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

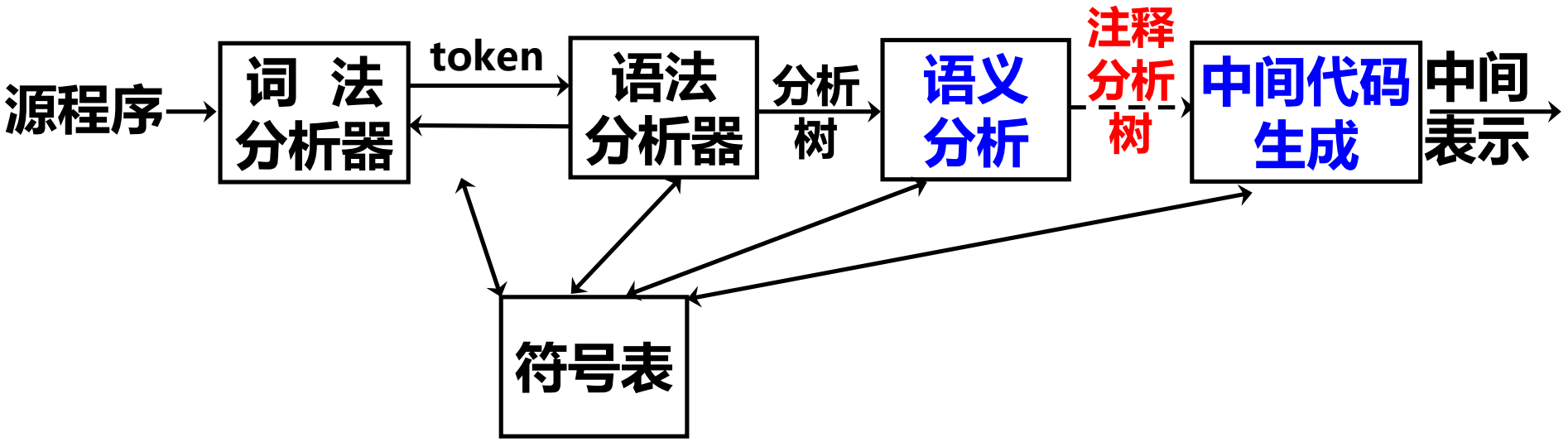
李 诚

9/10/2019

通知

- 10月14日课上，除了理论部分外，还将对第三部分实验的相关内容进行讲解
  - 主要是关于LLVM IR的介绍

主要内容



- 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;
```

语法制导翻译技术的应用

- 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
```

语法制导翻译技术的应用

- 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

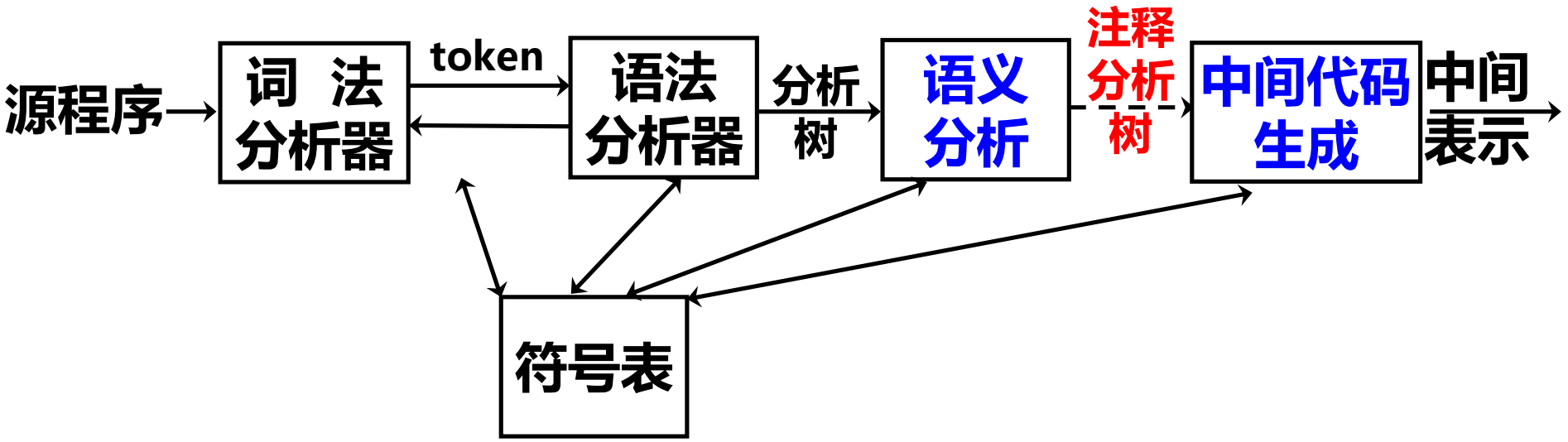
❑如何表示语义信息?

- ❖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

主要内容



- 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

语法制导的定义

❑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$A \to \alpha$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 (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

  $b=f(c_1, c_2, \ldots, c_k)$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

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

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?

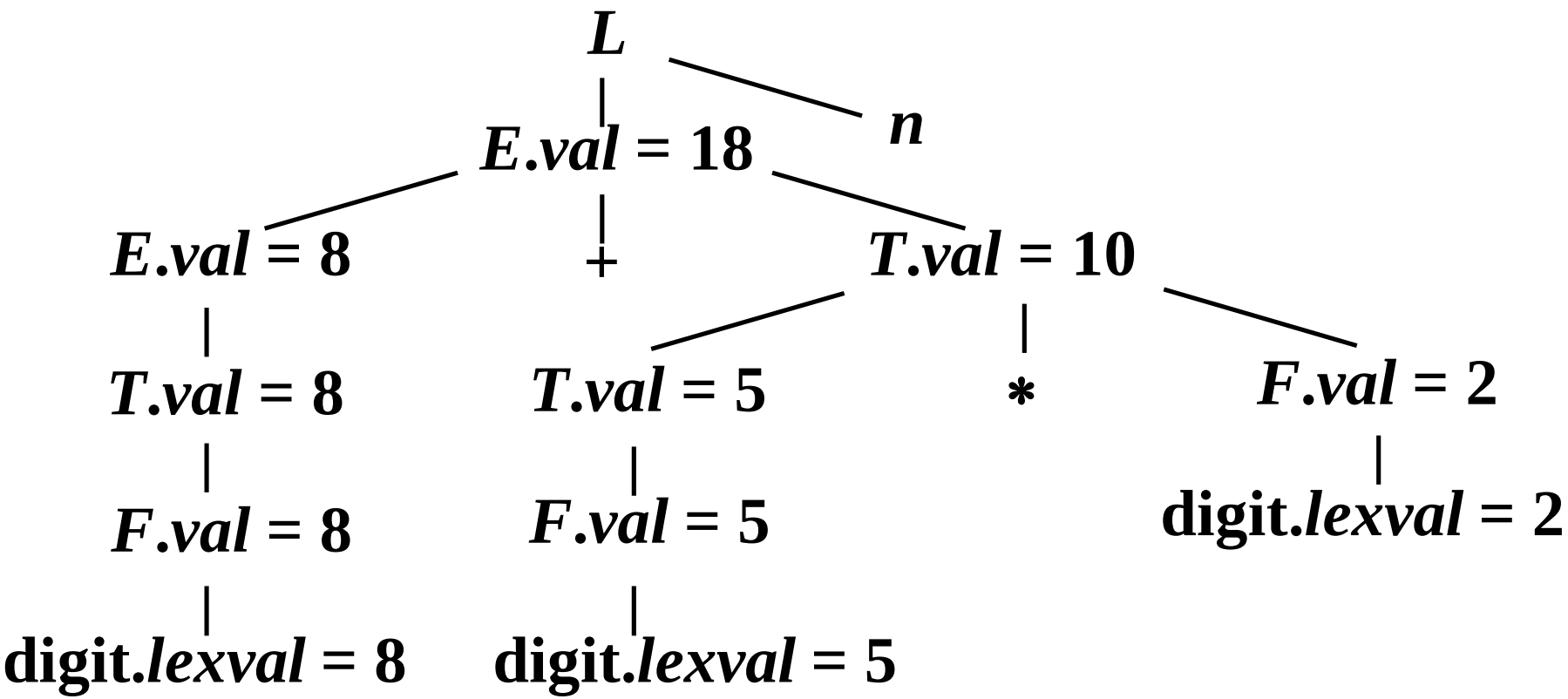
属性的计算

- 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?
- 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)

注释分析树

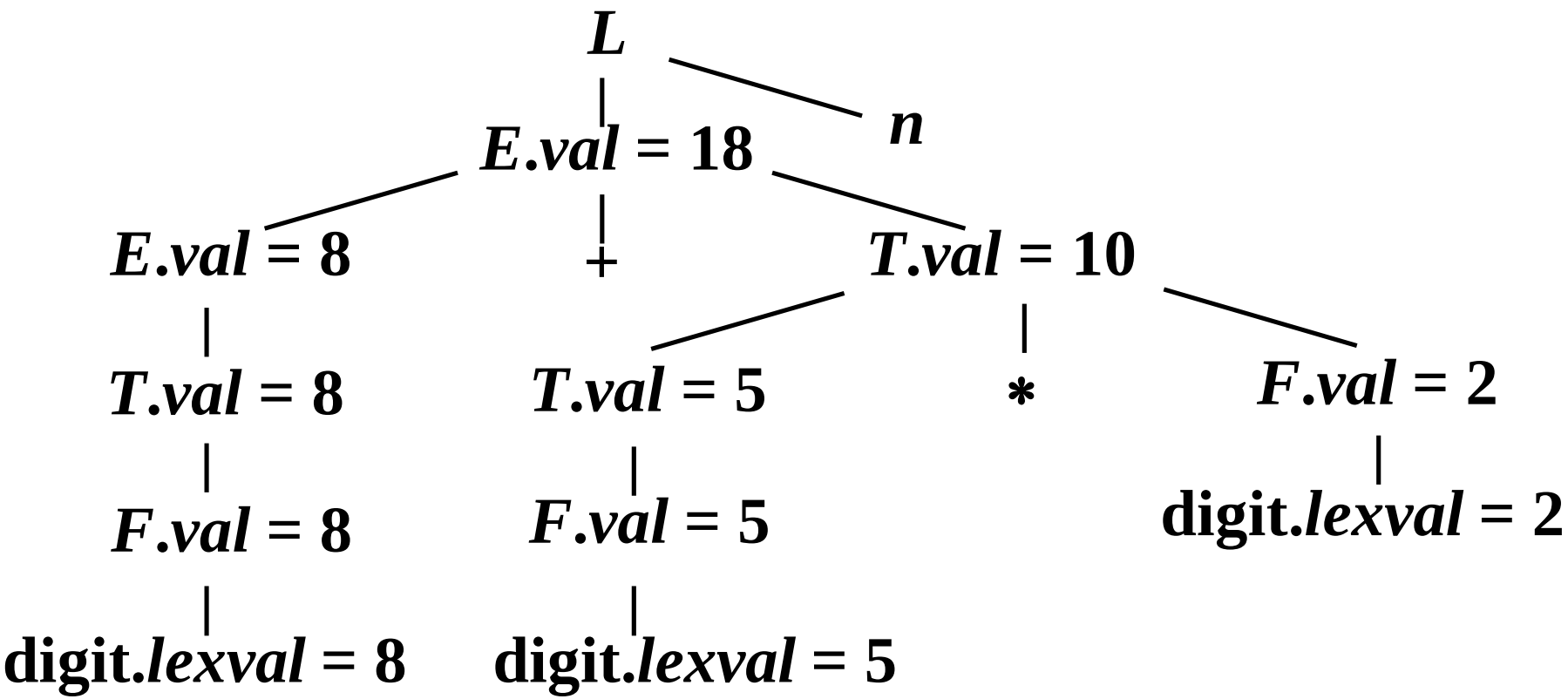
- 定义：结点的属性值都标注出来的分析树

8+5*2 n的注释分析树



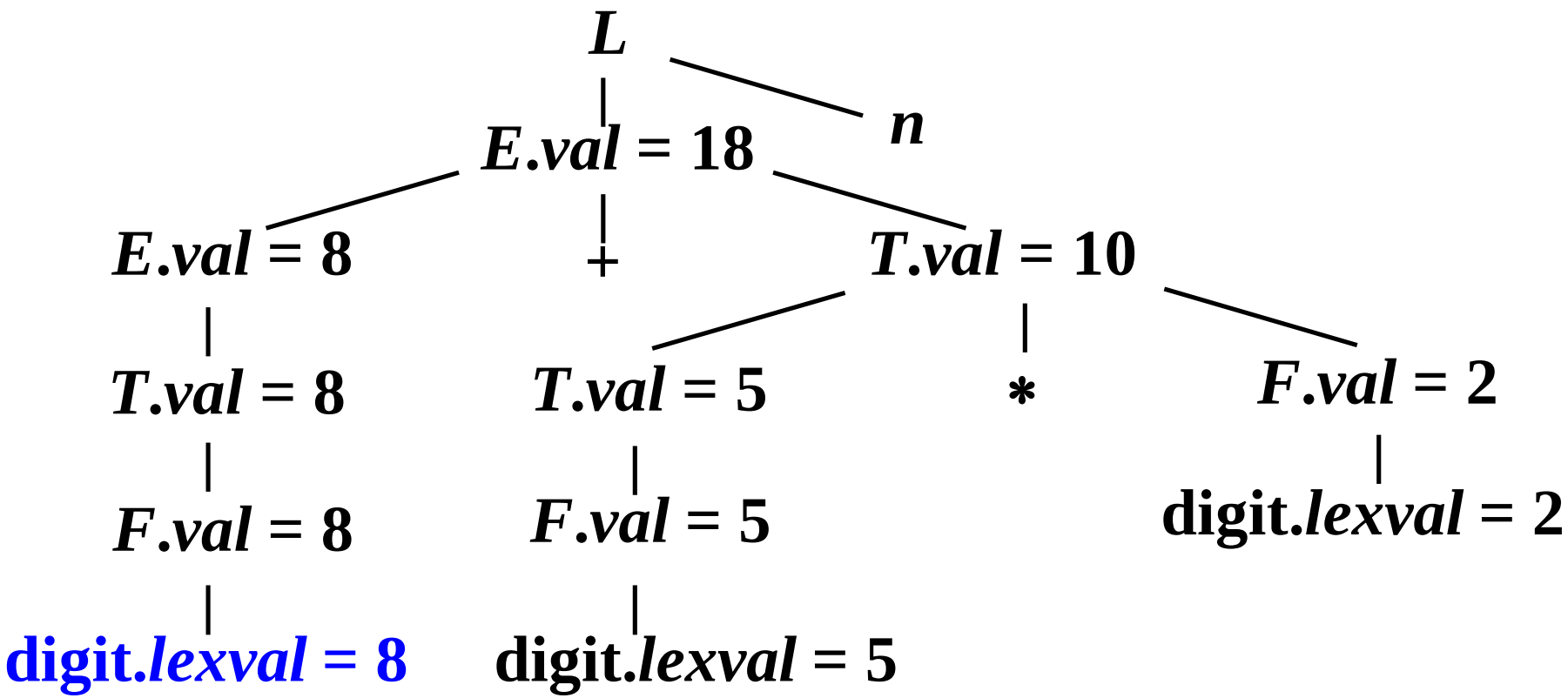
注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成

