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 I


**计算机科学与技术学院**

**李 诚**


**15/10/2018**

# Announcement



- Midterm exam will be held on Nov 8th (Thursday), 3B201
- Openbook (only textbook, lecture notes, homework papers, no electronic devices)
- Tow hours
- Content covered
  - Lexical analysis
  - Syntax analysis

# Announcement



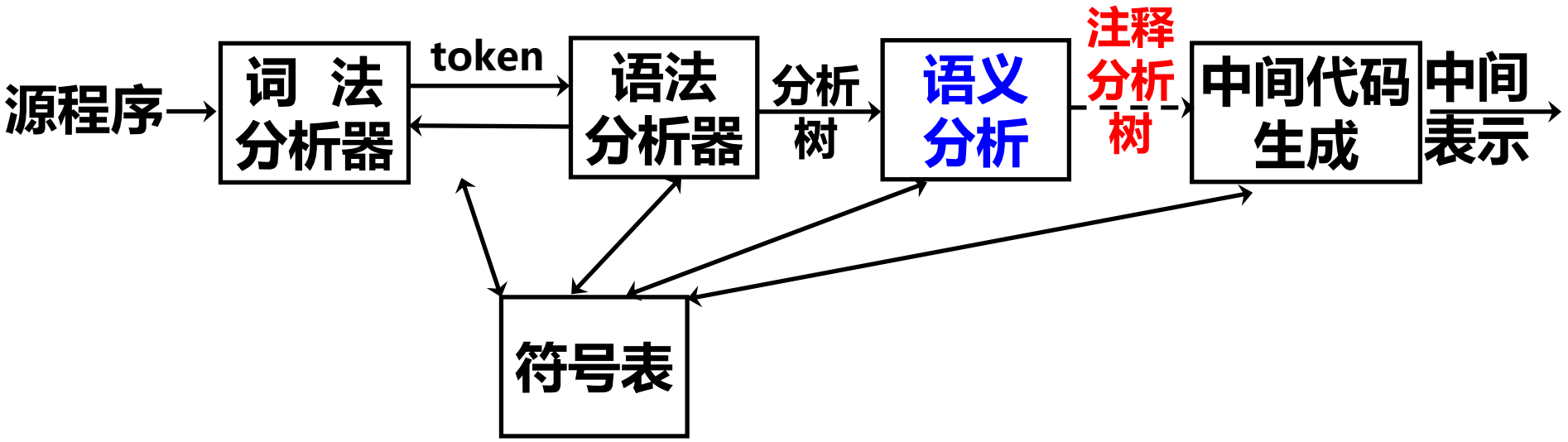
- Informal lecture: Advances in Systems Research

系统reading group

- Alias: USTC-SYS Reading Group
- http://210.45.114.146/wiki/doku.php?id=public:rg:reading group
- Welcome to join us!

# 主要内容






- ❑语法制导翻译简介
  - ❖语法制导定义、翻译方案
- ❑语法制导定义
  - ❖综合属性、继承属性
  - ❖属性依赖图、属性计算次序
  - ❖S属性的定义、L属性的定义

# 语法制导翻译简介



- 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
  - 语法分析、语义分析、中间代码生成可同时进行

## 语义分析的功能

- 审查每个语法结构的静态语义
  - 例：类型、运算、维数、越界

# 语义分析的功能

- 审查每个语法结构的静态语义
  - 例：类型、运算、维数、越界
- 在验证完静态语义后，才执行真正的翻译
  - 例：变量的存储分配
  - 例：表达式的求值
  - 例：语句的翻译（中间代码的生成）

- 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（语法规则）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

# 语义描述的方法



## ❑语法制导定义(Syntax-directed definition, SDD)

- ❖为每一个产生式写一个语义子程序，当该产生式获得匹配时，调用相应的语义子程序实现语义检查与翻译。

$E \rightarrow E_1 + T$ $\quad\quad$ $E.code = E_1.code \| T.code \| \text{‘+’}$

- ❖可读性好，更适于描述规范

| 产生式 | 语义规则 |
|---|---|
| $D \rightarrow T\ L$ | $L.inh = T.type$ |
| $T \rightarrow \text{int}$ | $T.type = int$ |
| $T \rightarrow \text{real}$ | $T.type = real$ |
| $L \rightarrow L_1, \text{id}$ | $L_1.inh = L.inh$ |
| … | … |

# 语义描述的方法



## ❑翻译方案(Translation scheme, SDT)

- ❖在产生式的右部的适当位置，插入相应的语义动作，按照分析的进程，执行遇到的语义动作。

  $E \rightarrow E_1 + T \quad \{ print\ '+' \}$

- ❖陈述了实现细节(如语义规则的计算时机)

$$D \rightarrow T \{ L.inh = T.type \} L$$

$$T \rightarrow \text{int} \{ T.type = int \}$$

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

$$L \rightarrow \{ L_1.inh = L.inh \} L_1, \text{id}$$

$$\ldots$$

## ❑SDD

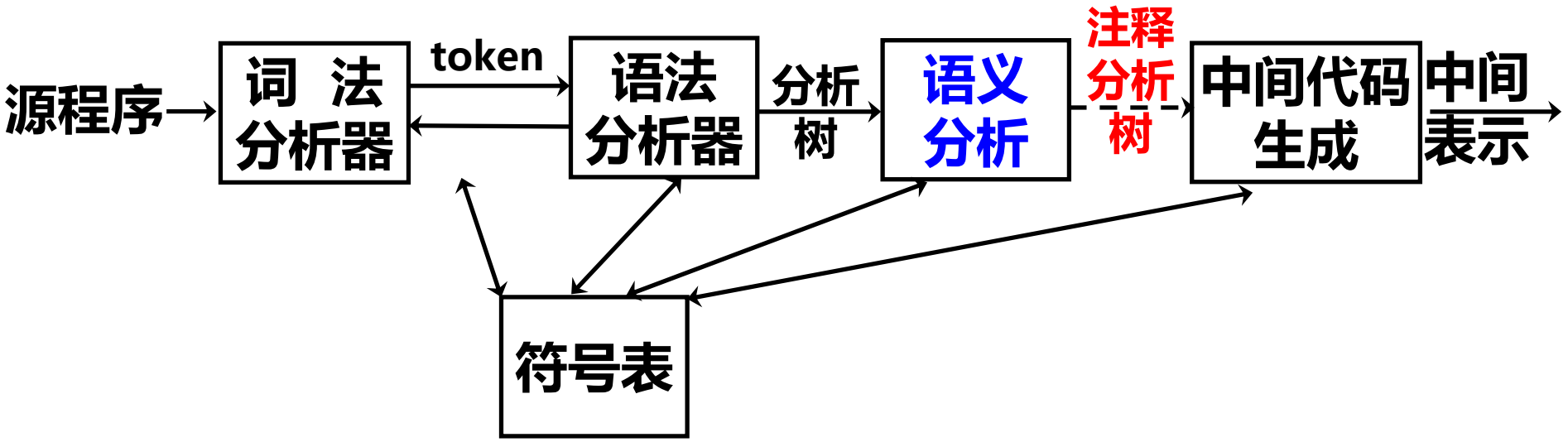
- ❖是关于语言翻译的高层次规格说明
- ❖隐蔽了许多具体实现细节，使用户不必显式地说明翻译发生的顺序

## ❑SDT

- ❖可以看作是对SDD的一种补充，是SDD的具体实施方案
- ❖显式地指明了语义规则的计算顺序，以便说明某些实现细节

# 主要内容






- ❑语法制导翻译简介
  - ❖语法制导定义、翻译方案
- ❑语法制导定义
  - ❖综合属性、继承属性
  - ❖属性依赖图、属性计算次序
  - ❖S属性的定义、L属性的定义

# 语法制导的定义

## ❑语法制导定义的形式

- ❖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

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

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

# 语法制导的定义

## ❑语法制导定义的形式

- ❖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
  $b=f(c_1, c_2, \ldots, c_k)$的语义规则，其中$f$是函数
  $b$和$c_1, c_2, \ldots, c_k$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，$c_1, c_2, \ldots, c_k$是产生式右部文法符号的属性或$A$的其它属性
- ❖继承属性(inherited attribute)：如果$b$是右部某文法符号$X$的属性

# 语法制导的定义



## ❑语法制导定义的形式

- ❖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

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

  $b$和$c_1, c_2, \ldots, c_k$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，$c_1, c_2, \ldots, c_k$是产生式右部文法符号的属性或$A$的其它属性
- ❖继承属性(inherited attribute)：如果$b$是右部某文法符号$X$的属性

**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 |
|---|---|
| $L \rightarrow E$ 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

