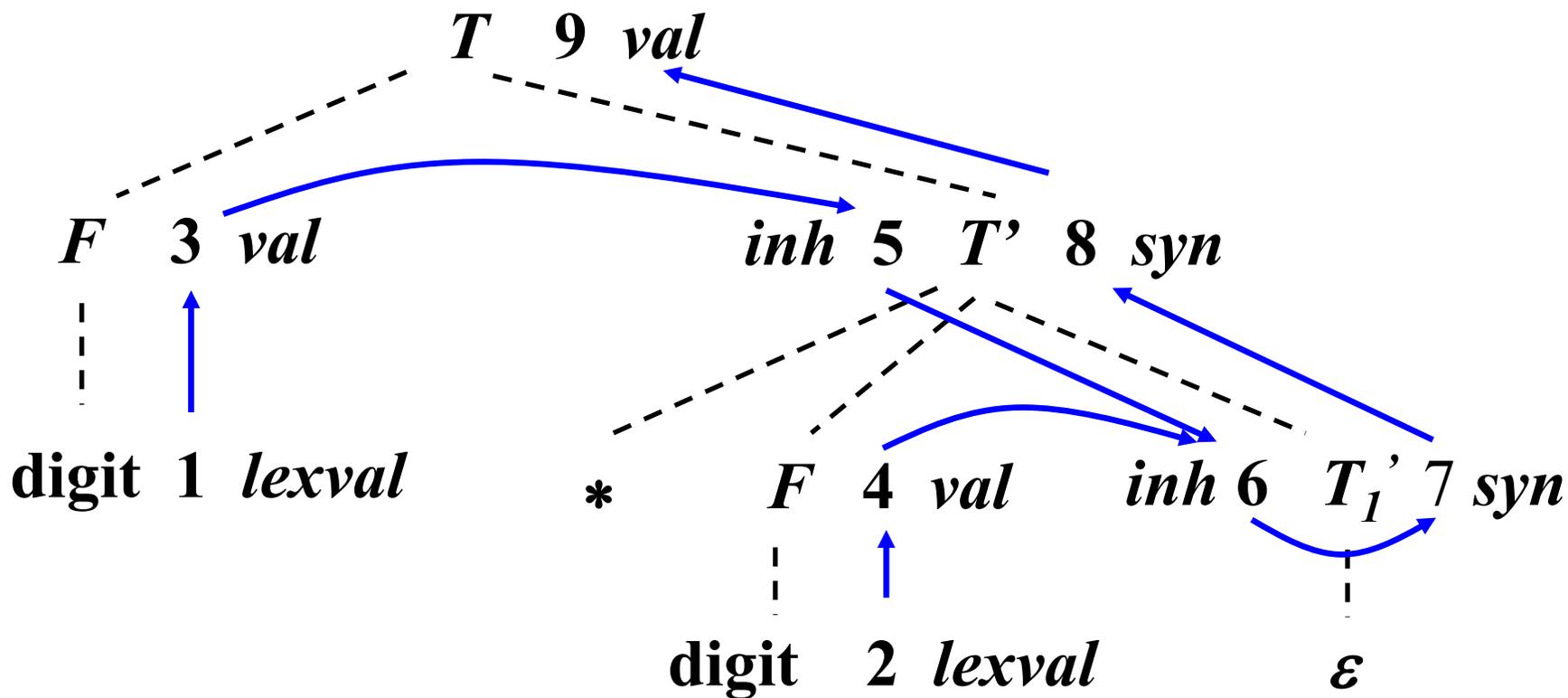
- 可行的求值顺序是满足下列条件的结点序列 N_1, N_2, \dots, N_k :
 - 如果依赖图中有一条从结点 N_i 到 N_j 的边 ($N_i \rightarrow N_j$), 那么, 在节点序列中, N_i 排在 N_j 前面
 - 该排序称为这个图的 **拓扑排序 (topological sort)**