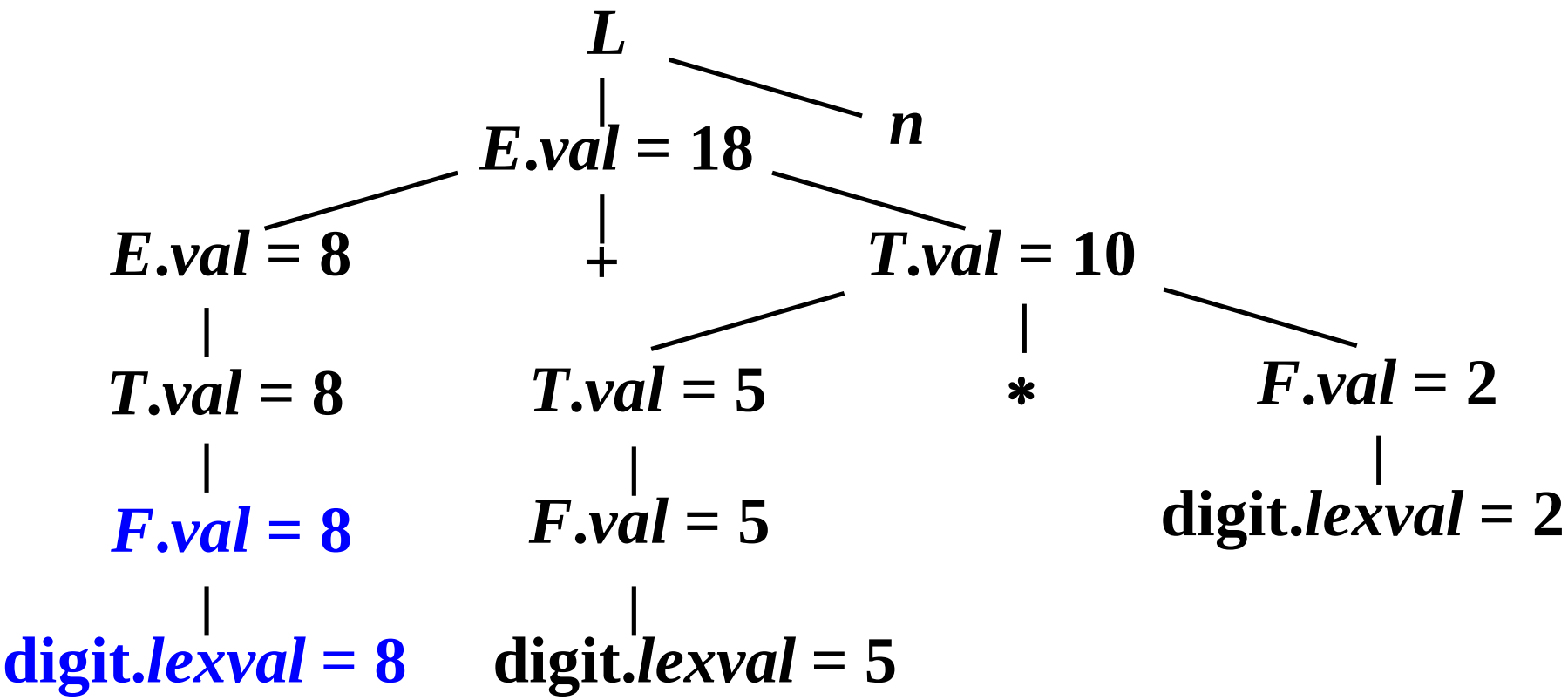


注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成

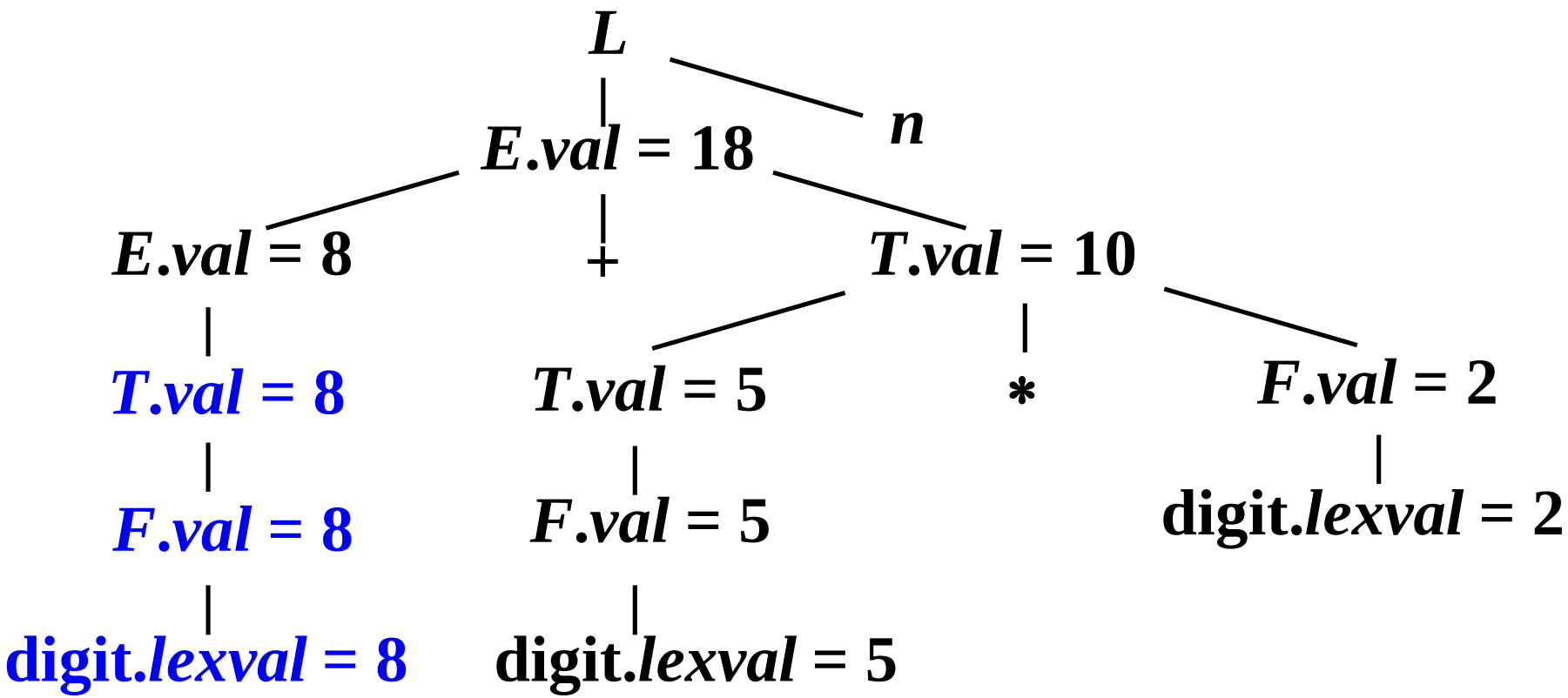


- 各结点综合属性的计算可以自底向上地完成



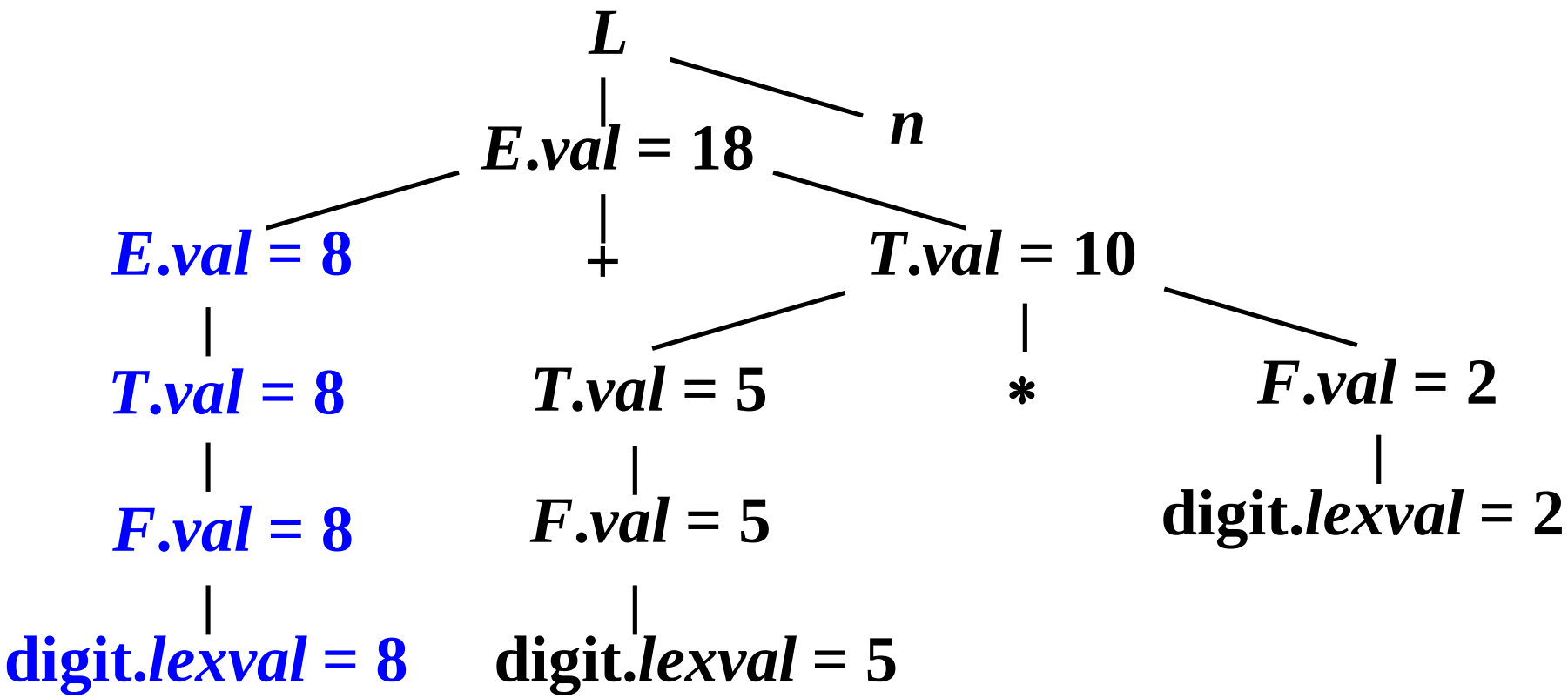
注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成



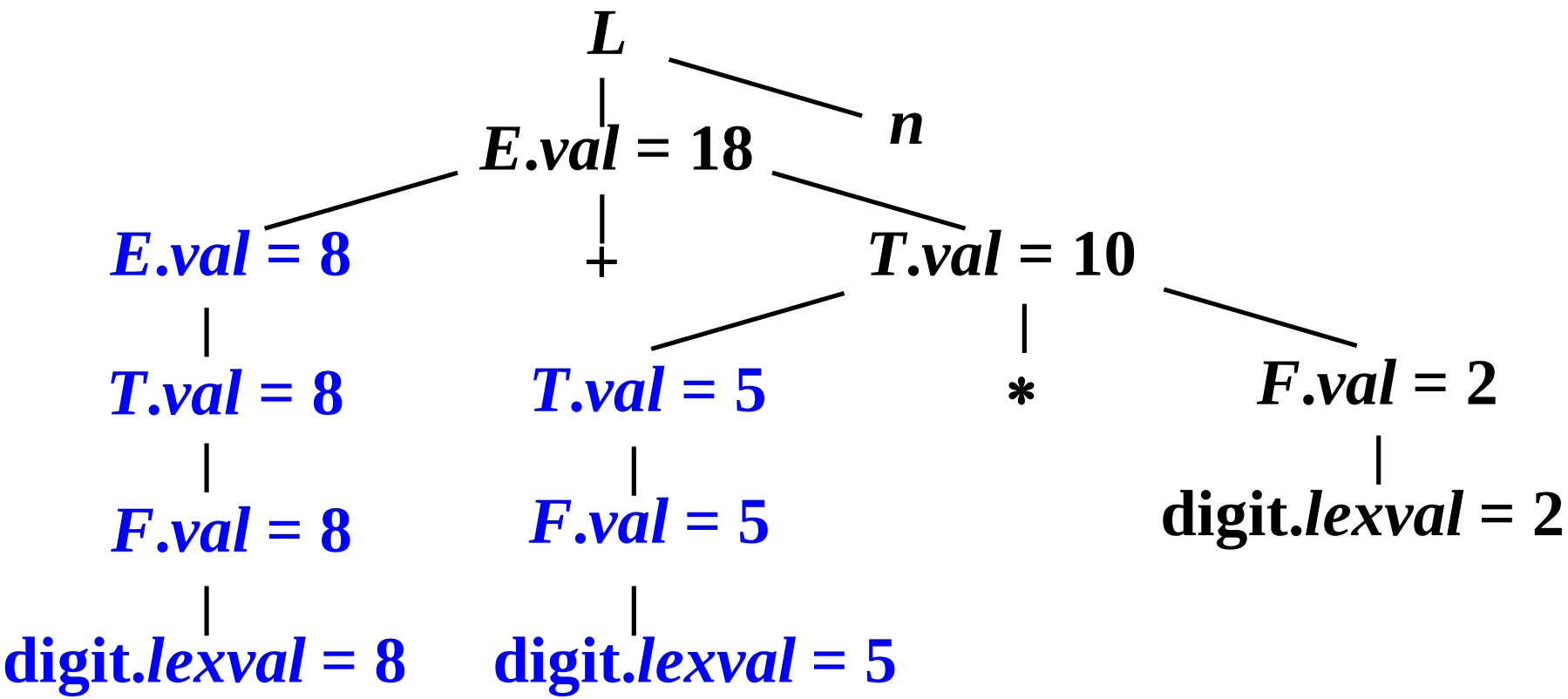
注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成



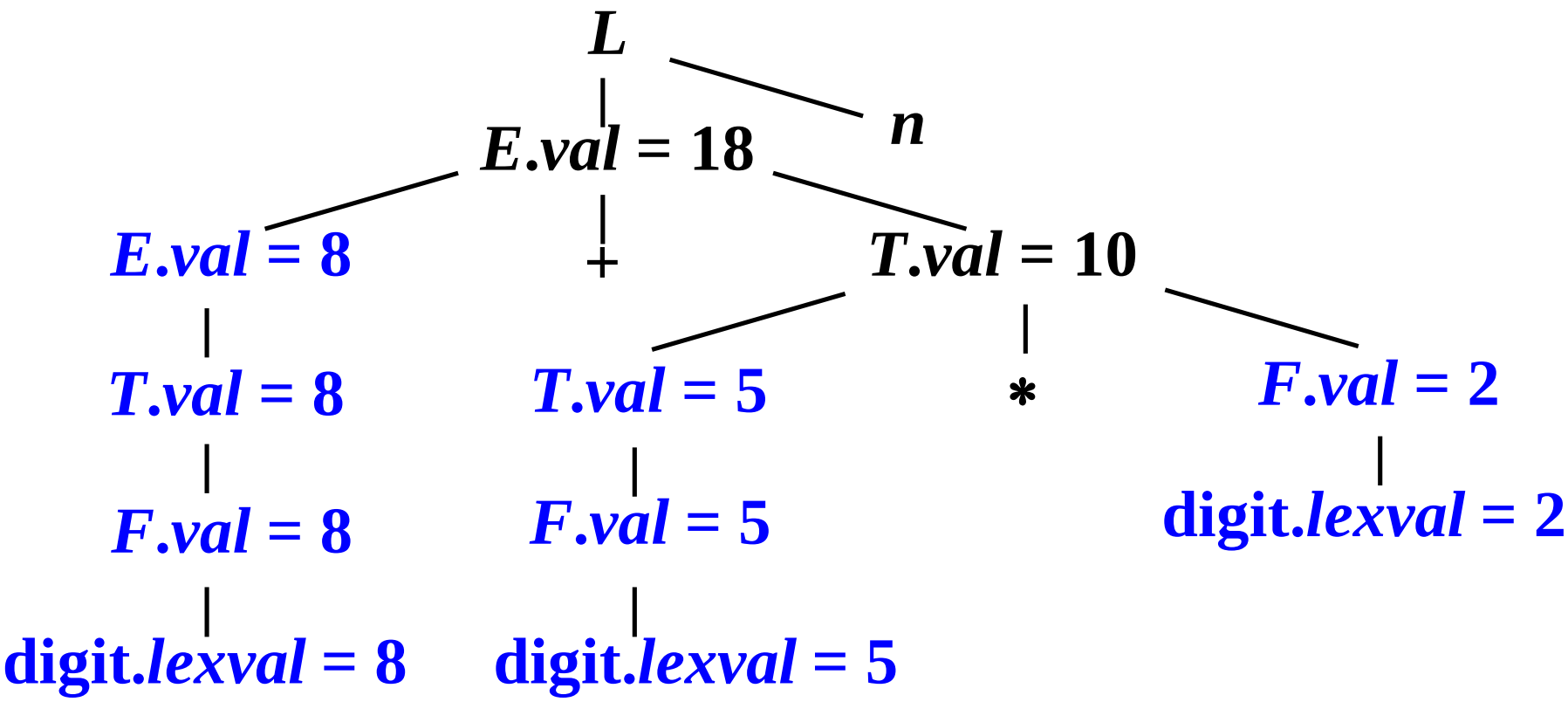
注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成