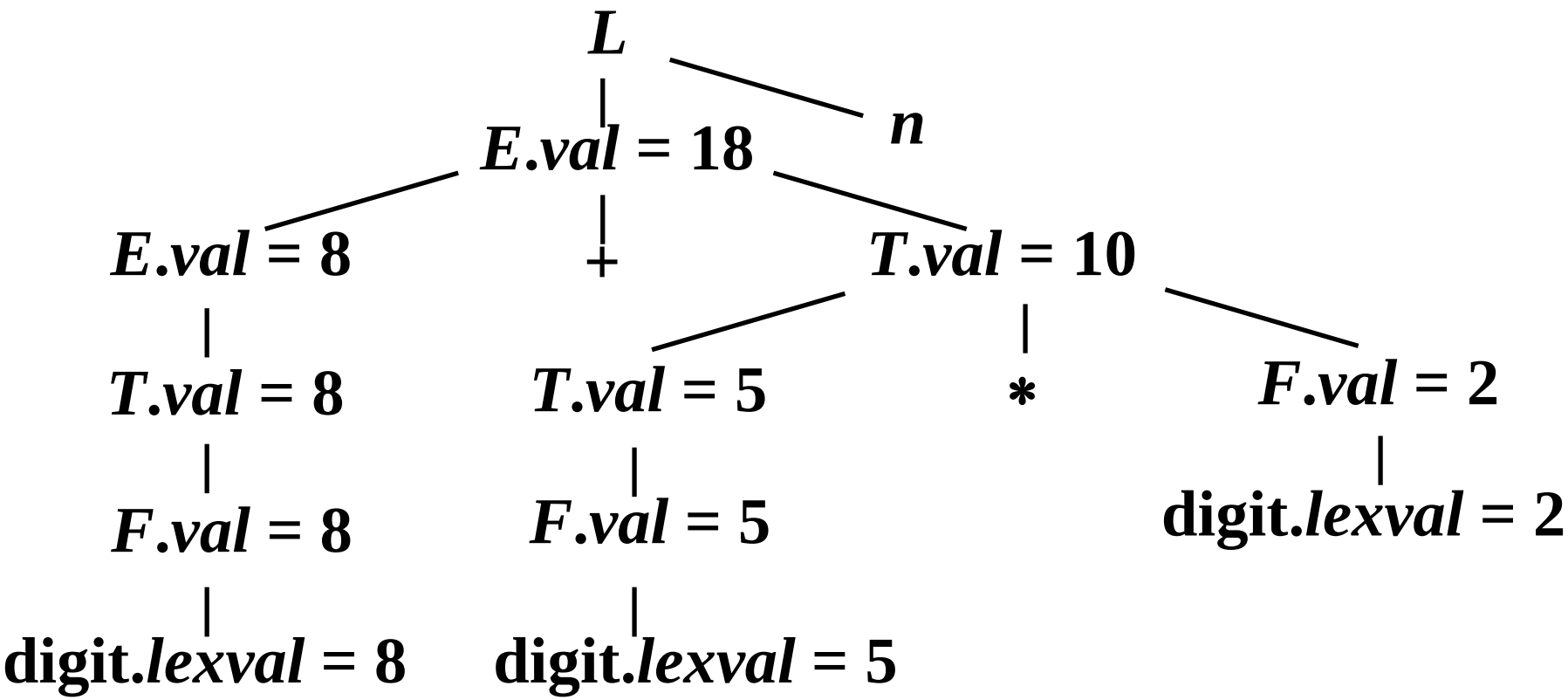
词法分析给定

# 注释分析树



- 定义：结点的属性值都标注出来的分析树

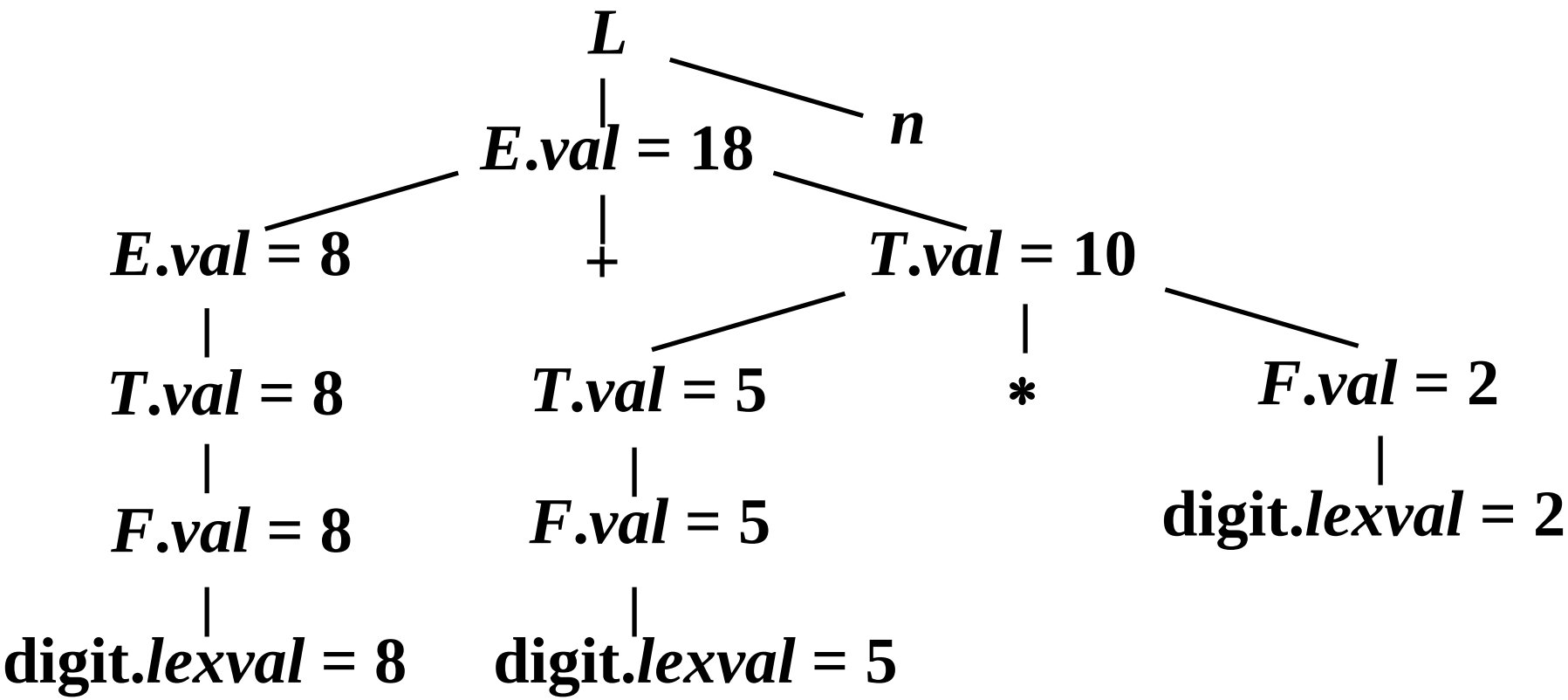
8+5*2 n的注释分析树



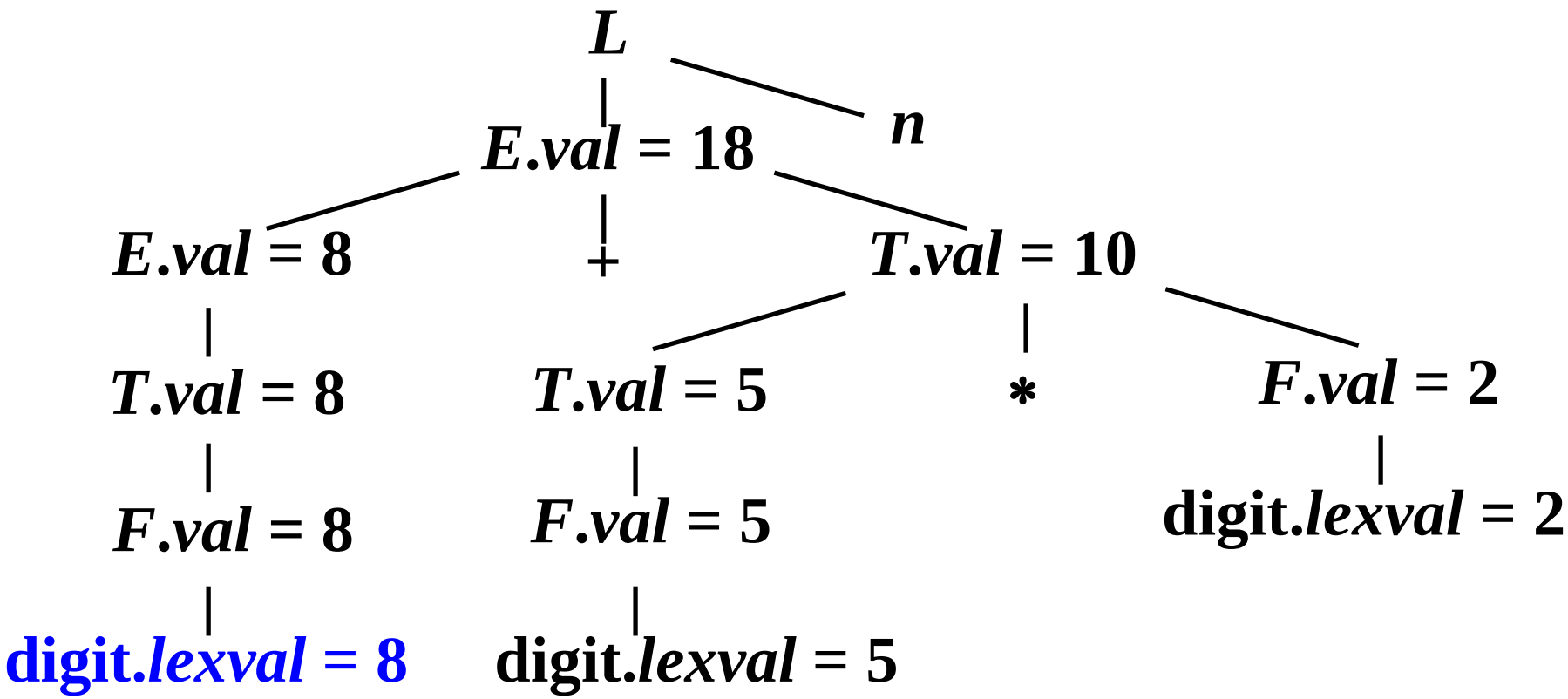
# 注释分析树+综合属性计算



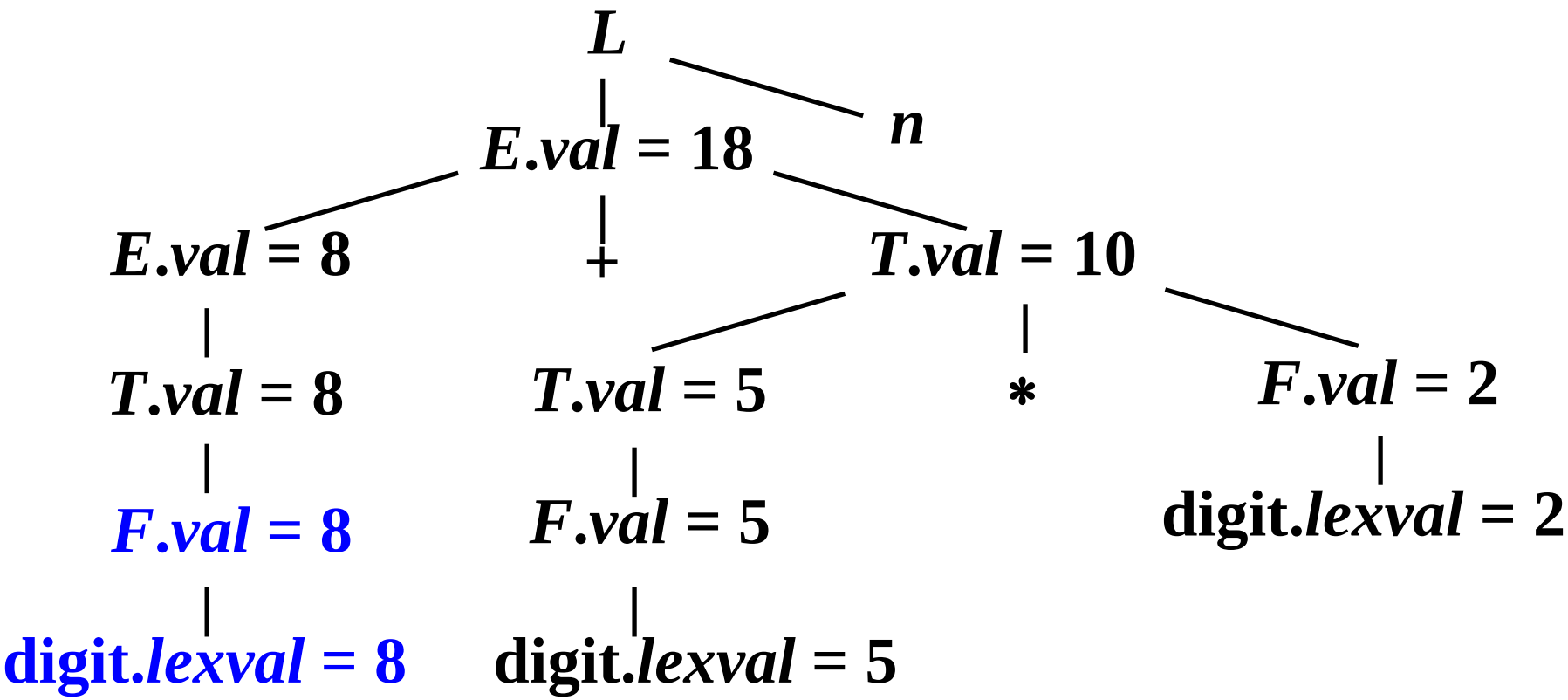
- 各结点综合属性的计算可以自底向上地完成



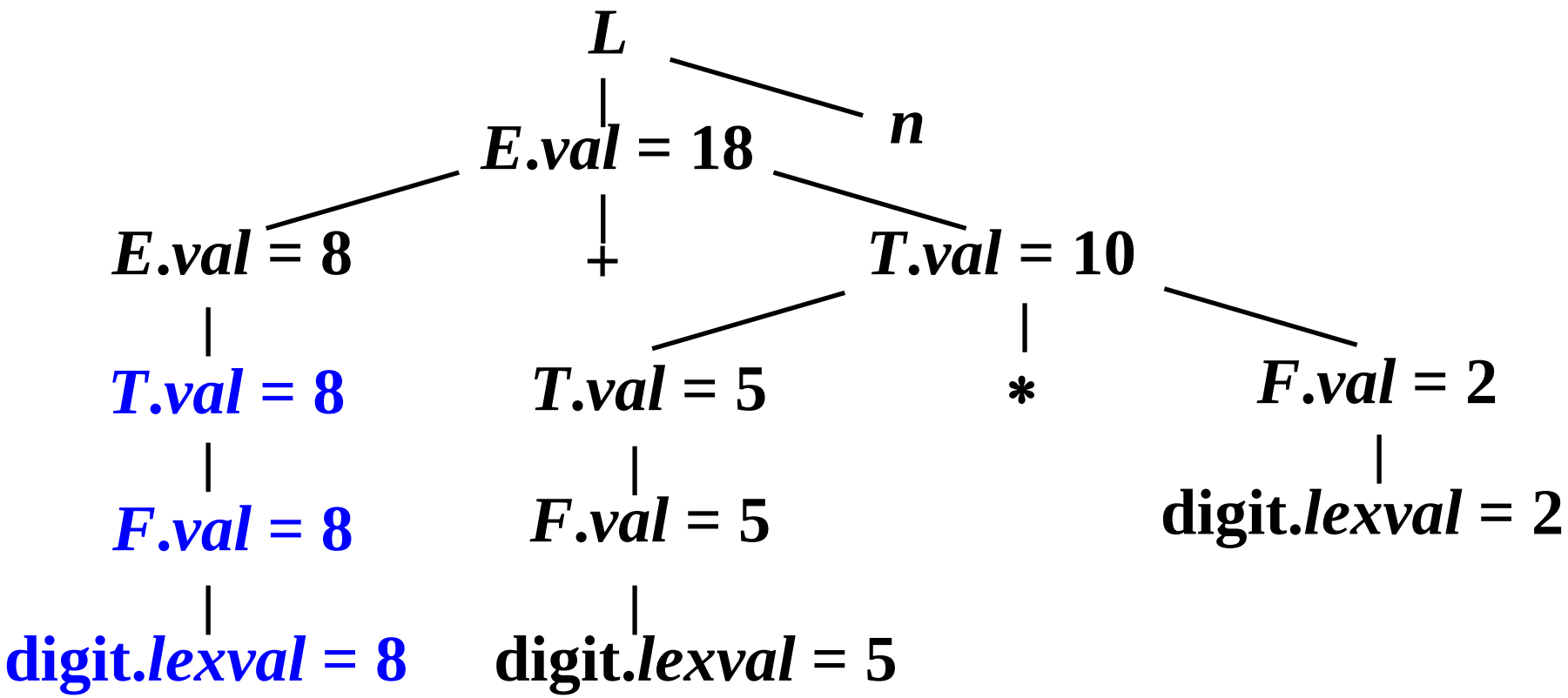
- 各结点综合属性的计算可以自底向上地完成



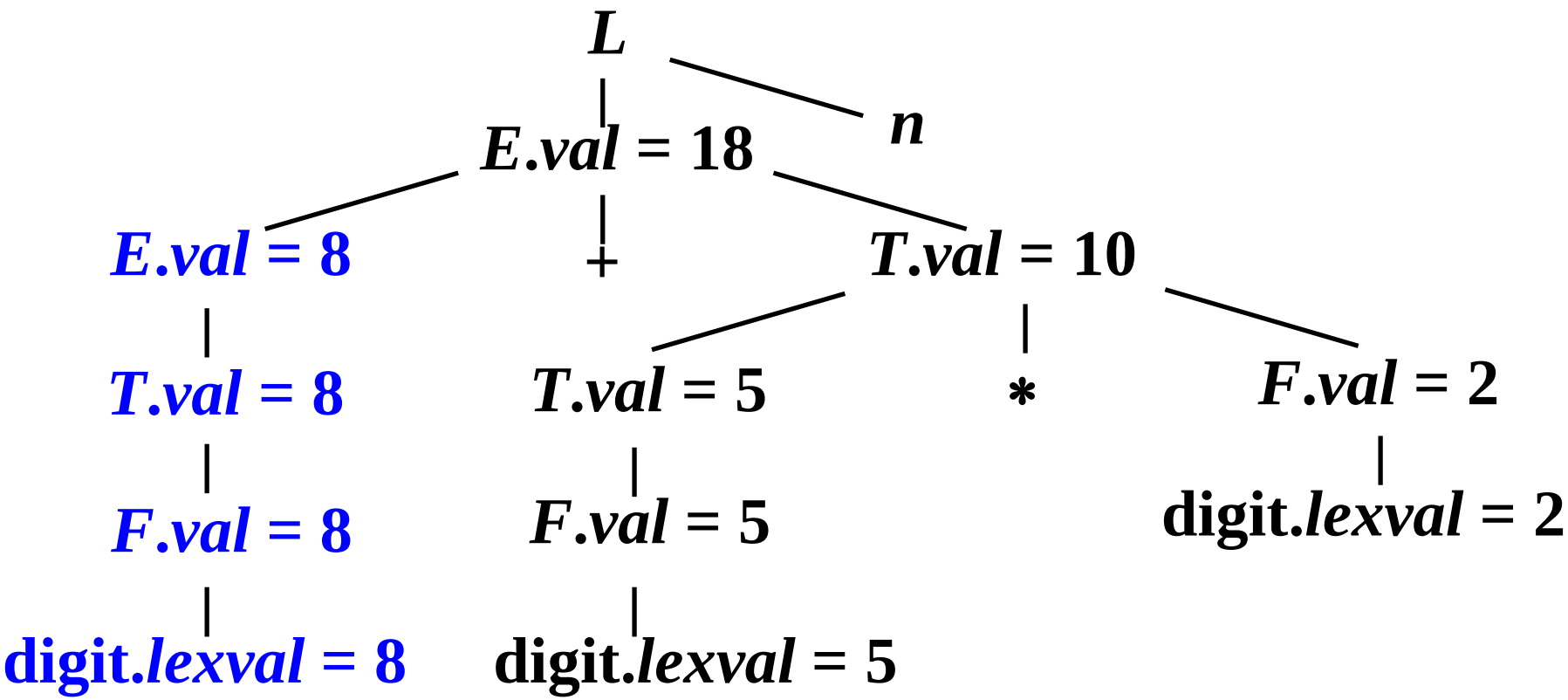