属性计算次序：举例



- 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



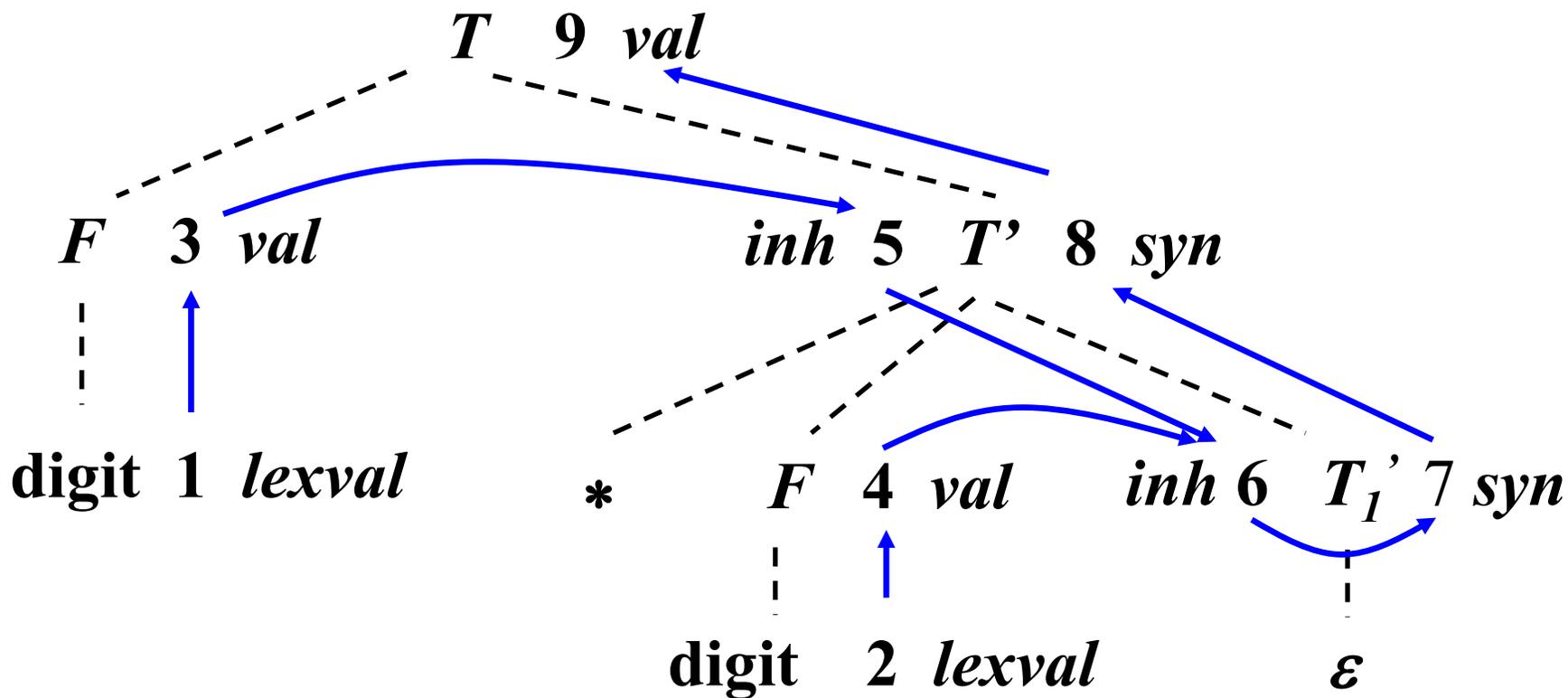


属性计算次序：举例



- 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



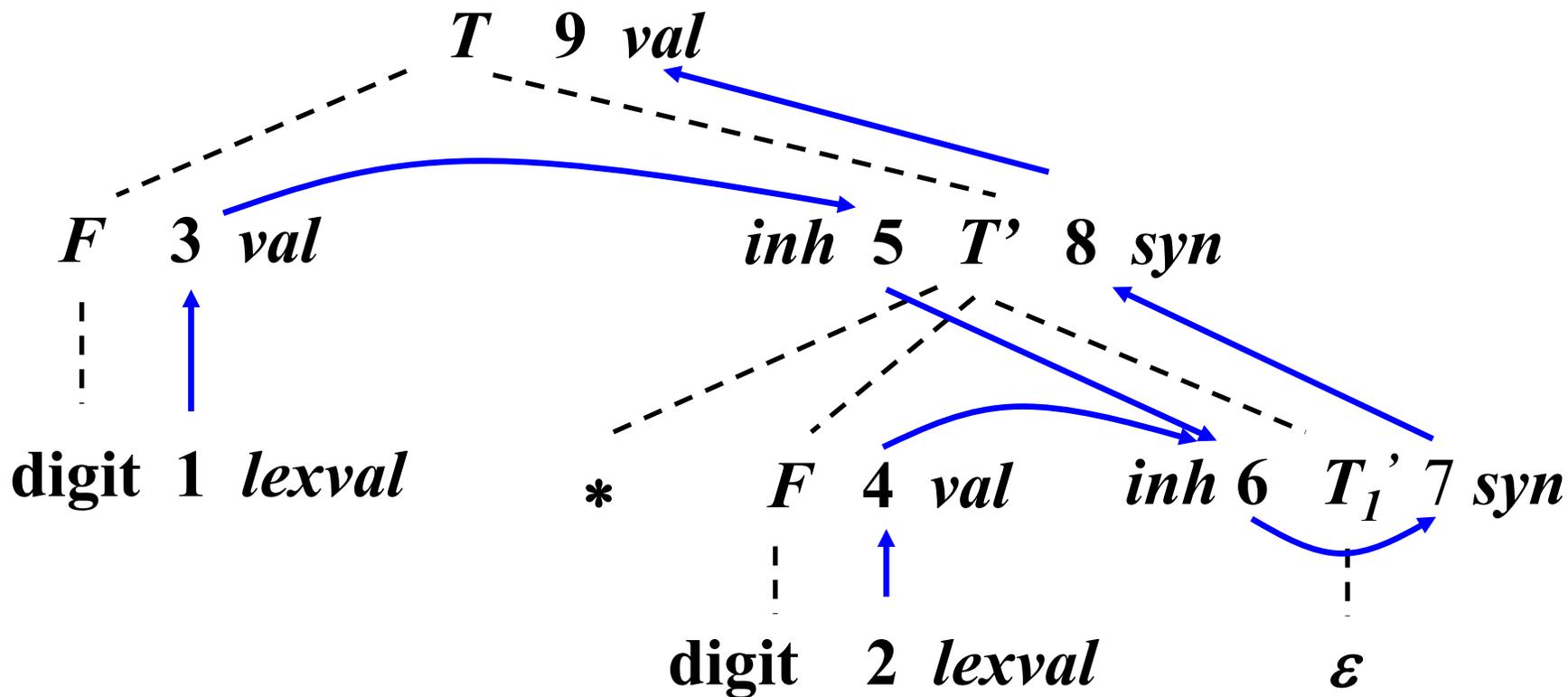


属性计算次序：举例



- 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





属性计算的问题



- 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



一起努力 打造国产基础软硬件体系!

李诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2024年9月30日