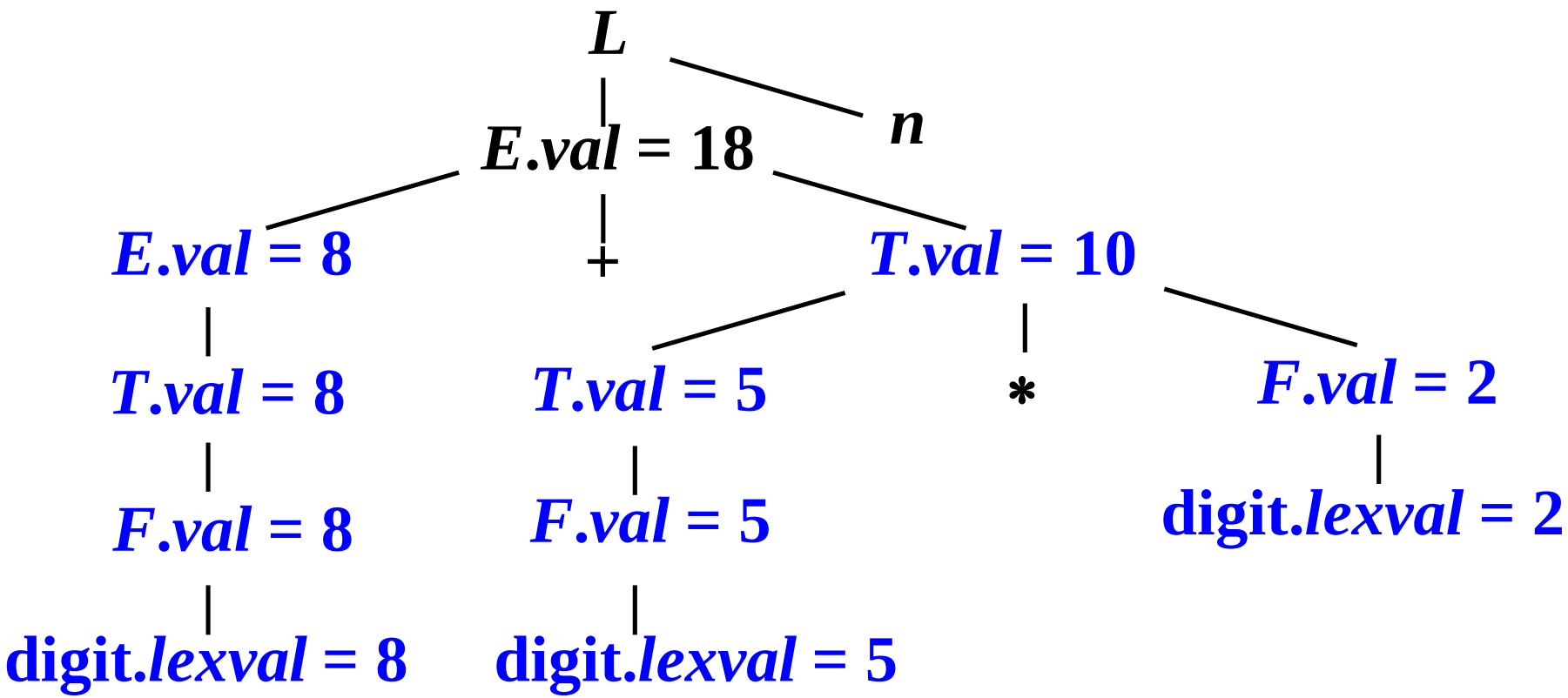


注释分析树+综合属性计算

- 各结点综合属性的计算可以自底向上地完成

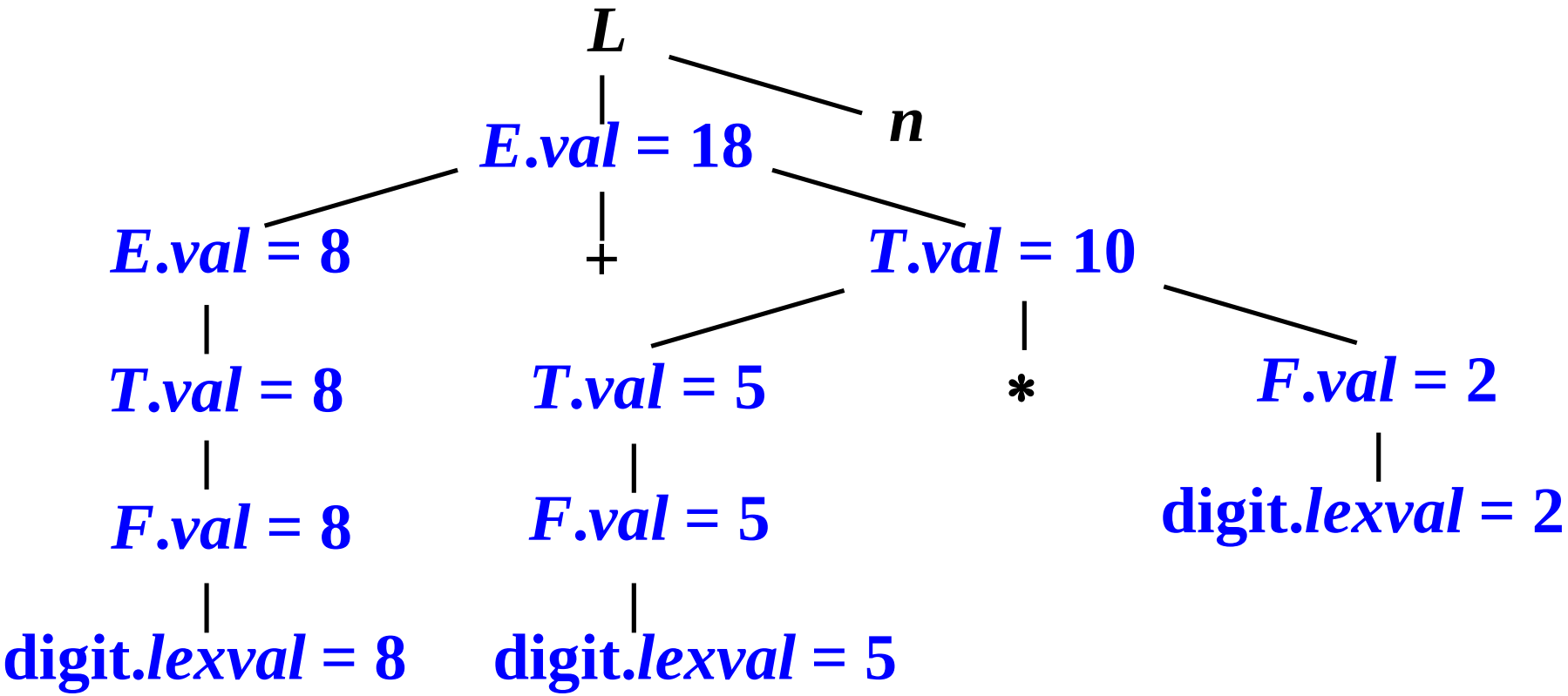


- 各结点综合属性的计算可以自底向上地完成



- 各结点综合属性的计算可以自底向上地完成

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：

$$E \rightarrow E - T \mid T$$

$$T \rightarrow num \mid num.num$$

  - 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：

  $E \rightarrow E - T \mid T$

  $T \rightarrow num \mid num.num$

  - 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 = if (E_1.type == T.type) T.type else float
$E \rightarrow T$	E.type = T.type
$T \rightarrow num$	T.type = integer
$T \rightarrow num.num$	T.type = float

综合属性的局限

- 考虑消除左递归的算术表达式文法

产 生 式	语 义 规 则
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow$ digit	$F.val =$ digit.$lexval$

消除左递归

产 生 式
$T \rightarrow FT'$
$T' \rightarrow *FT_1'$
$T' \rightarrow \varepsilon$
$F \rightarrow$ 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?

综合属性的局限

- 考虑消除左递归的算术表达式文法

产 生 式	语 义 规 则
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow$ digit	$F.val =$ digit.$lexval$

消除左递归

产 生 式
$T \rightarrow FT'$
$T' \rightarrow *FT_1'$
$T' \rightarrow \varepsilon$
$F \rightarrow$ 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

语法制导的定义

❑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

  $b=f(c_1, c_2, \ldots, c_k)$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

借助继承属性

- 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

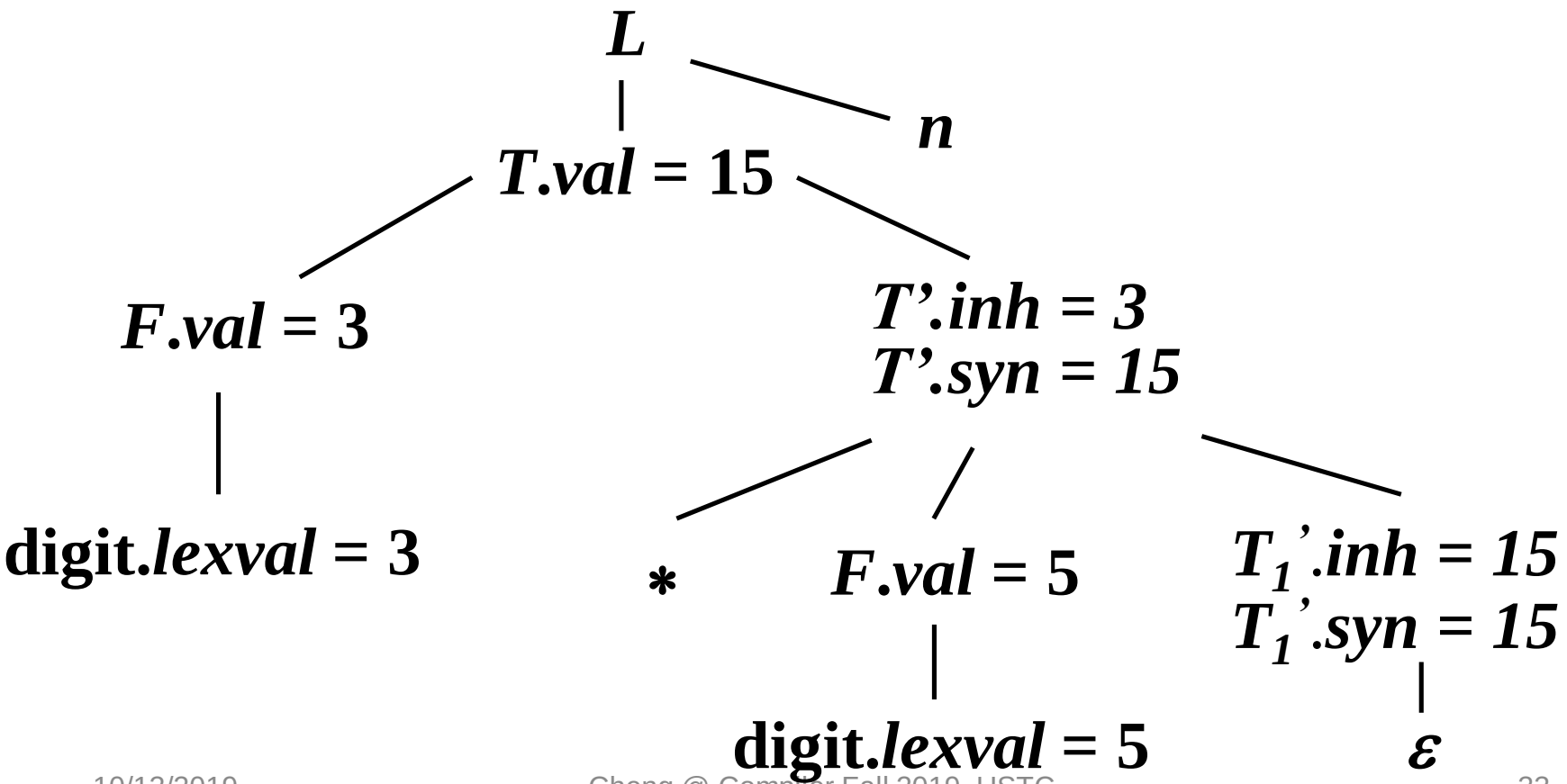
产 生 式	语 义 规 则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