- 各结点综合属性的计算可以自底向上地完成



- 各结点综合属性的计算可以自底向上地完成



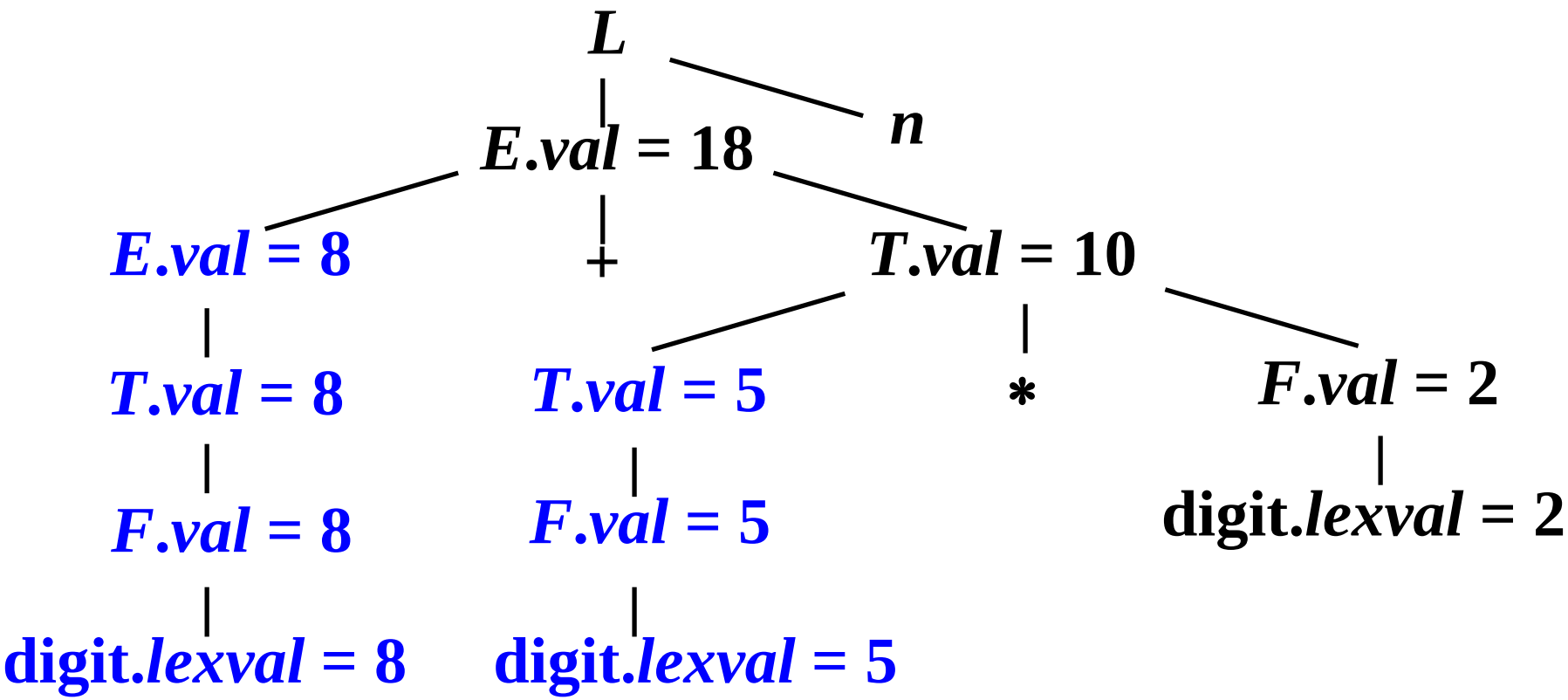
- 各结点综合属性的计算可以自底向上地完成



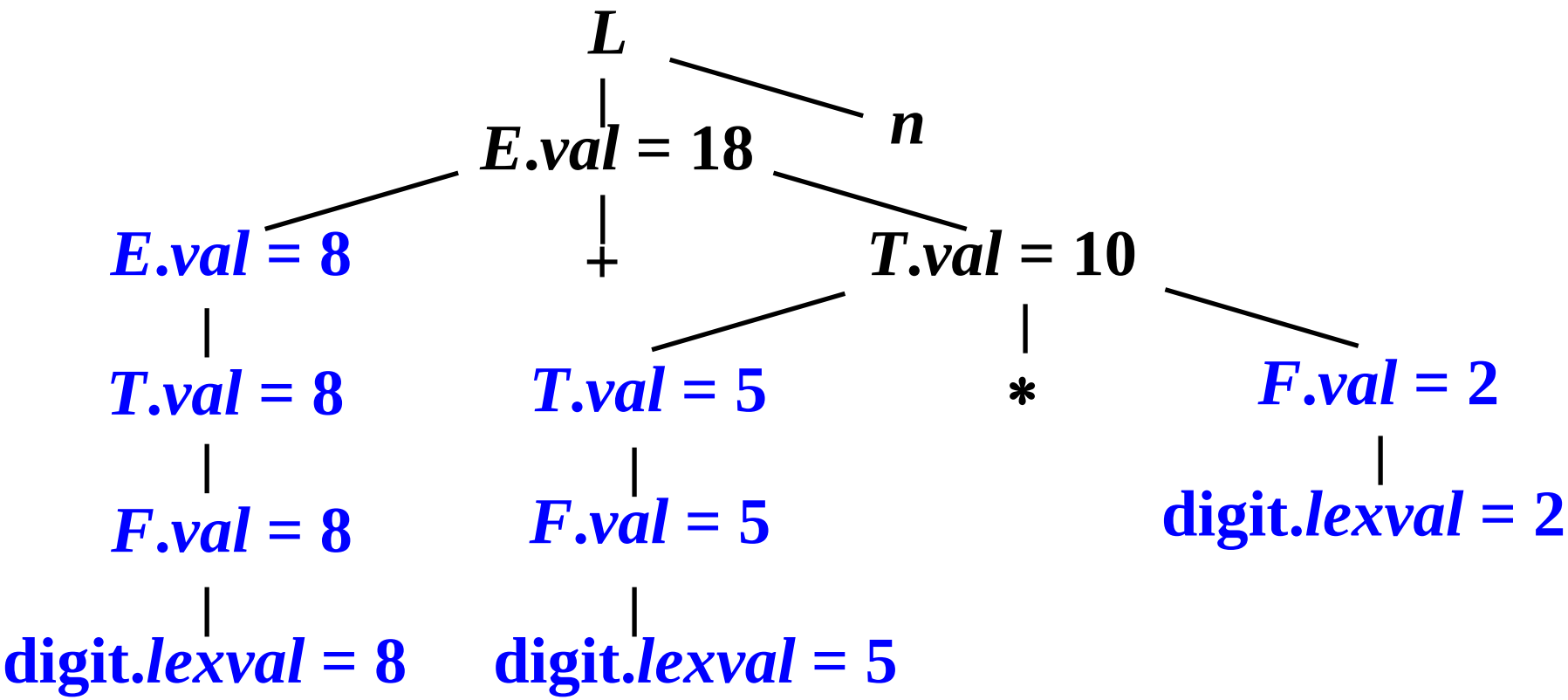
# 注释分析树+综合属性计算



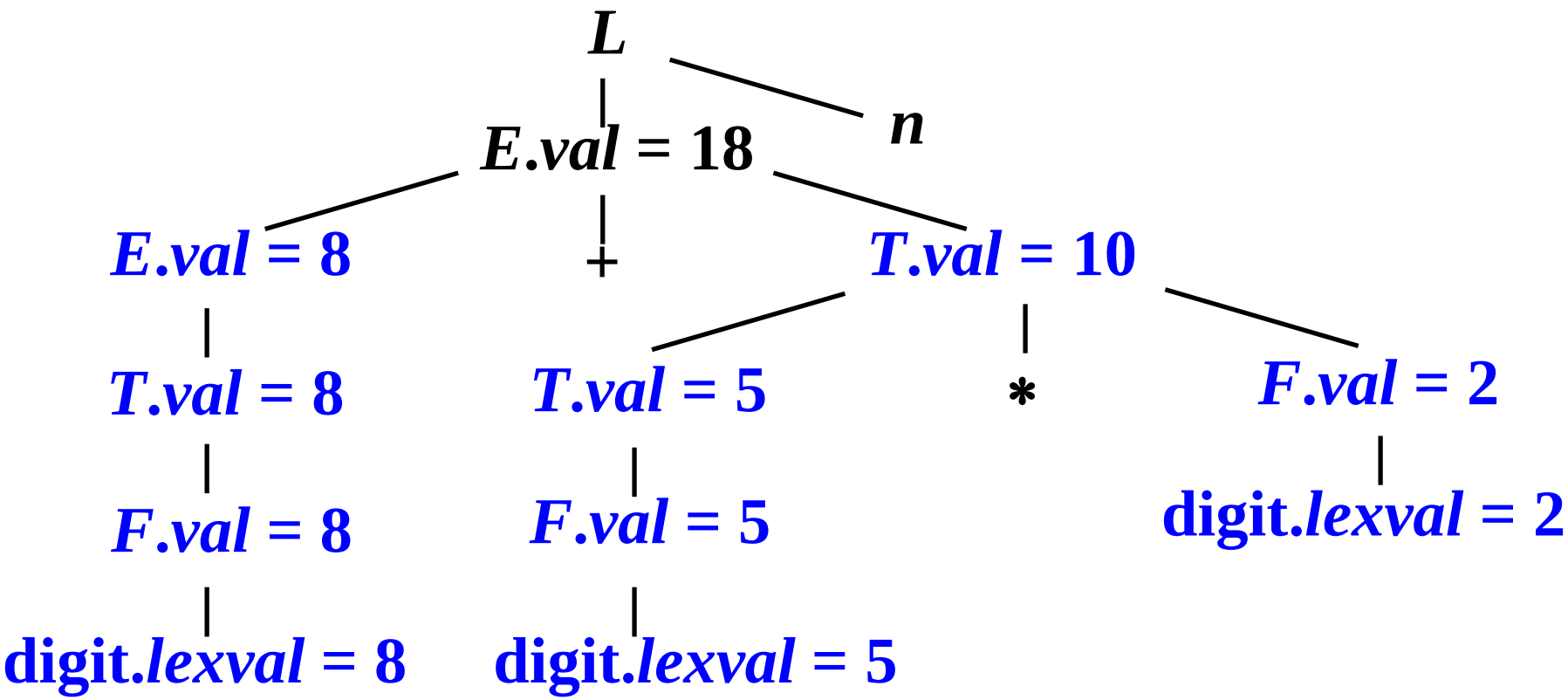
- 各结点综合属性的计算可以自底向上地完成



- 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

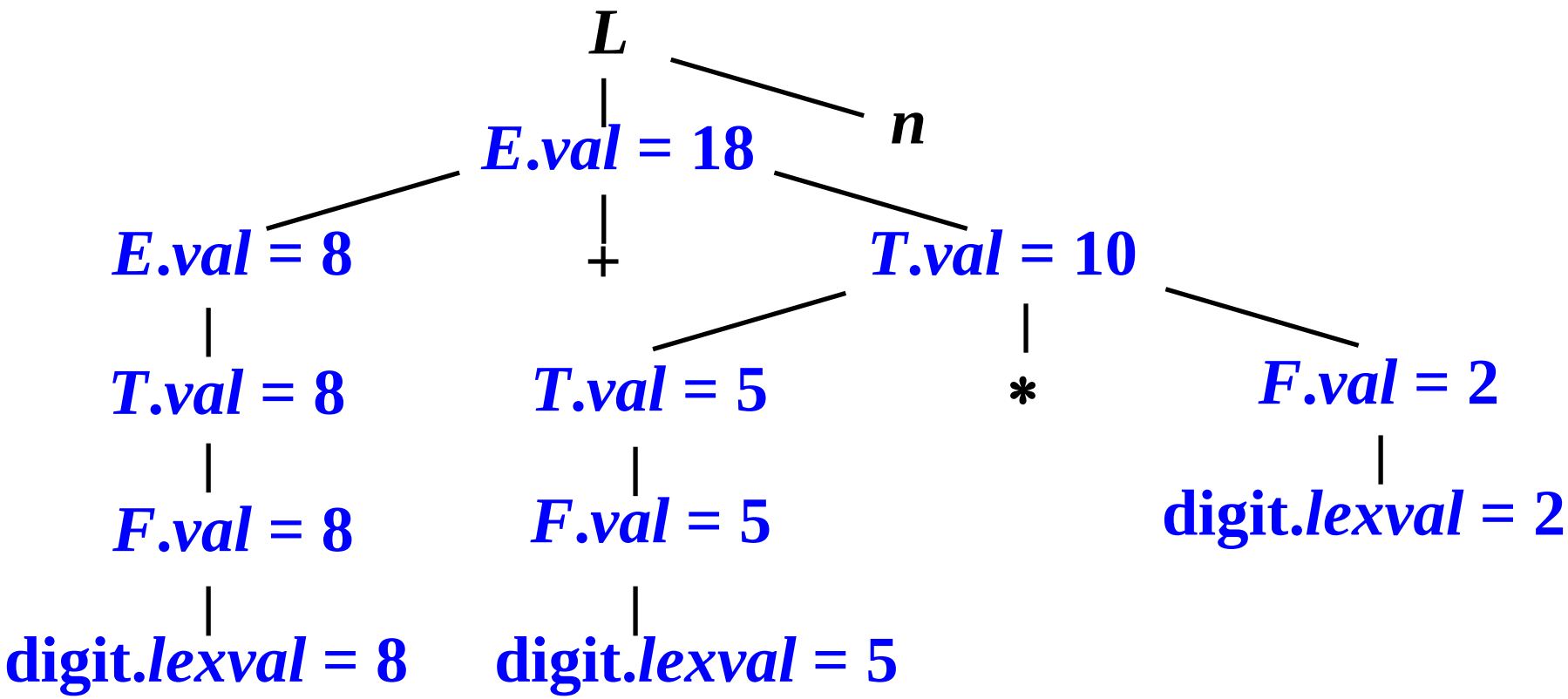


# 注释分析树+综合属性计算



- 各结点综合属性的计算可以自底向上地完成

**S属性的定义可以和LR分析器一起自然地实现。**



# 继承属性：举例



**int id, id, id**
**标识符声明**

| 产　生　式 | 语 义 规 则 |
|---|---|
| $D \rightarrow TL$ | $L.in = T.type$ |
| $T \rightarrow$ int | $T.type = integer$ |
| $T \rightarrow$ real | $T.type = real$ |
| $L \rightarrow L_1$, id | $L_1.in = L.in$;<br>$addType$(id.$entry$, $L.in$) |
| $L \rightarrow$ id | $addType$(id.$entry$, $L.in$) |

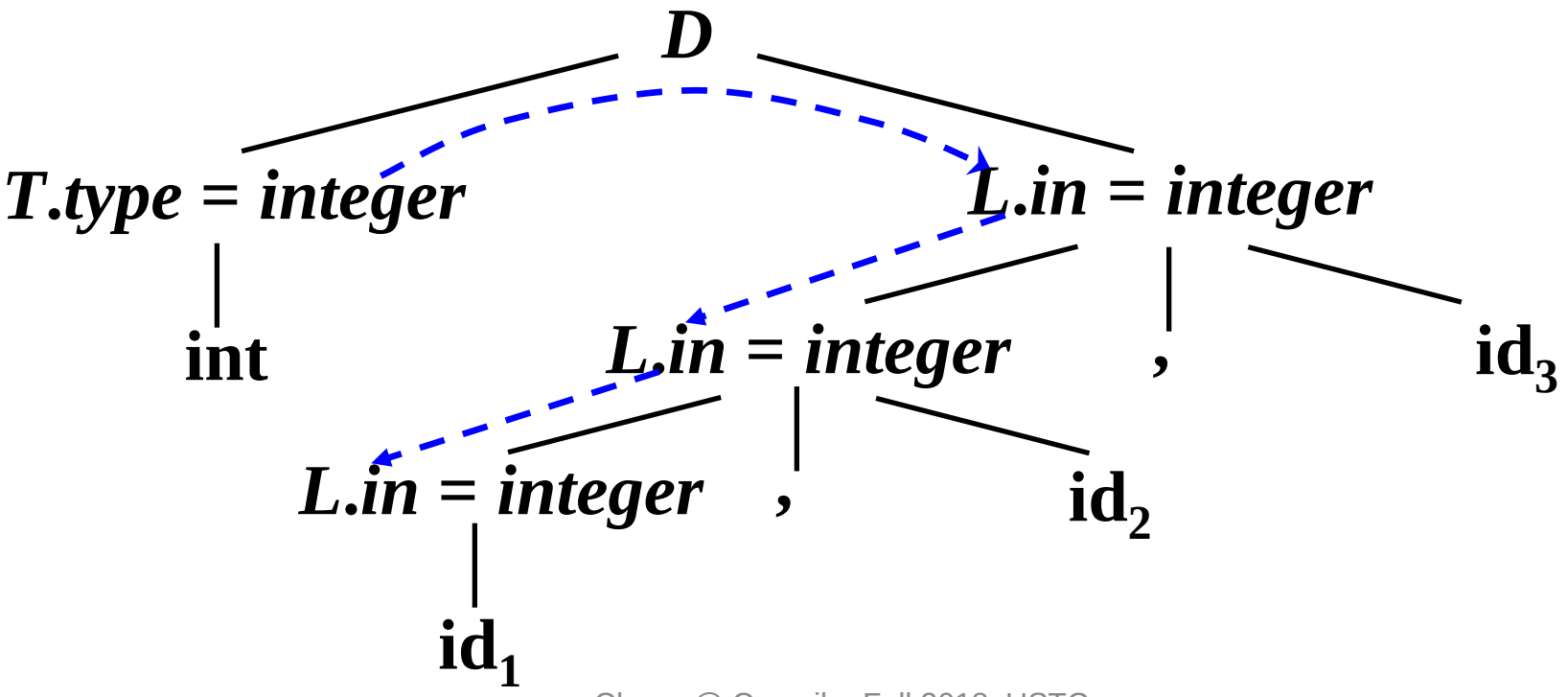
- *type* – *T*的综合属性
- *in* – *L*的继承属性，把声明的类型传递给标识符列表
- *addType*– 把类型信息加到符号表中的标识符条目里

# 继承属性：举例



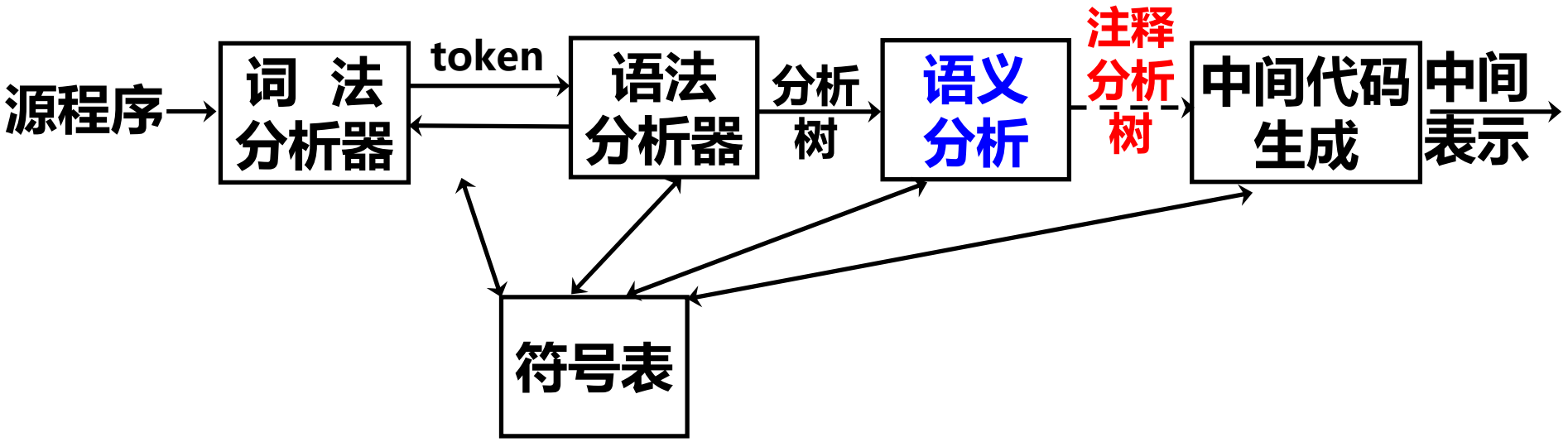
❑例 int $id_1$, $id_2$, $id_3$的标注了部分属性的分析树

不可能像综合属性那样自底向上标注属性



# 主要内容






- ❑语法制导翻译简介
  - ❖语法制导定义、翻译方案
- ❑语法制导定义
  - ❖综合属性、继承属性
  - ❖属性依赖图、属性计算次序
  - ❖S属性的定义、L属性的定义

# SDD的求值顺序



- 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

# 属性依赖图



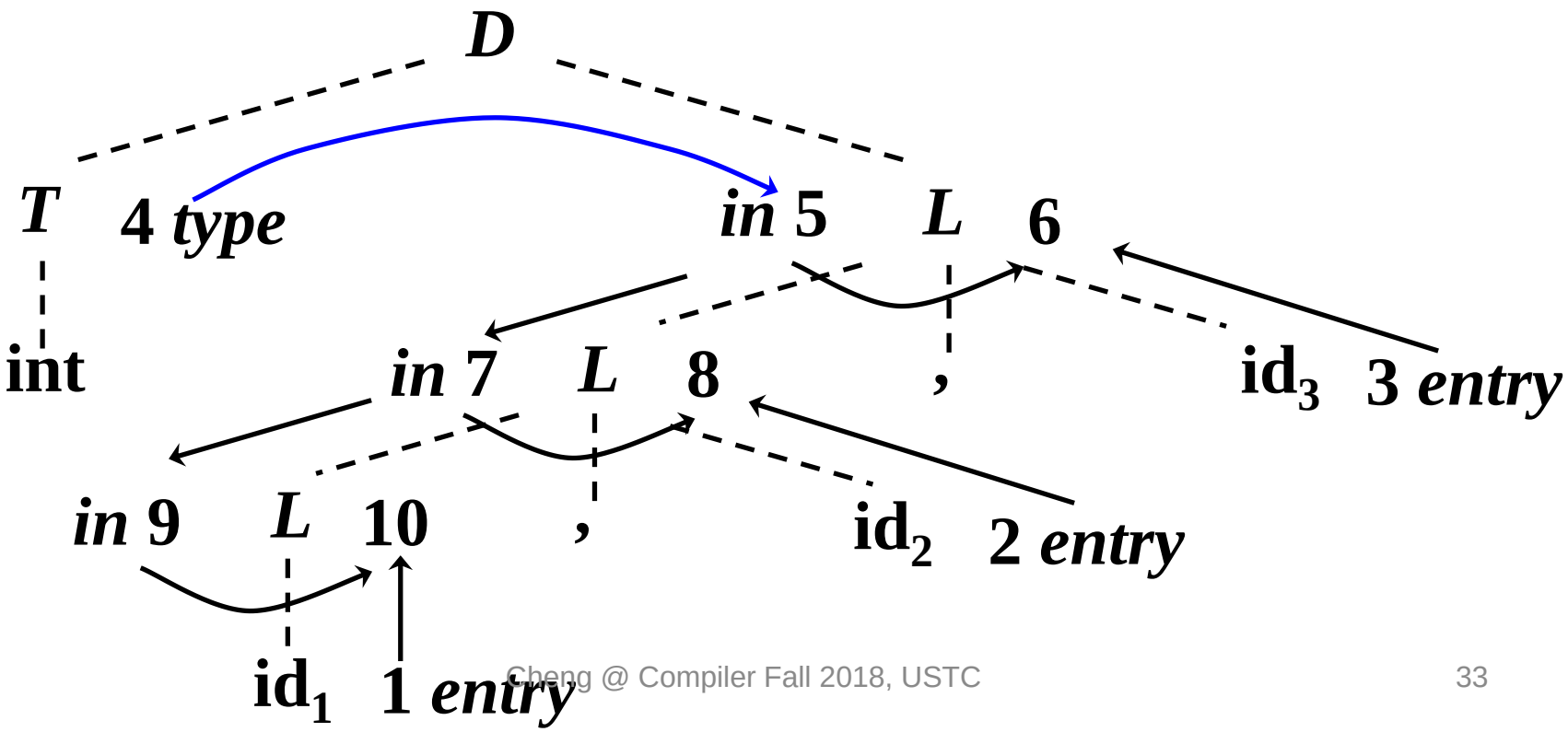
- 依赖图是一个描述了分析树中结点属性间依赖关系的有向图
  - 分析树中每个标号为*X*的结点的每个属性a都对应着依赖图中的一个结点
  - 如果属性*X.a*的值依赖于属性*Y.b*的值，则依赖图中有一条从*Y.b*的结点指向*X.a*的结点的有向边