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

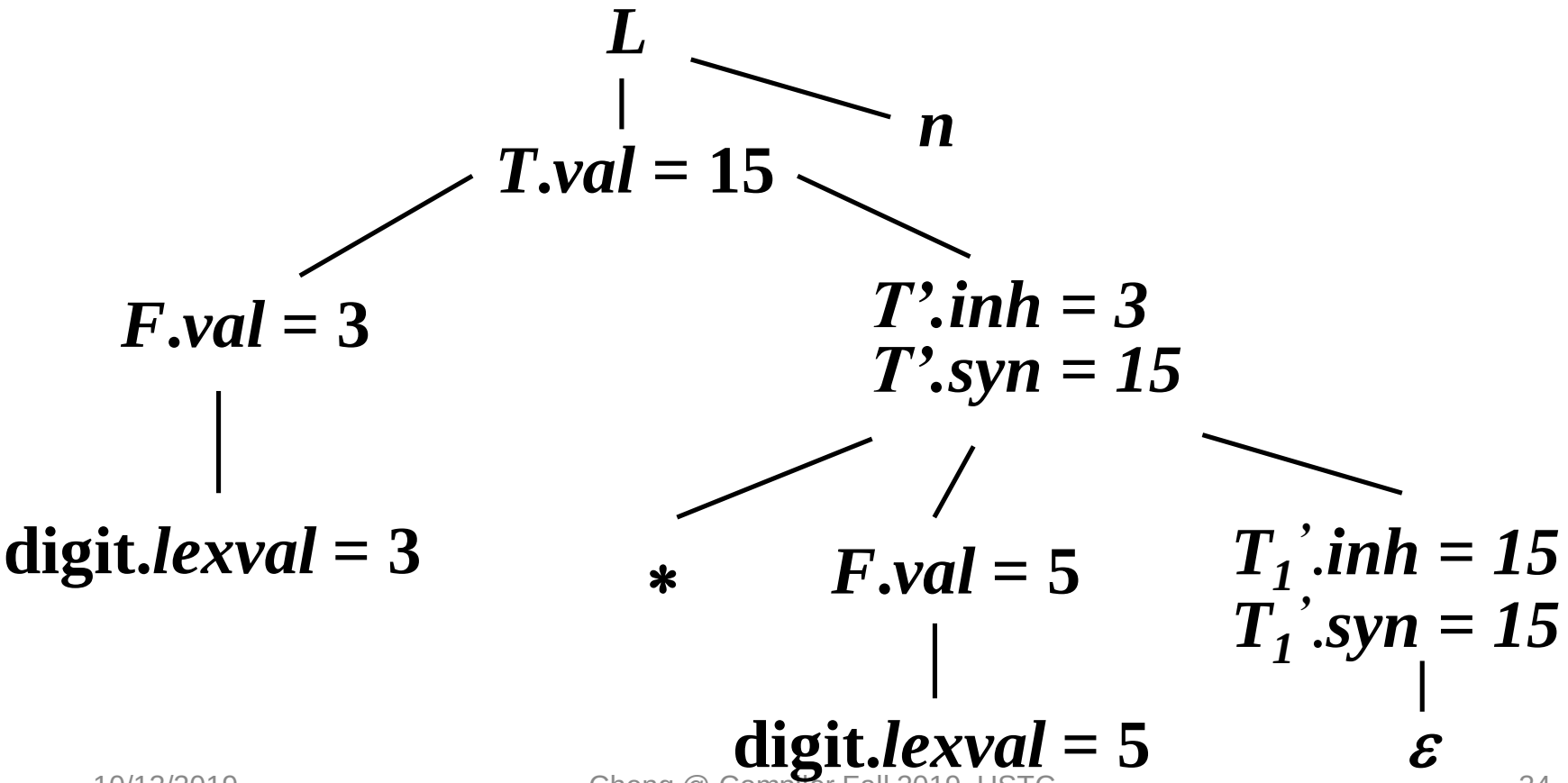
注释分析树+继承属性计算

❑3*5的注释分析树



注释分析树+继承属性计算

- 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

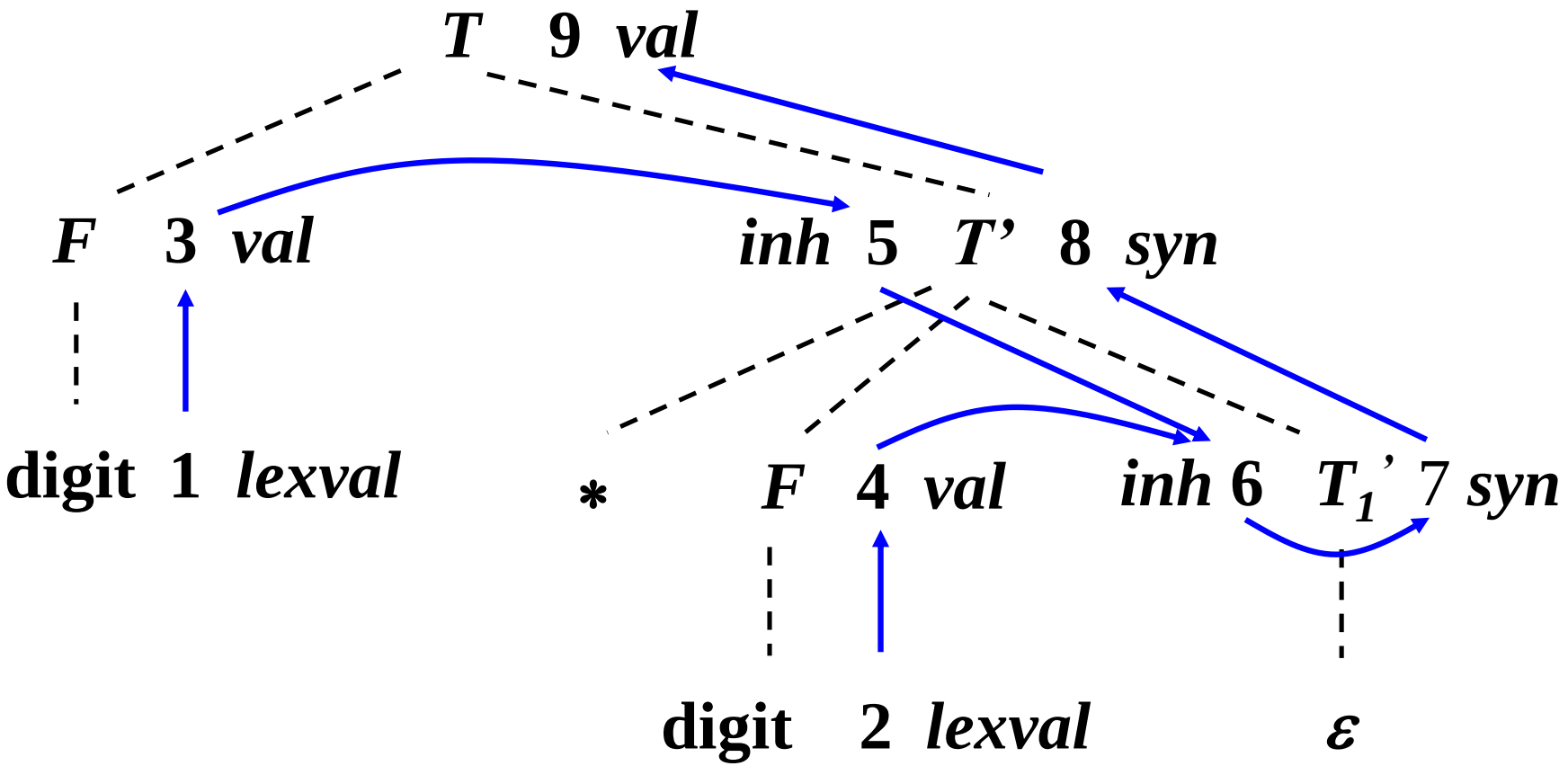
- 按照什么顺序计算属性值?
  - 语义规则建立了属性之间的依赖关系，在对语法分析树节点的一个属性求值之前，必须首先求出这个属性值所依赖的所有属性值

属性依赖图

- 依赖图(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

属性依赖图

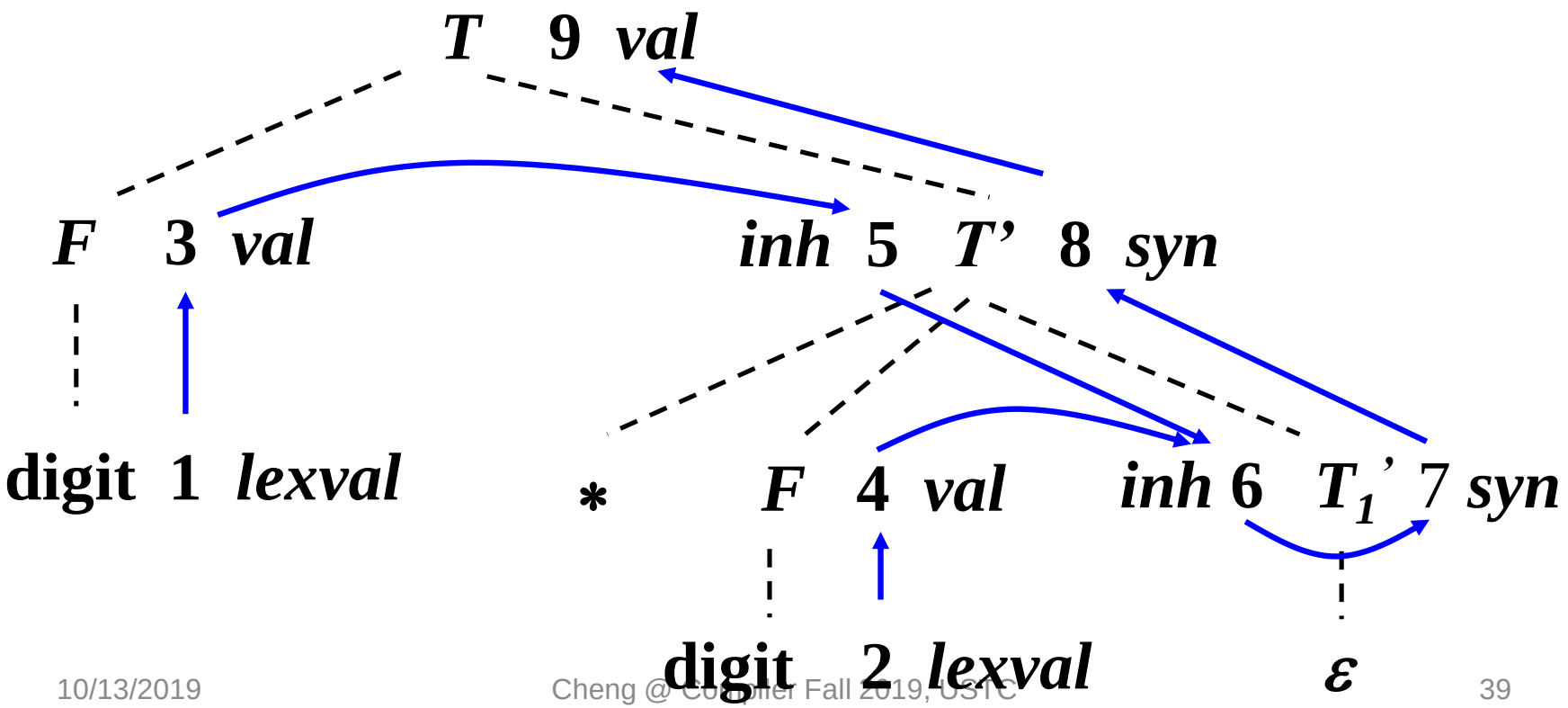
❑以消除左递归的算术表达式文法和3*5为例



- 可行的求值顺序是满足下列条件的结点序列 $N_1, N_2, \ldots, N_k$：
  - 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，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)**

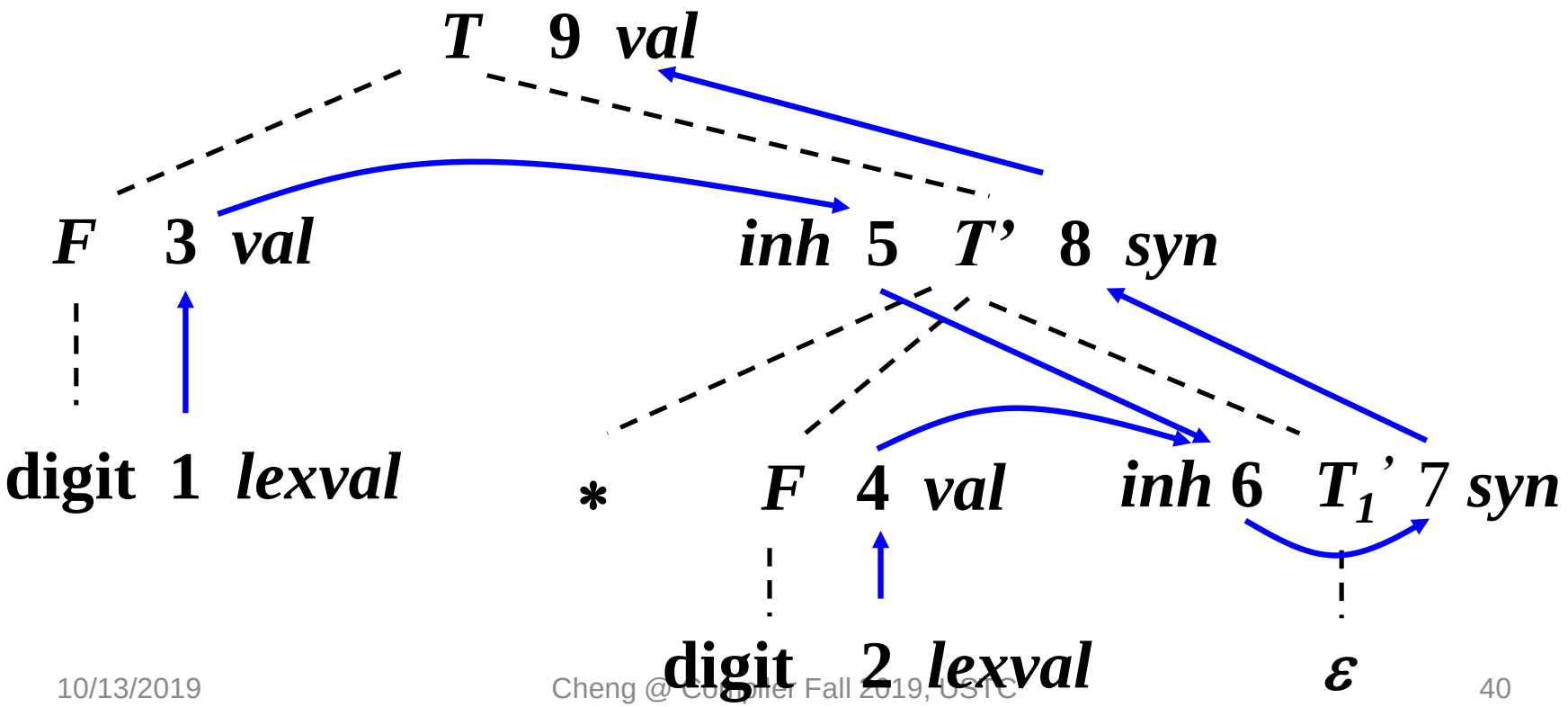
属性计算次序：举例

- 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



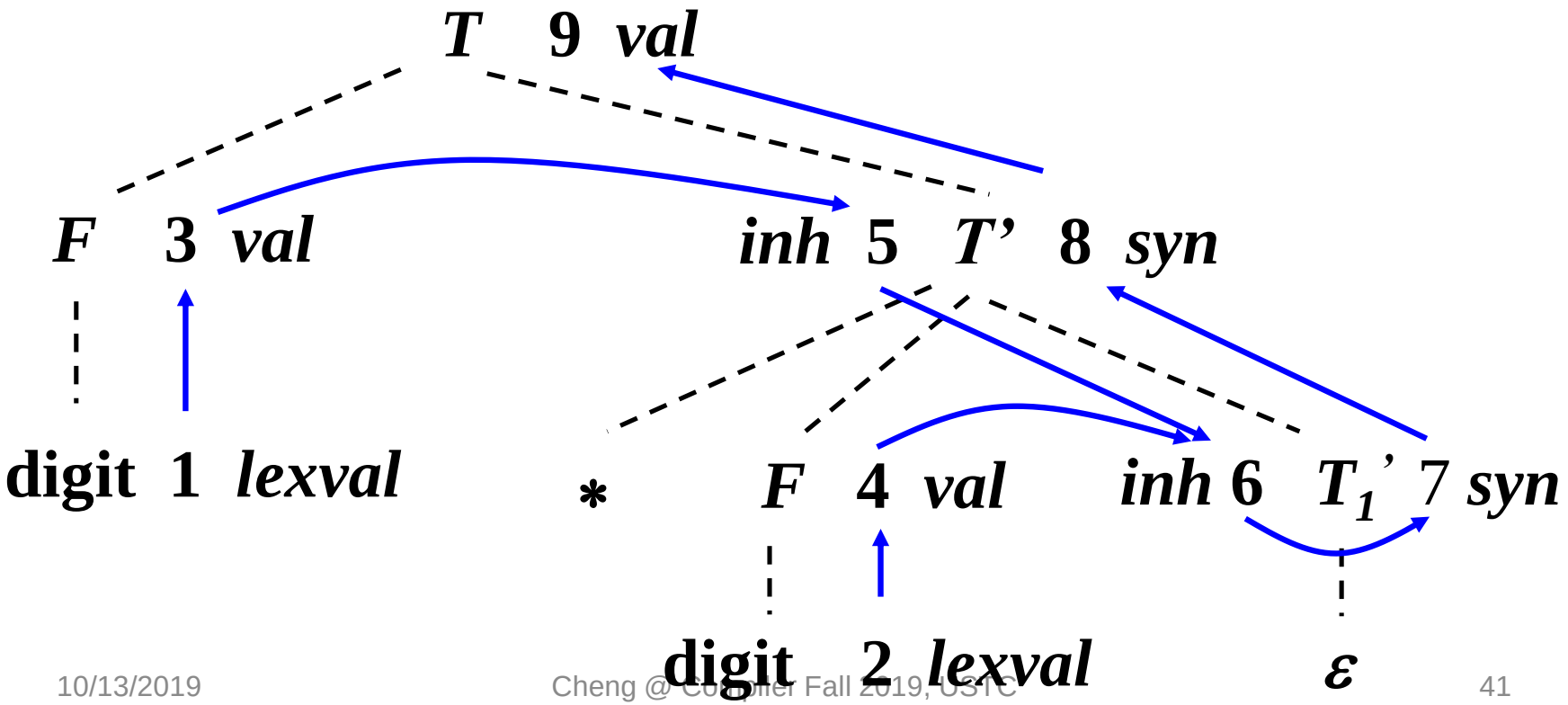
属性计算次序：举例

- 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?