# 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）

$D \rightarrow TL$　　$L.in = T.type$



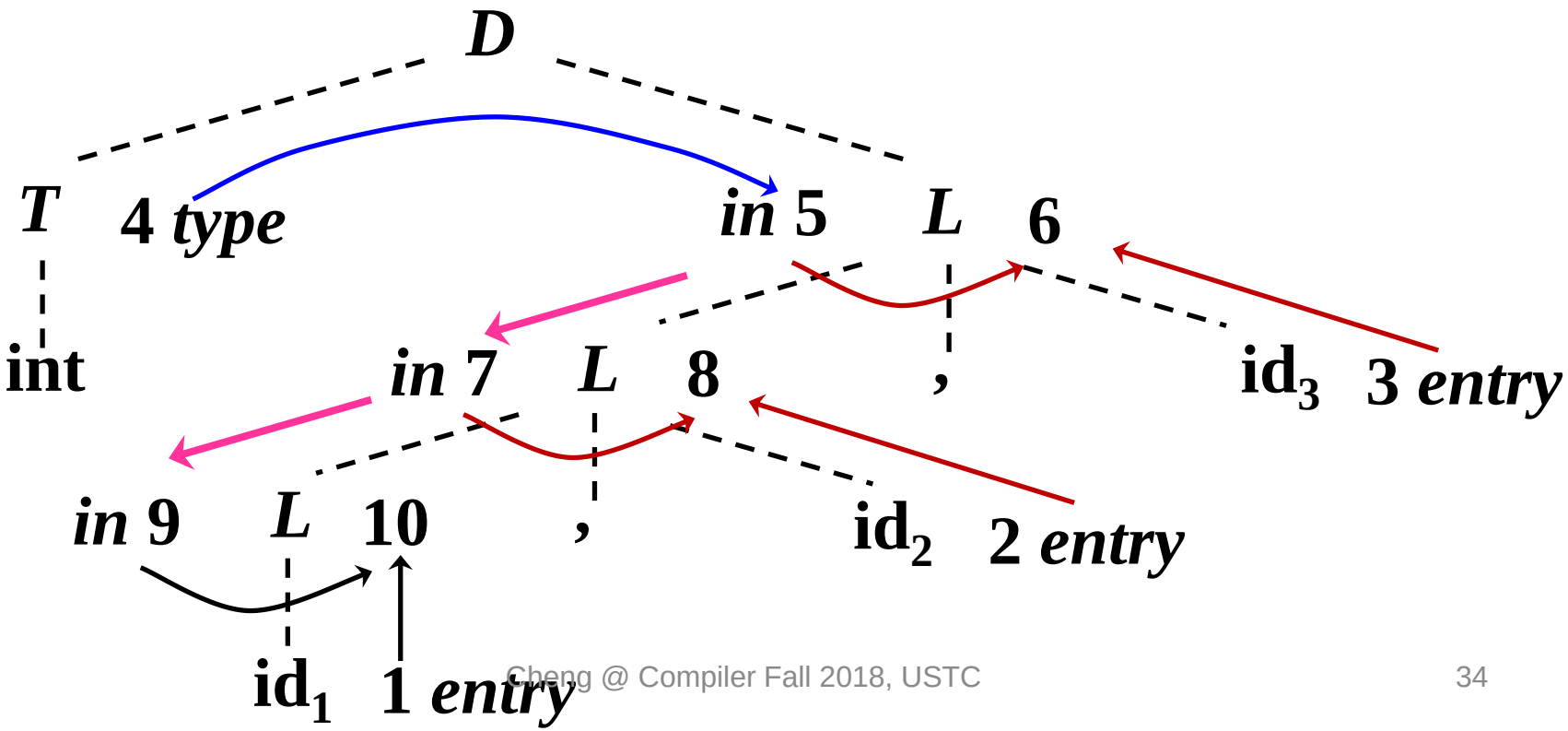
# 属性依赖图



❑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*)



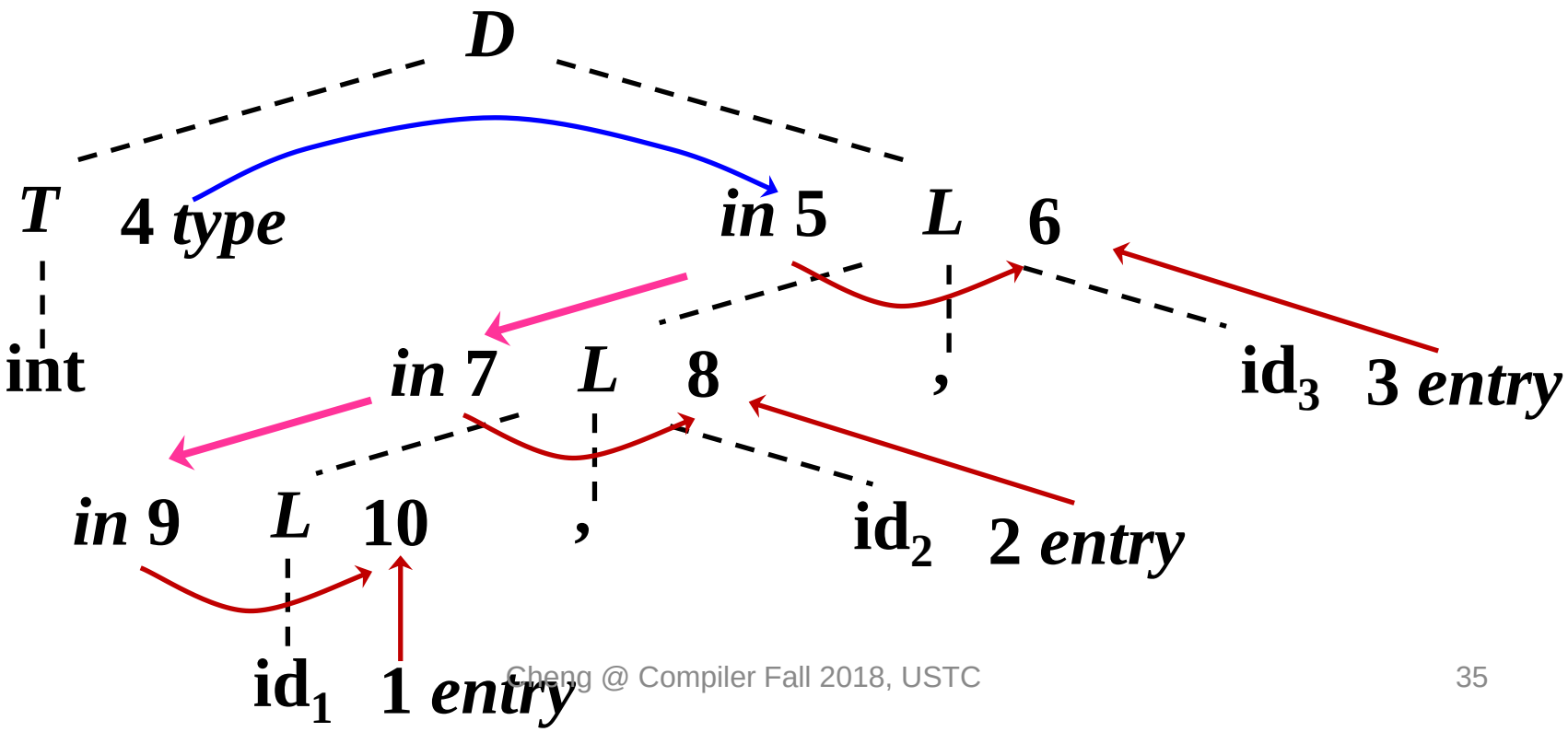
# 属性依赖图



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

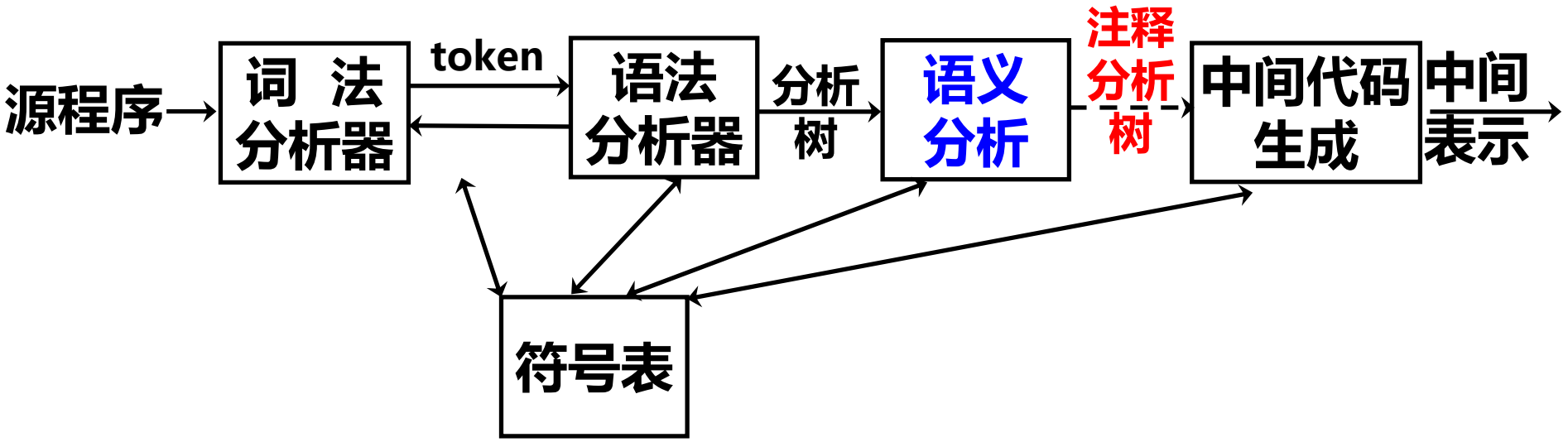
$L \rightarrow$ id

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



# 主要内容





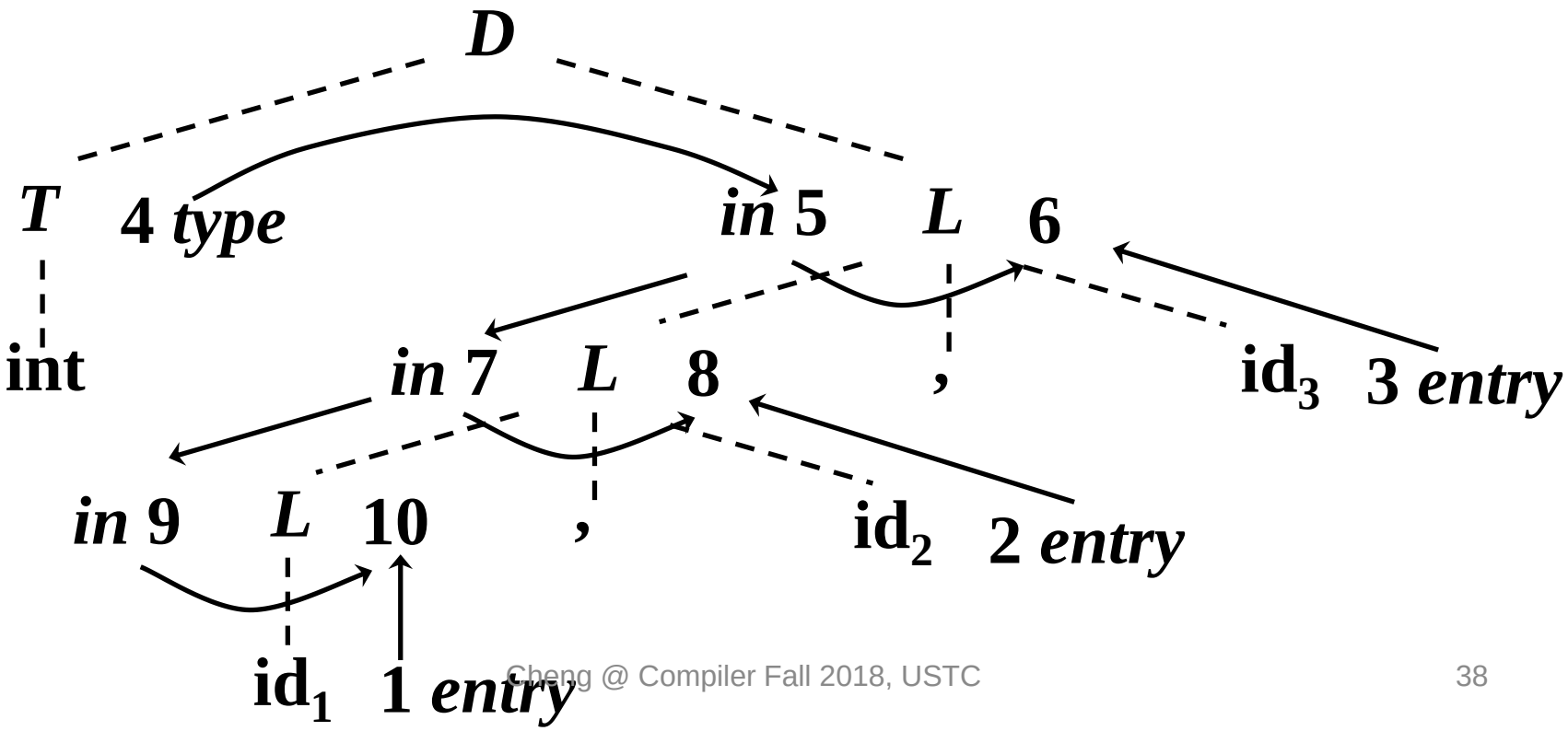

- ❑语法制导翻译简介
  - ❖语法制导定义、翻译方案
- ❑语法制导定义
  - ❖综合属性、继承属性
  - ❖属性依赖图、属性计算次序
  - ❖S属性的定义、L属性的定义

- 可行的求值顺序是满足下列条件的结点序列 $N_1, N_2, \ldots, N_k$：
  - 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$i < j$ (即：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



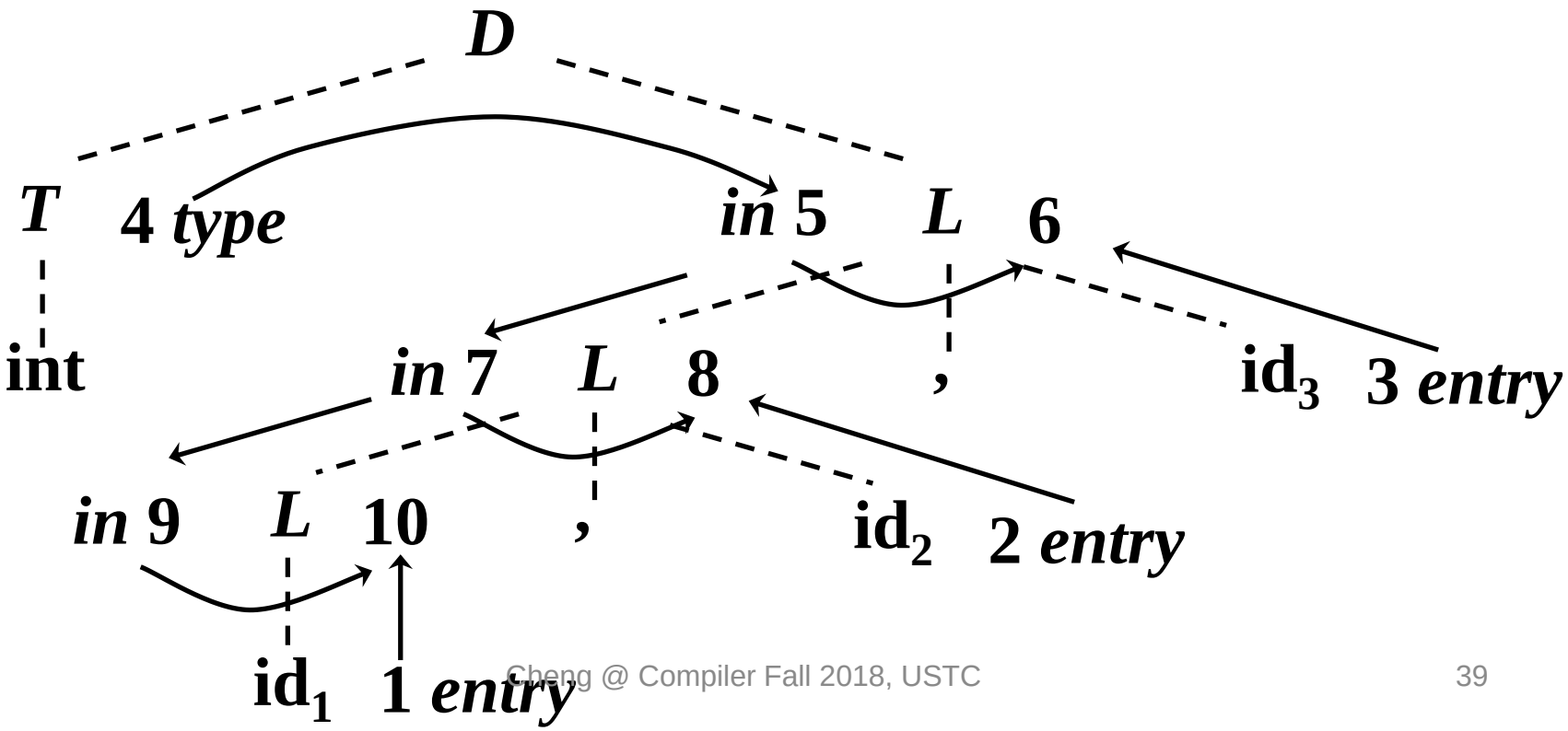
- 构造输入的分析树，构造属性依赖图，对结点进行拓扑排序，按拓扑排序的次序计算属性



# 属性计算次序：举例



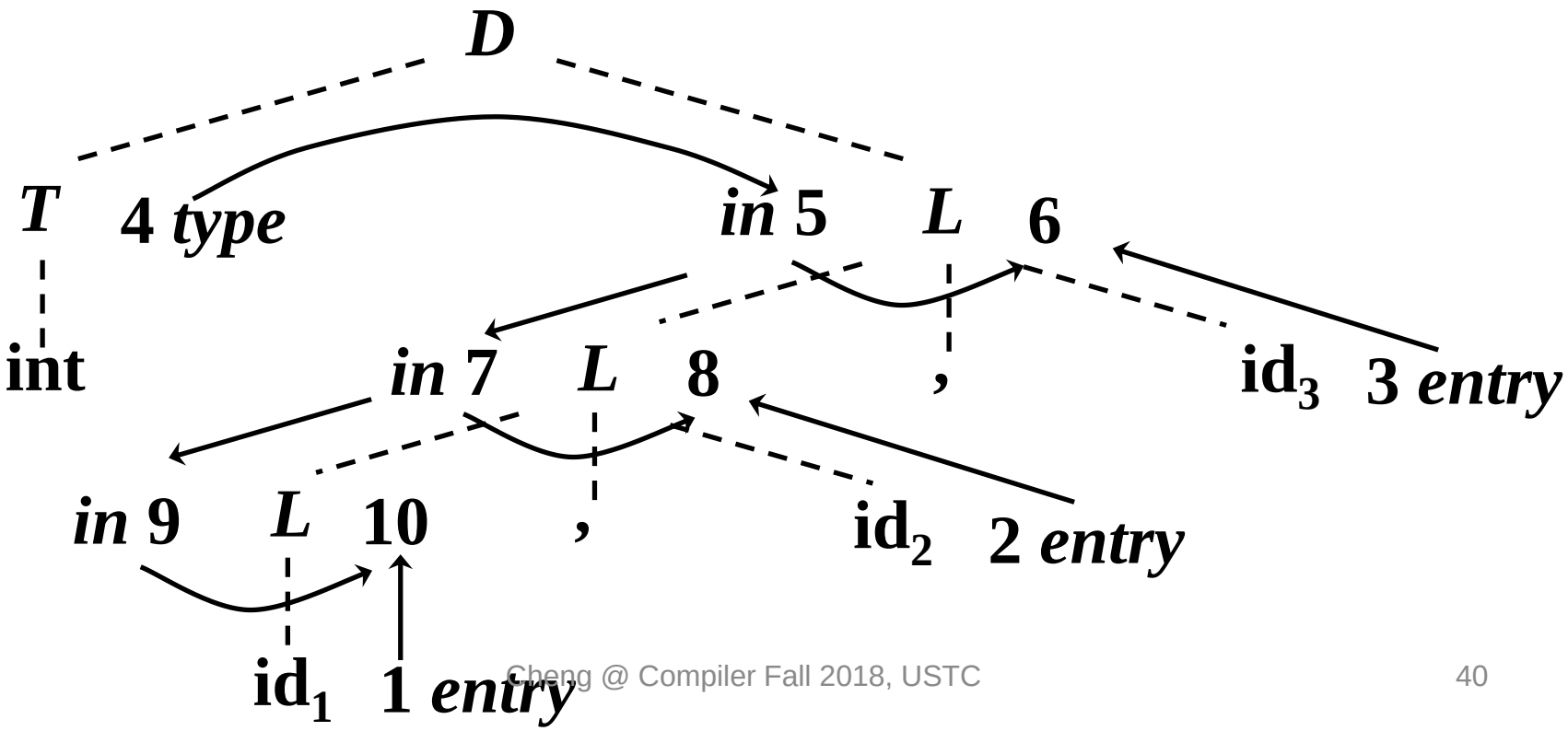
- 构造输入的分析树，构造属性依赖图，对结点进行拓扑排序，按拓扑排序的次序计算属性
  - 可行排序一：1, 2, 3, 4, 5, 6, 7, 8, 9, 10