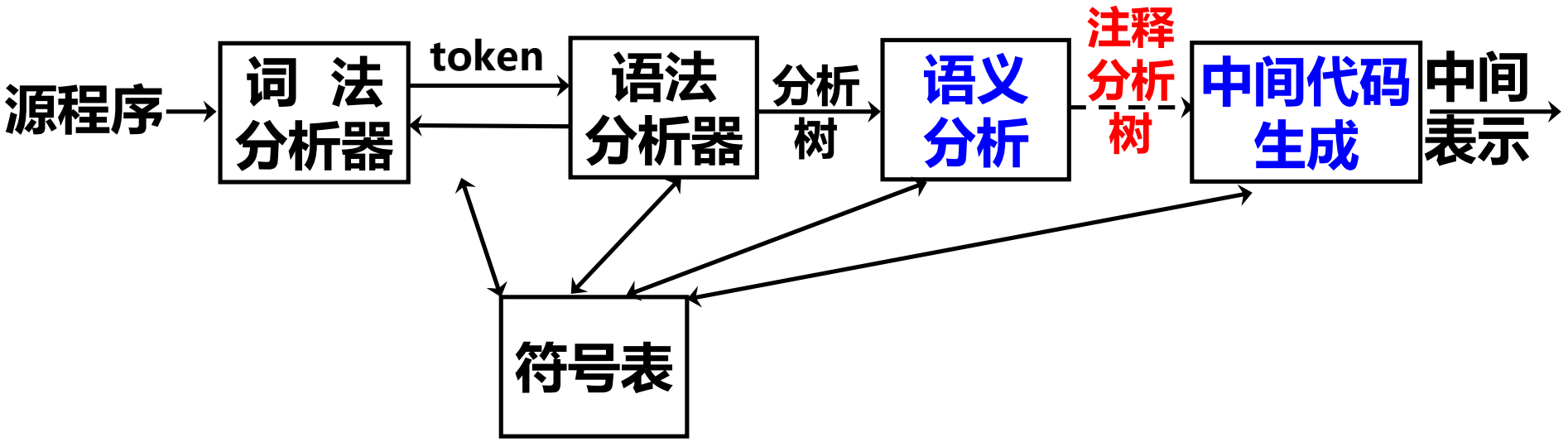
属性计算的问题

- 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
    - S属性的SDD和L属性的SDD

主要内容



- ❑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

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
    - 例如：LR分析器

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
$L \rightarrow E$ 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*

例2

- 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2

- 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3

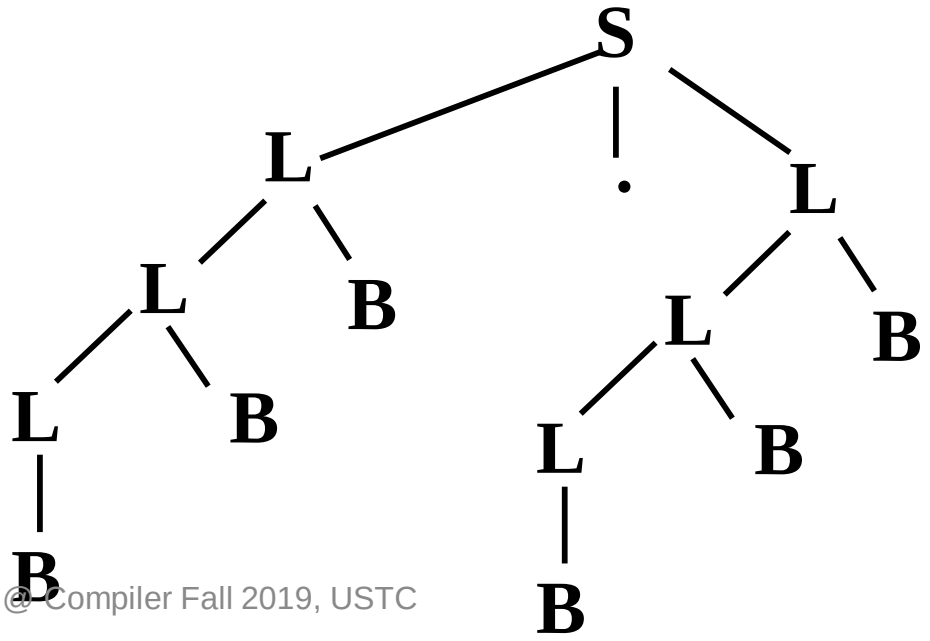
❑为下面文法写一个语法制导的定义，用S的综合属性*val*给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625（可以修改文法）

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3

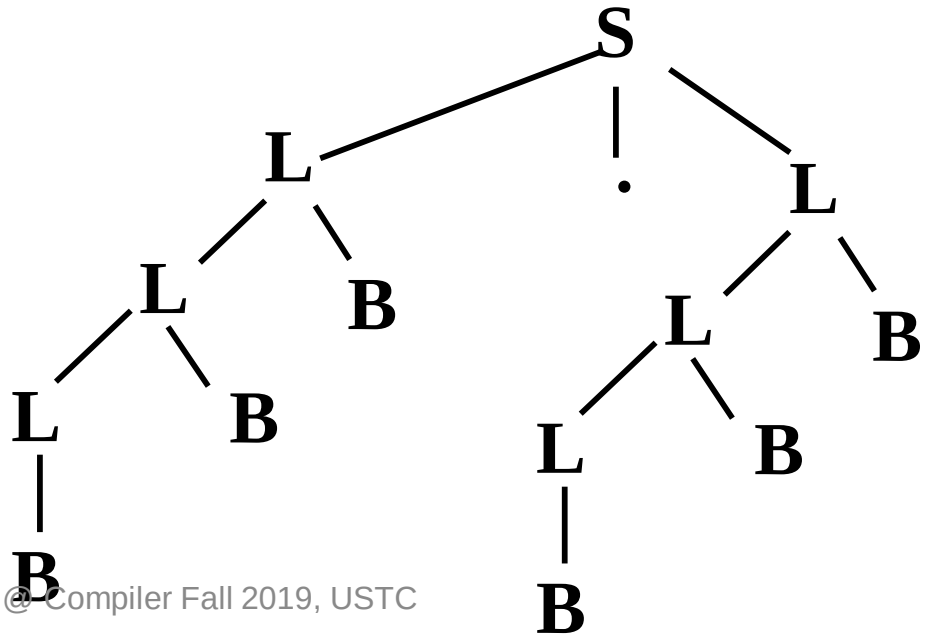
❑为下面文法写一个语法制导的定义，用S的综合属性*val*给出下面文法中S产生的二进制数的值。例如，输入101.101时，S.*val* = 5.625（可以修改文法）

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3

❑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）

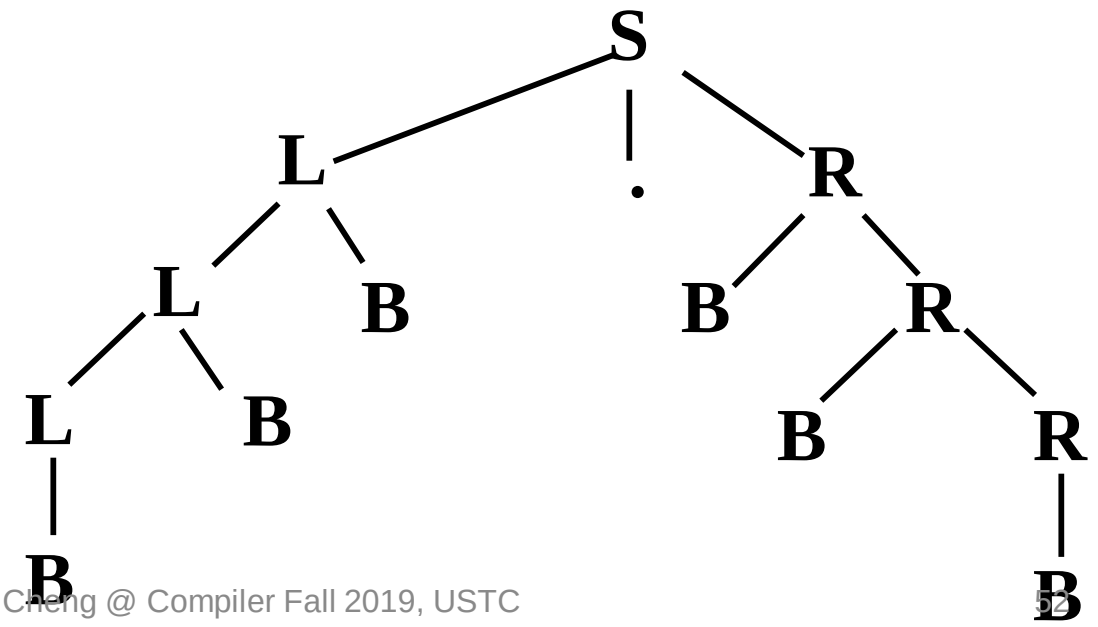
更清楚的办法是将文法改成下面的形式

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

$L \rightarrow L\ B \mid B$

$R \rightarrow B\ R \mid B$

$B \rightarrow 0 \mid 1$



例3

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

Homework 12/10/2019

- 借助继承属性完成例3

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)
  - 可以在LR分析器或者LL分析器中实现

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$X_1, X_2, \ldots, X_{i-1}$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

L属性的定义

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

属性文法vs.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

属性文法vs.副作用

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

- 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

L属性的定义：举例

int id, id, id
标识符声明

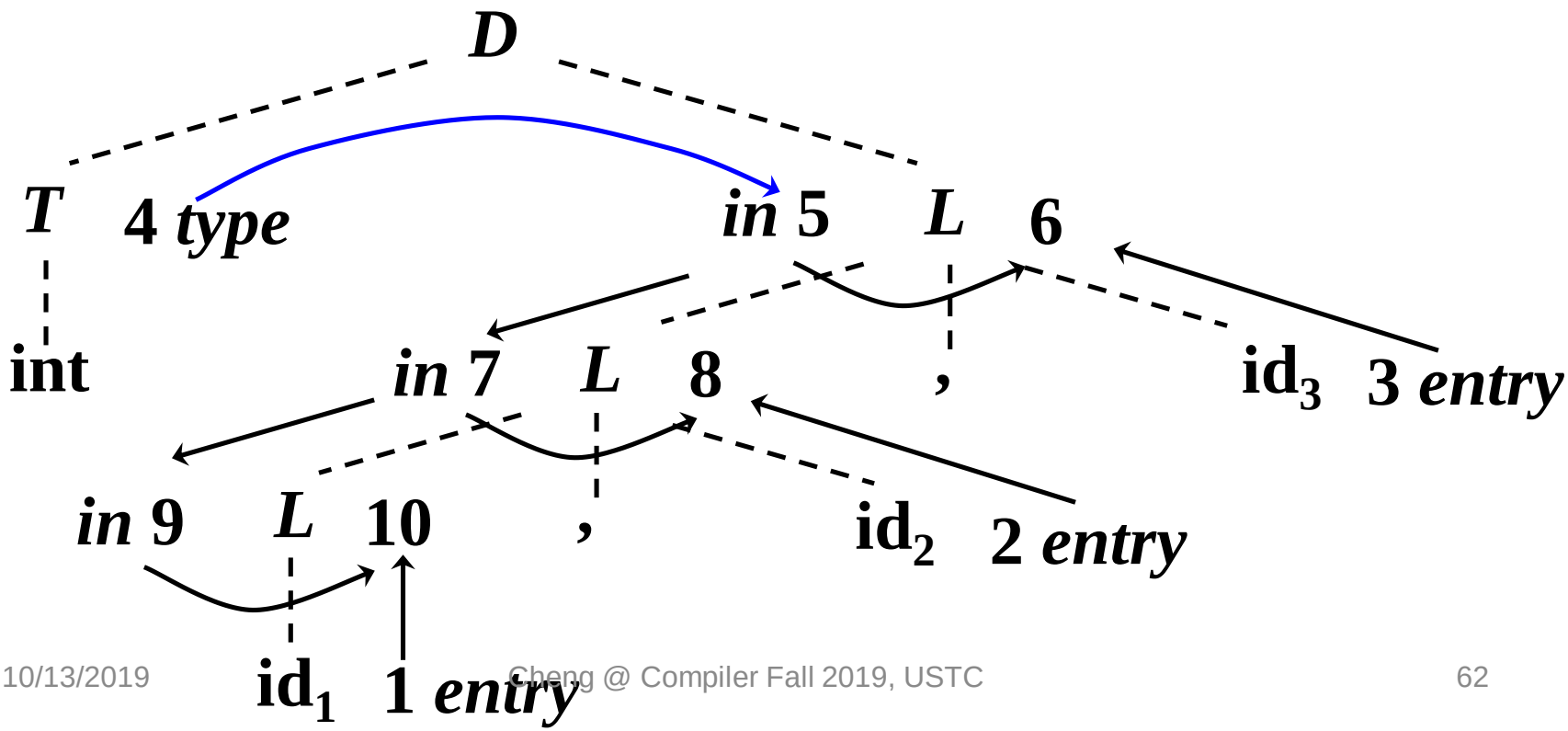
产　生　式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addType(\text{id}.entry, L.in)$

- *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)