# 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, 10
  - 可行排序二：4, 3, 2, 1, 5, 7, 6, 9, 8, 10



# 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?

# 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?
  - 当图中出现环的时候
  - SDD的属性之间存在循环依赖关系

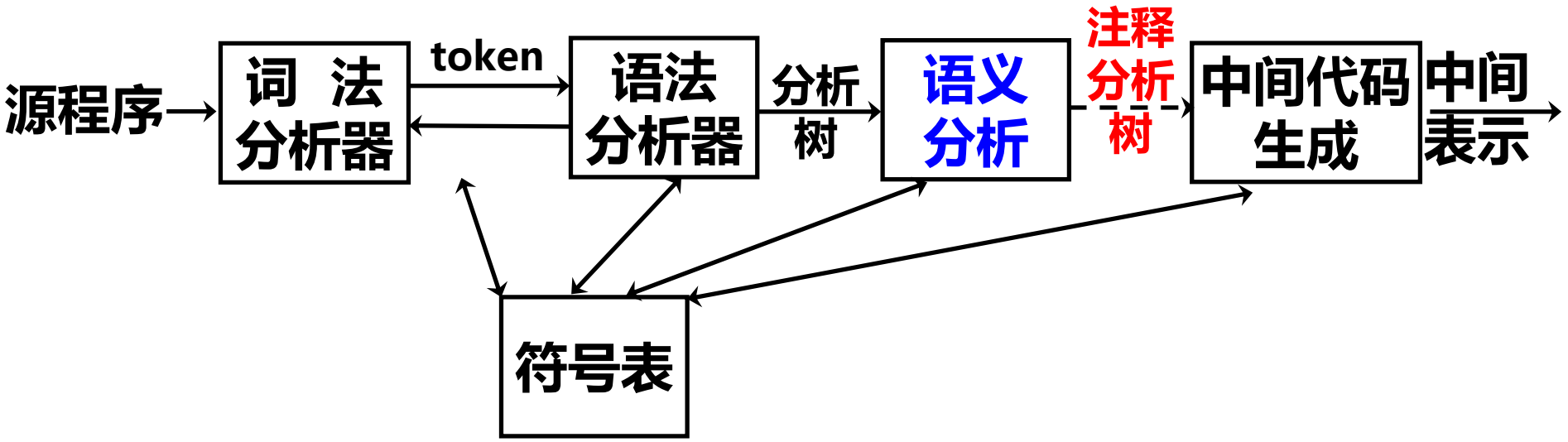
# 属性计算的问题



- 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?
  - 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

# 主要内容






- ❑语法制导翻译简介
  - ❖语法制导定义、翻译方案
- ❑语法制导定义
  - ❖综合属性、继承属性
  - ❖属性依赖图、属性计算次序
  - ❖S属性的定义、L属性的定义

# S属性的定义(S-SDD)



- 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

# S属性的定义(S-SDD)



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

# L属性的定义



- 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)

# L属性的定义



- 任意产生式$A \rightarrow X_1 X_2 \ldots X_n$，其右部符号$X_i (1 \le i \le n)$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Xi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$X_1, X_2, \ldots, X_{i-1}$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*

继承属性

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

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

# 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

# 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



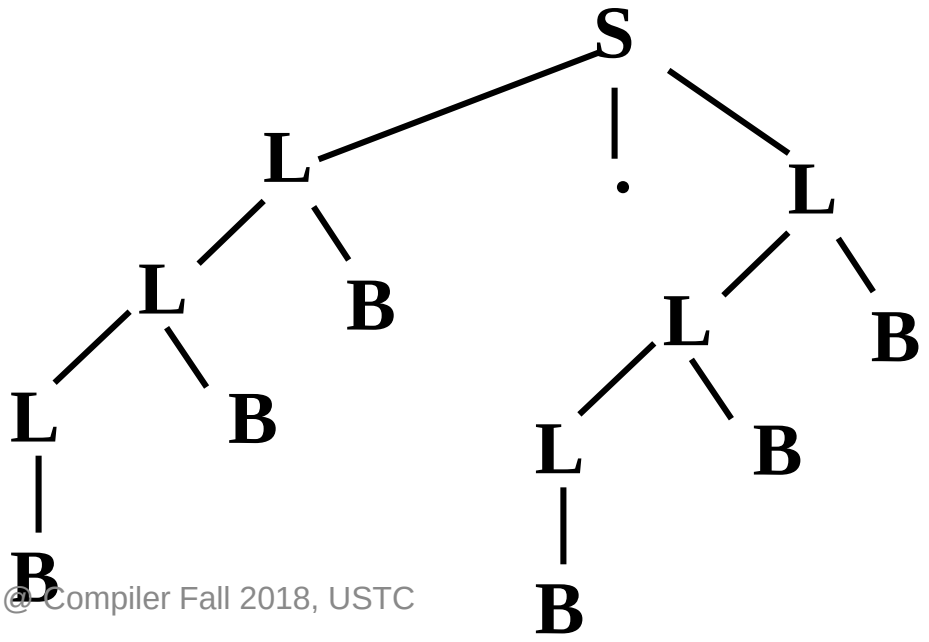
❑为下面文法写一个语法制导的定义，用S的综合属性*val*给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625（可以修改文法）

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

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

$L \rightarrow L\ B \mid B$

$B \rightarrow 0 \mid 1$



# 例题2



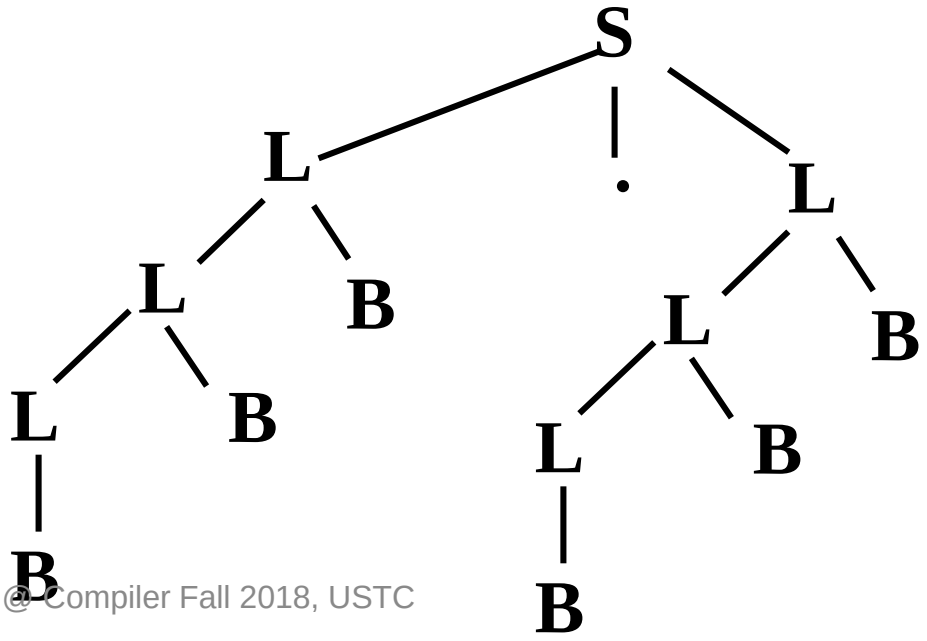
❑为下面文法写一个语法制导的定义，用S的综合属性*val*给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625（可以修改文法）

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

$S \rightarrow L \,.\, L \mid L$

$L \rightarrow L\ B \mid B$

$B \rightarrow 0 \mid 1$



# 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）

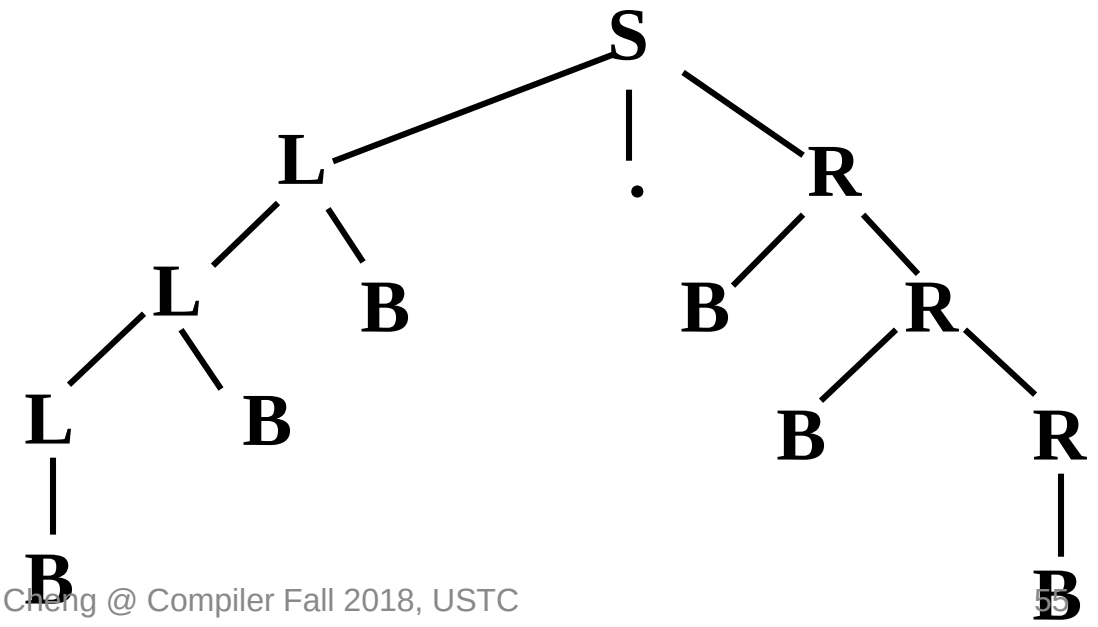
更清楚的办法是将文法改成下面的形式

$S \rightarrow L . R \mid L$

$L \rightarrow L B \mid B$

$R \rightarrow B R \mid B$

$B \rightarrow 0 \mid 1$



# 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$a * (b + c) * d$

**两种方法：**

- 先把表达式的括号都去掉，然后在必要的地方再加括号
- 去掉表达式中的冗余括号，保留必要的括号

## 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



$T \rightarrow F$

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

$F \rightarrow id$

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

$F \rightarrow (\ E\ )$

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

# 例题3



## 第二种方法

- 给$E$，$T$和$F$两个继承属性*left_op*和*right_op*分别表示左右两侧算符的优先级
- 给它们一个综合属性*self_op*表示自身主算符的优先级
- 再给一个综合属性*code*表示没有冗余括号的代码
- 分别用1和2表示加和乘的优先级，用3表示id和$(E)$的优先级，用0表示左侧或右侧没有运算对象的情况

# 例题3



$S' \rightarrow E$

$E.left\_op = 0; E.right\_op = 0; print(E.code)$

$E \rightarrow E_1 + T$

$E_1.left\_op = E.left\_op; E_1.right\_op = 1;$

$T.left\_op = 1; T.right\_op = E.right\_op;$

$E.code = E_1.code \| \text{“+”} \| T.code ; E.self\_op = 1;$

$E \rightarrow T$

$T.left\_op = E.left\_op;$

$T.right\_op = E.right\_op;$

$E.code = T.code;\ E.self\_op = T.self\_op$

# 例题3



$T \rightarrow T_1 * F \quad \ldots$

$T \rightarrow F \quad \ldots$

$F \rightarrow \mathbf{id}$

*F. code = id. lexeme; F. self_op* = 3

# 例题3



$F \rightarrow ( E )$

*E. left_op* = 0; *E. right_op* = 0;

*F. self_op* =

if (*F. left_op* < *E. self_op*) and

(*E. self_op* >= *F. right_op*)

then *E. self_op* else 3

*F. code* =

if (*F. left_op* < *E. self_op*) and

(*E. self_op* >= *F. right_op*)

then *E. code* else “(” || *E. code* || “)”

# 《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***