L属性的定义：举例

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

$D \rightarrow TL$　　$L.in = T.type$

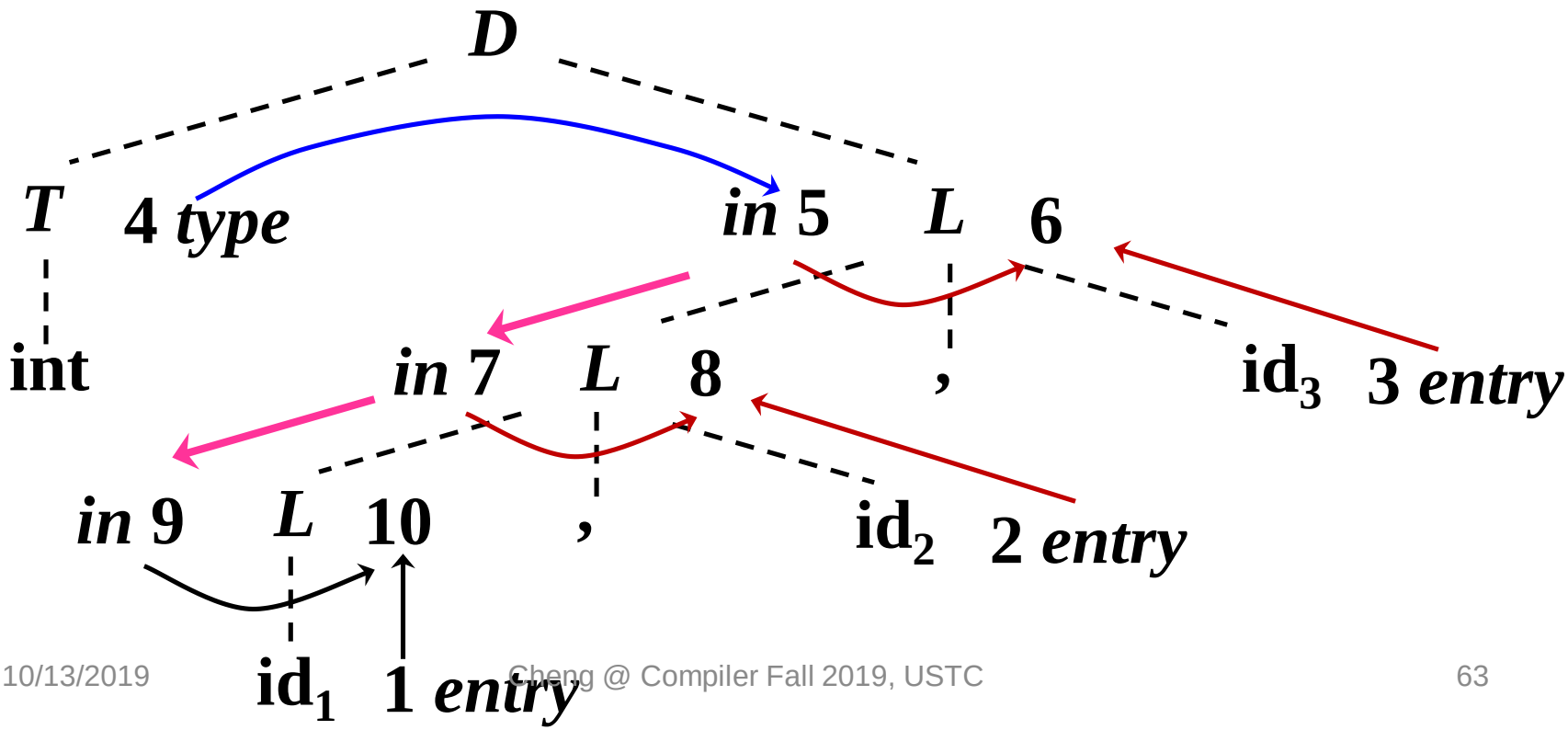


L属性的定义：举例

❑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*)

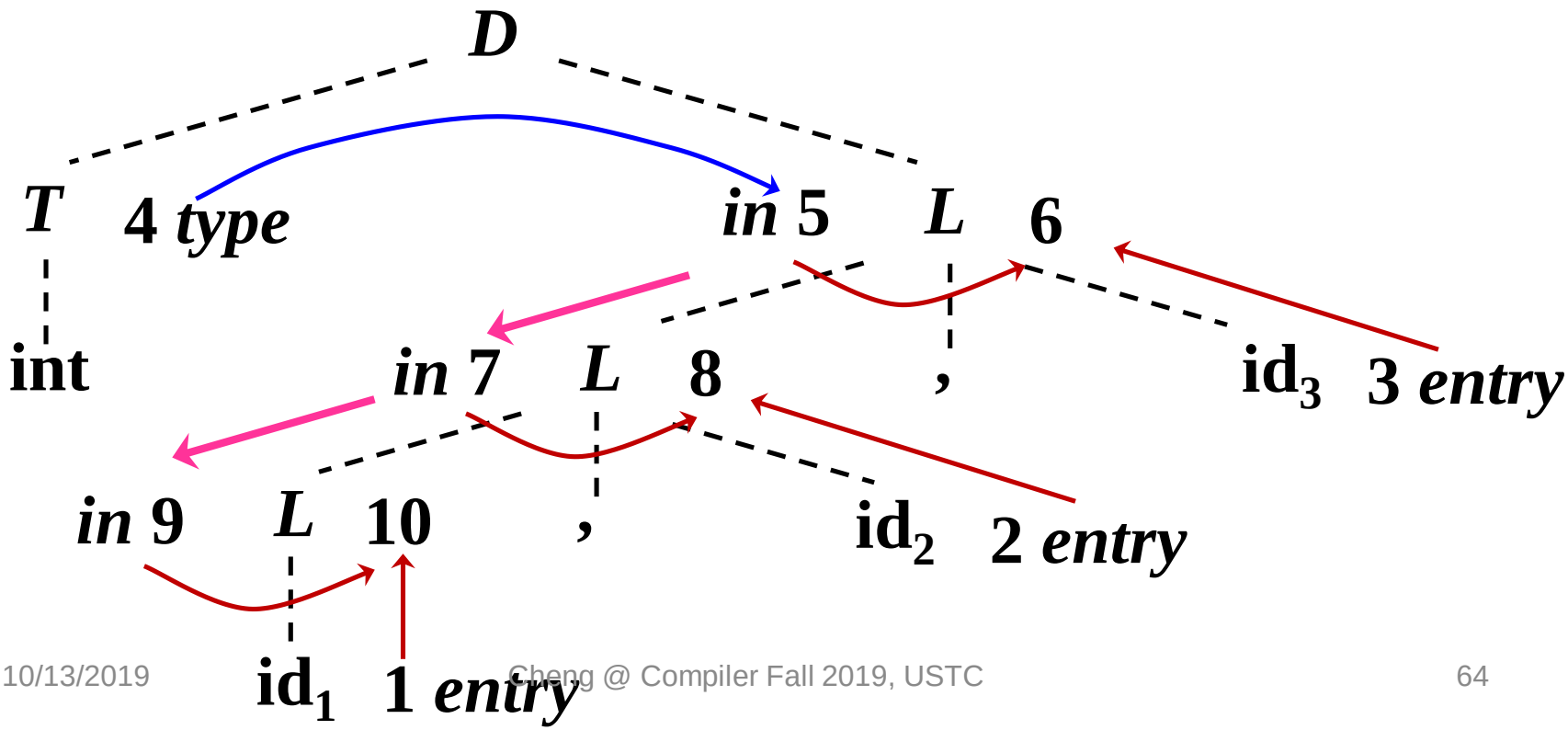


L属性的定义：举例

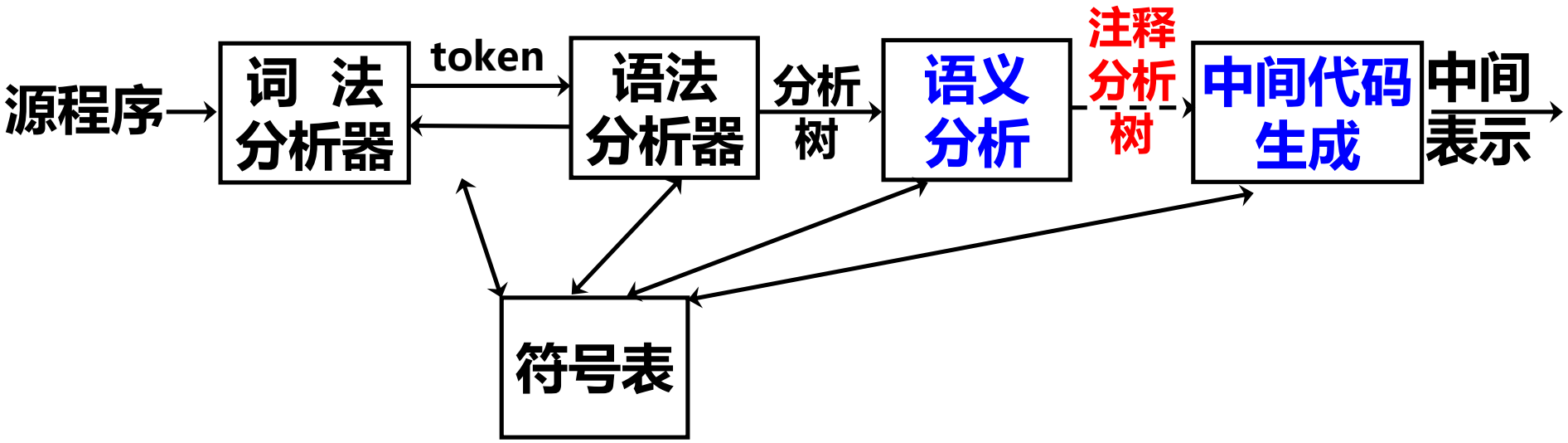
❑例 int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

$L \rightarrow$ id

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



主要内容



- 语法制导翻译简介
- 语法制导定义
  - 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

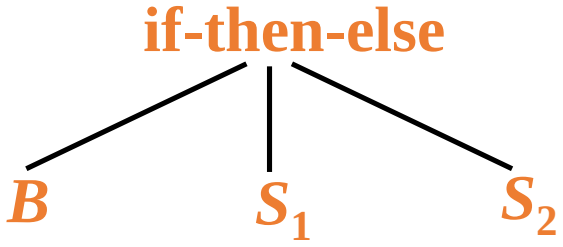
语法制导定义的应用

- 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
- 中间代码生成

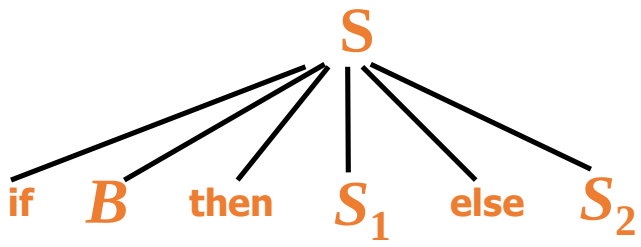
❑抽象语法树(Abstract syntax tree, AST)

- ❖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→ if B then S1 else S2

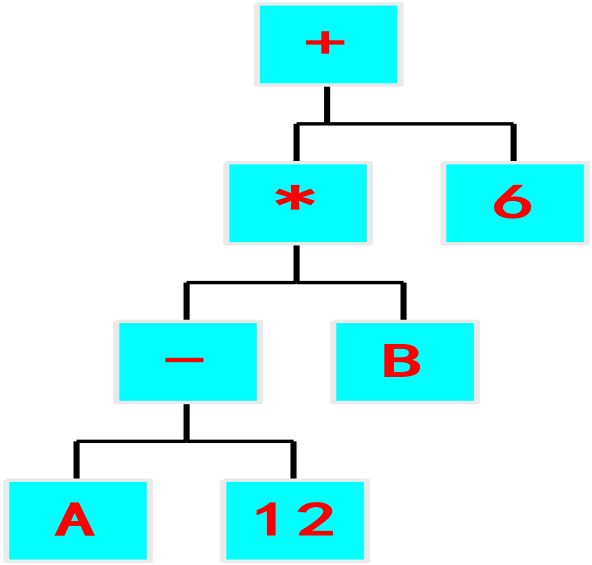


语法树



分析树

❑例：表达式 (A - 12) * B + 6 的语法树。



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。

构造语法树的语法制导定义

❑以算数表达式为例

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

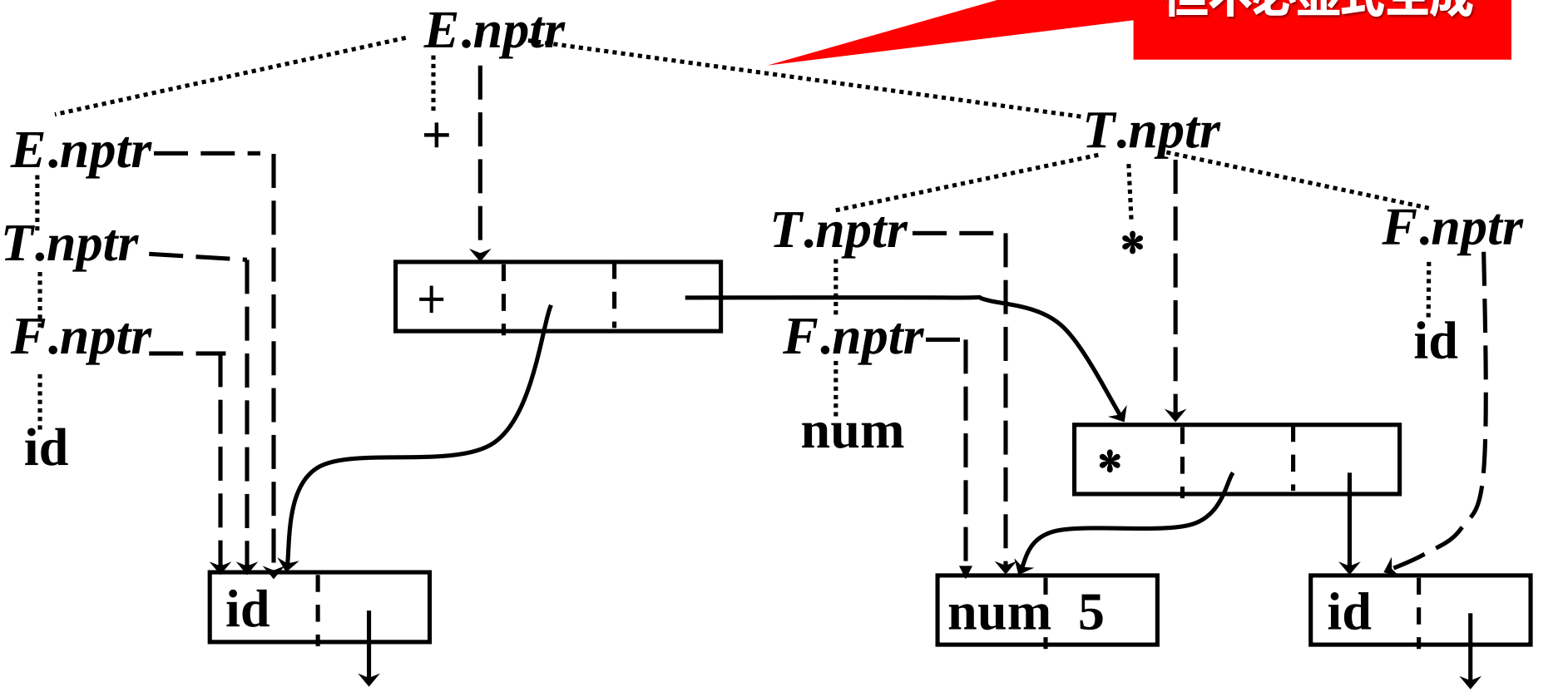
❑注意事项：

- ❖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）

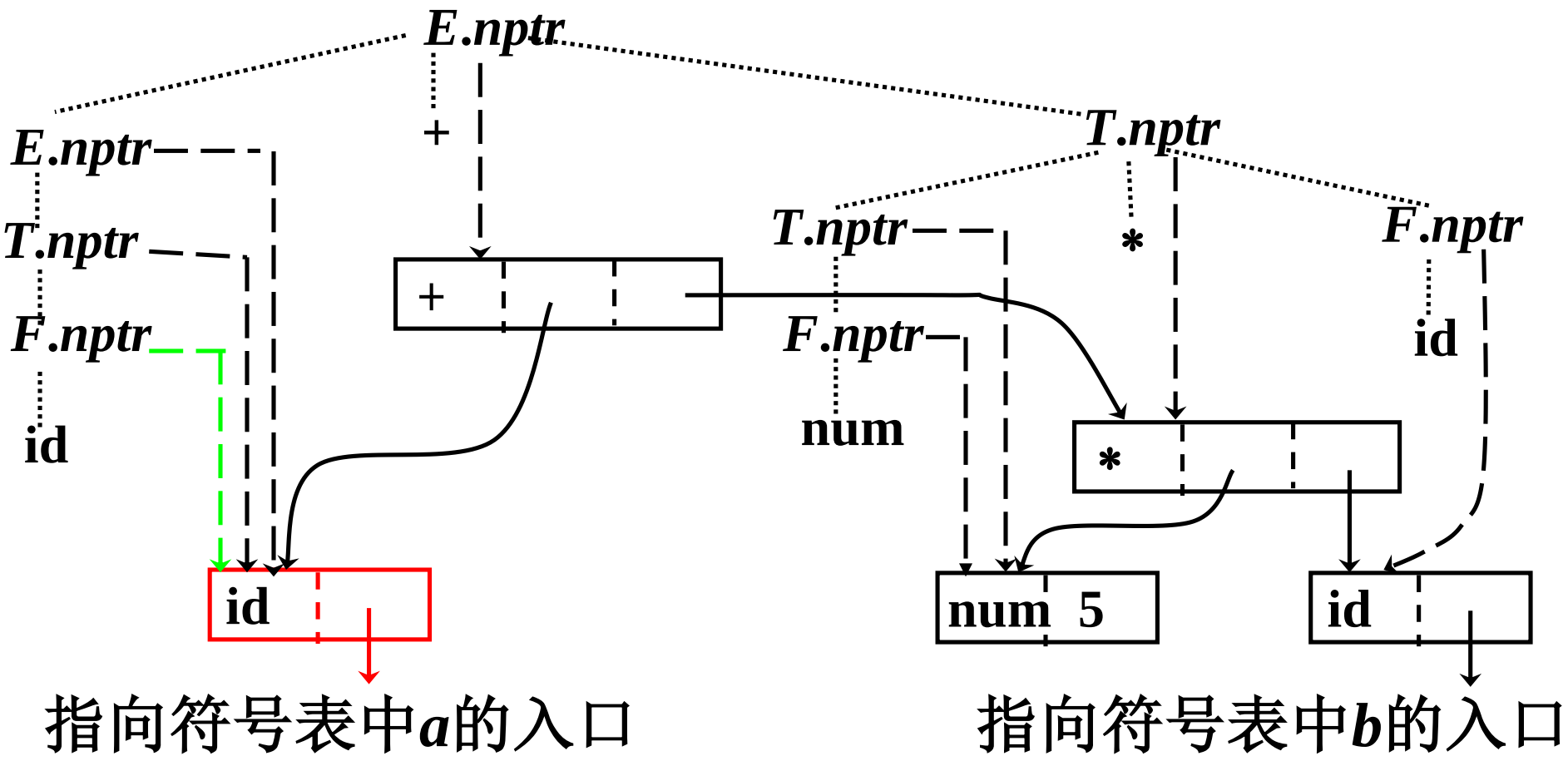
S属性定义的语法树构造

a+5**b*的语法树的构造

不带箭头的虚线代表语法分析树，但不必显式生成

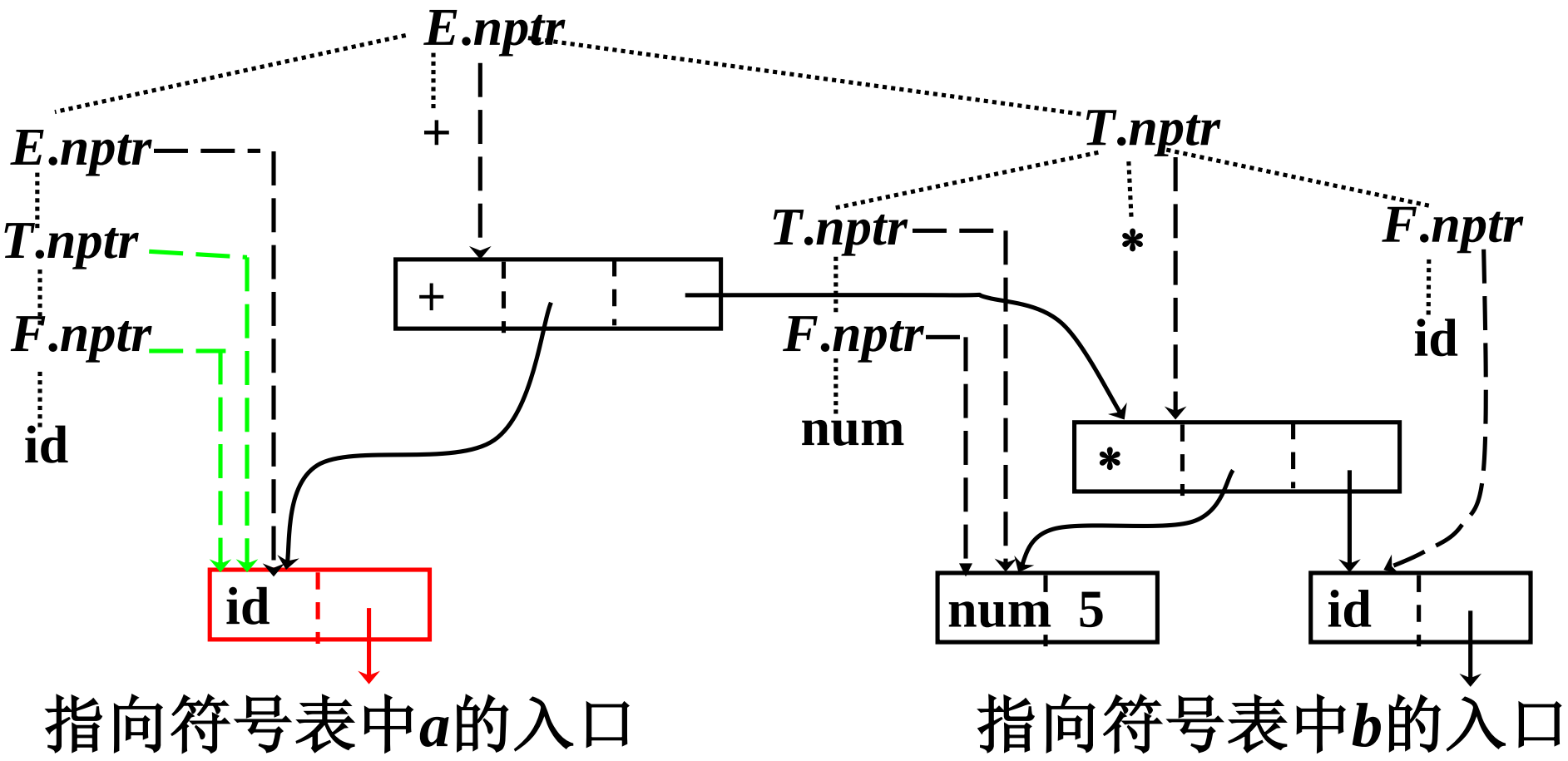


a+5**b*的语法树的构造

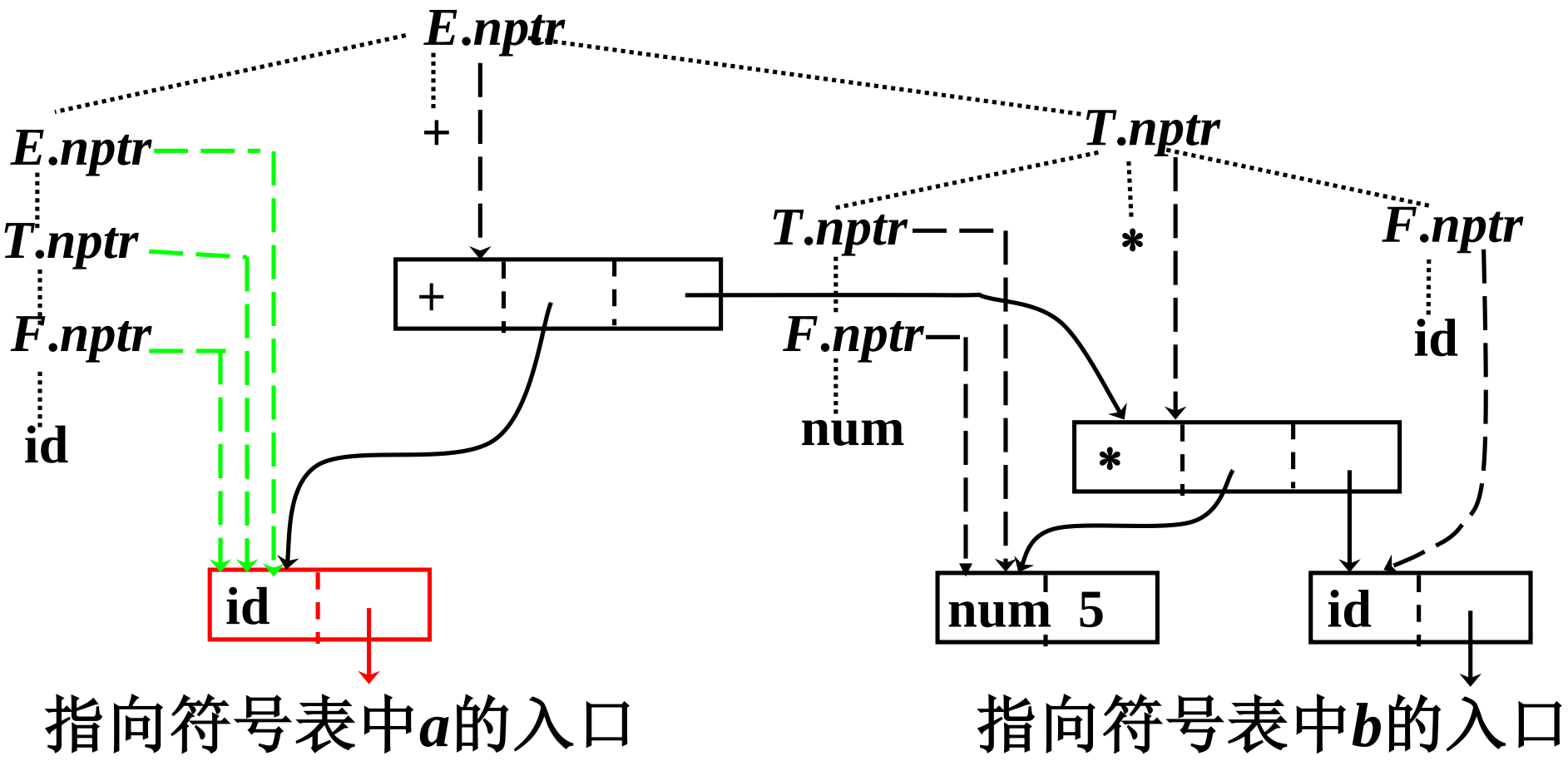

E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
id
num
*
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

S属性定义的语法树构造

a+5**b*的语法树的构造

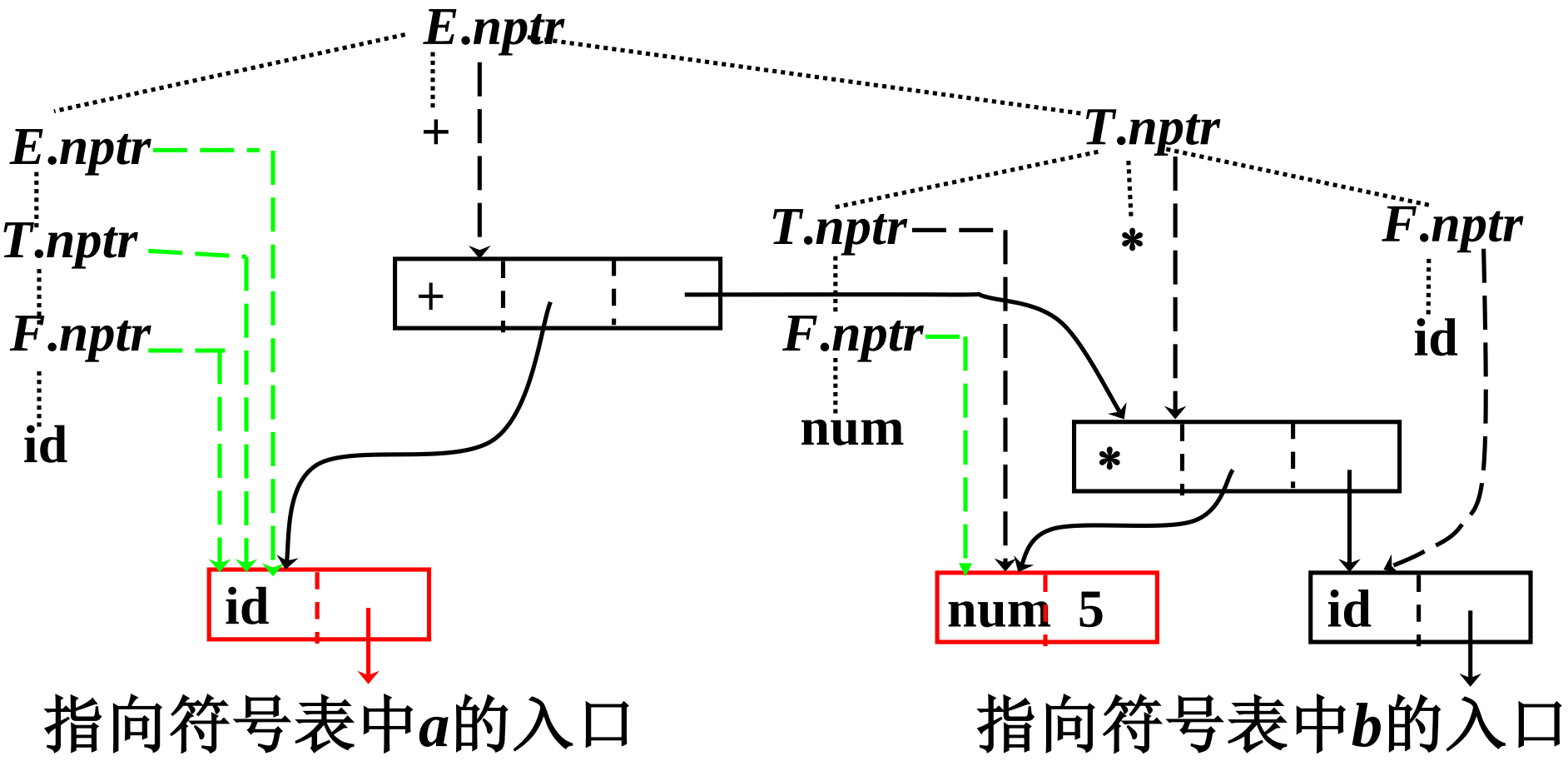


a+5**b*的语法树的构造

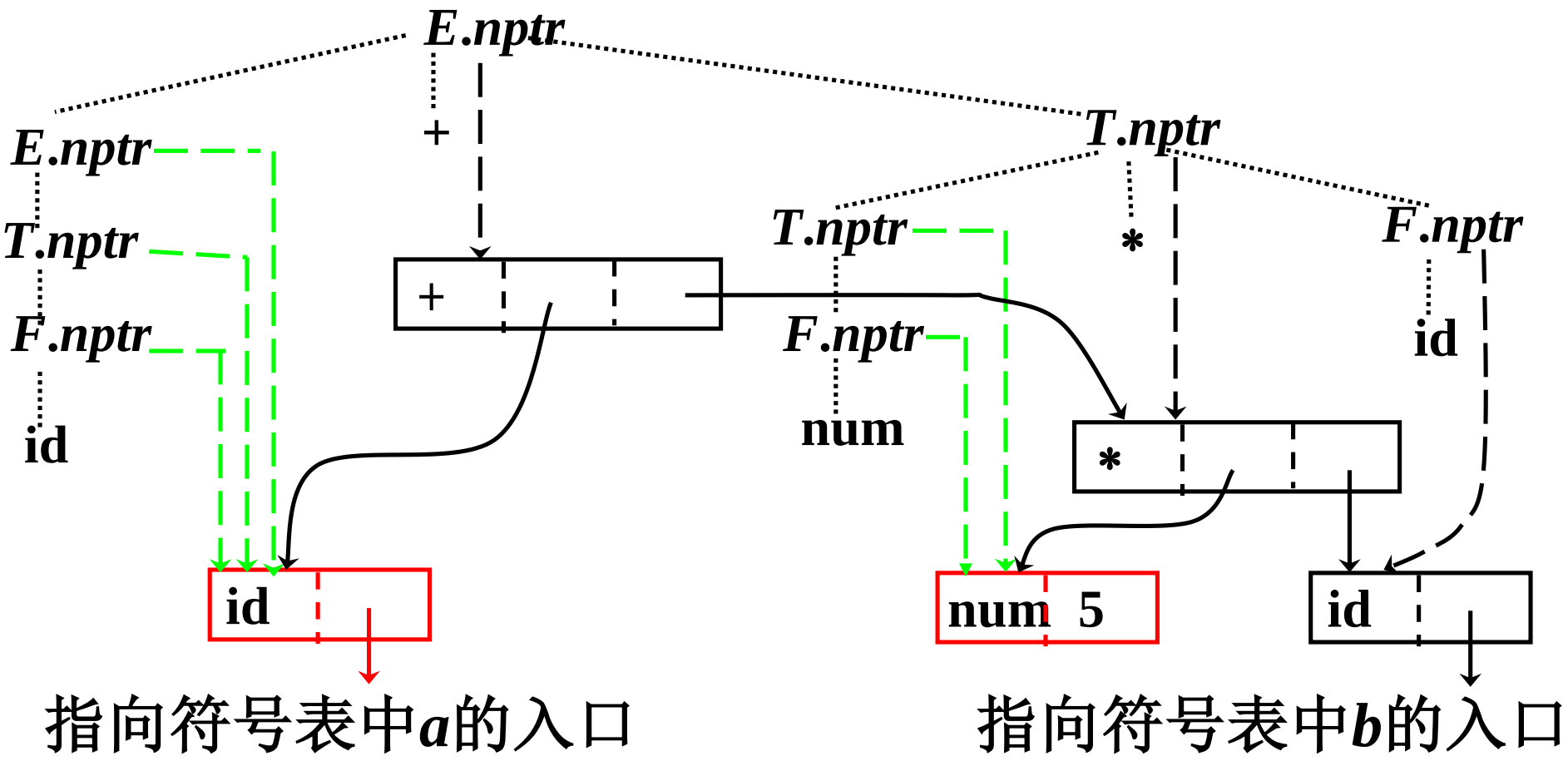

E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
num
*
id
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

S属性定义的语法树构造

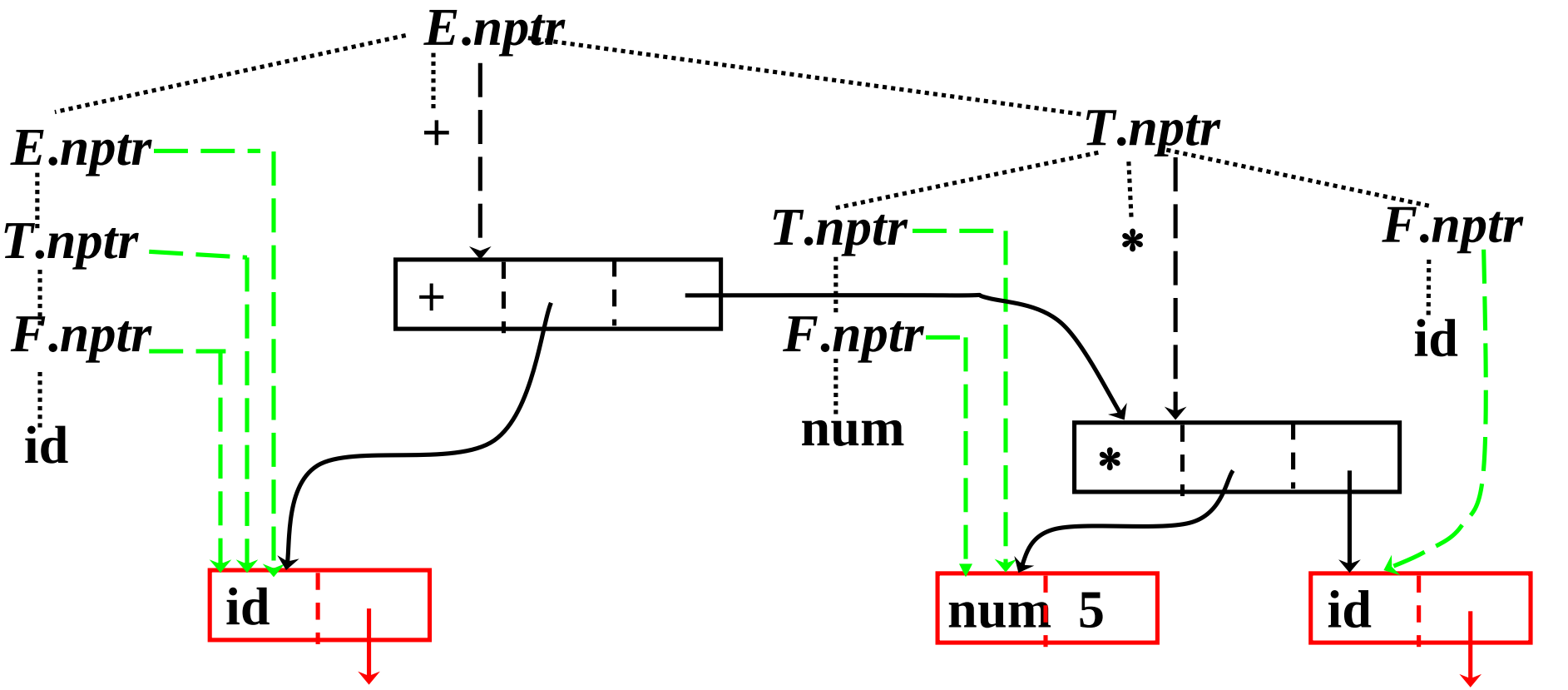
a+5**b*的语法树的构造



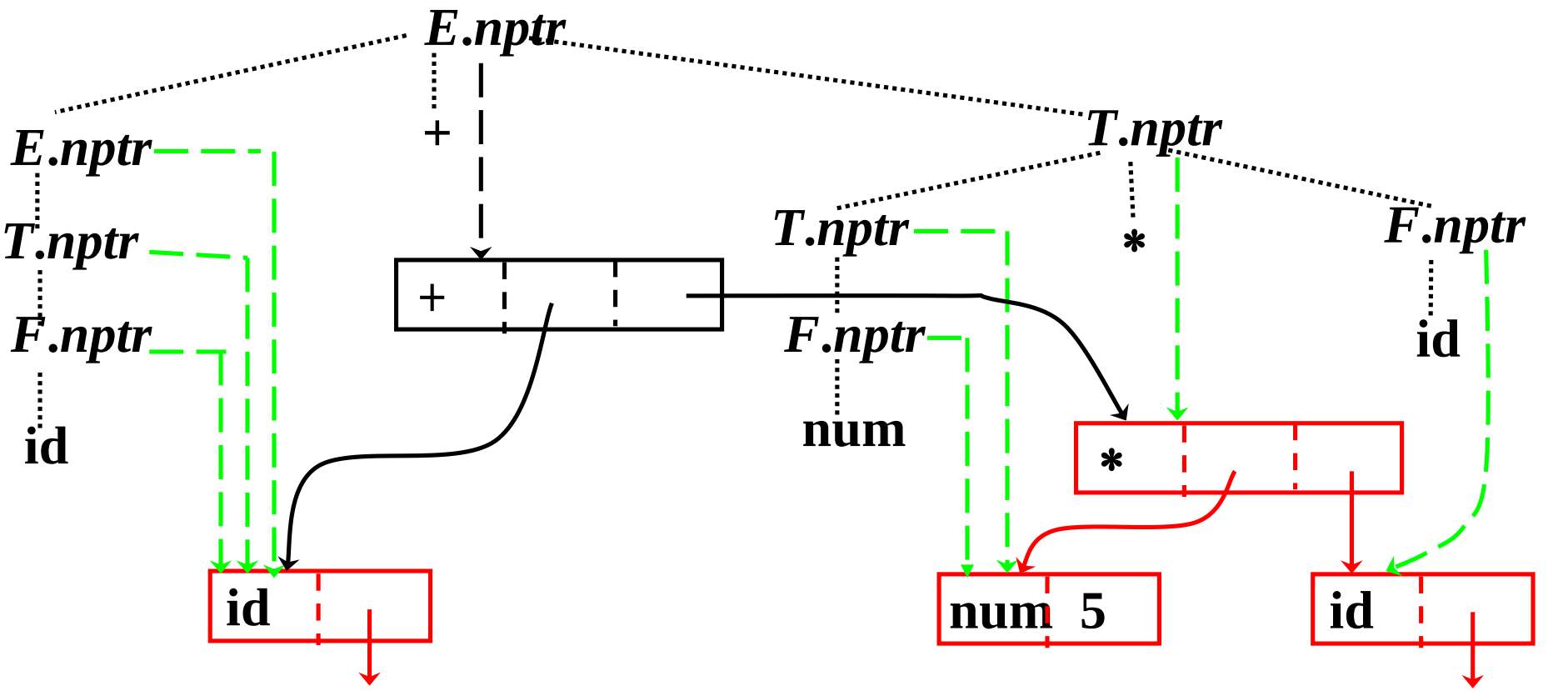
a+5**b*的语法树的构造


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

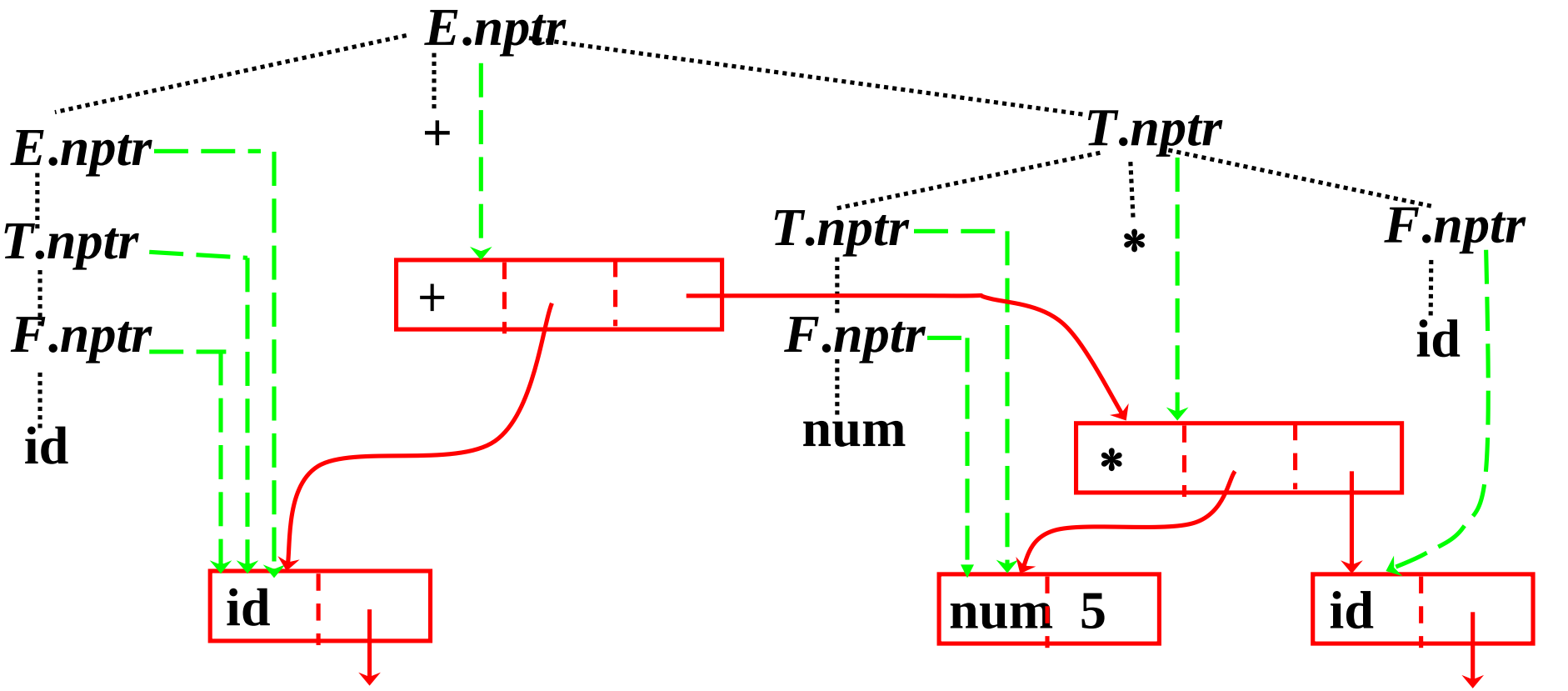

E.nptr
E.nptr
+
T.nptr
T.nptr
T.nptr
*
F.nptr
F.nptr
+
F.nptr
id
num
*
id
id
num 5
id
指向符号表中a的入口
指向符号表中b的入口

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
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



构造 a-4+c语法树的步骤

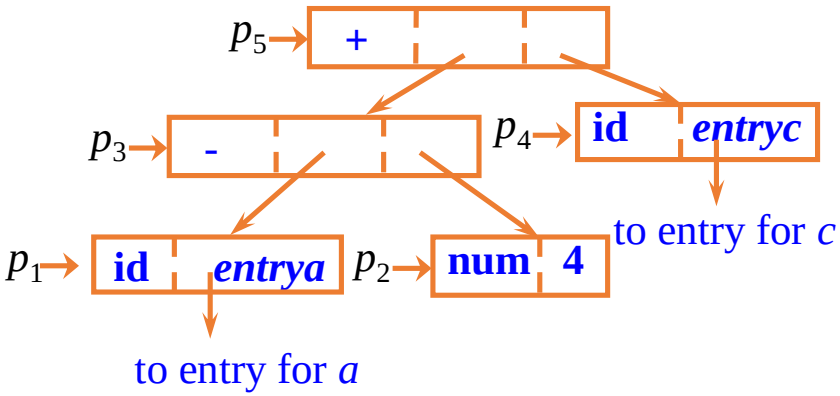
(1) p1:=mkleaf(id,entry a);

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

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

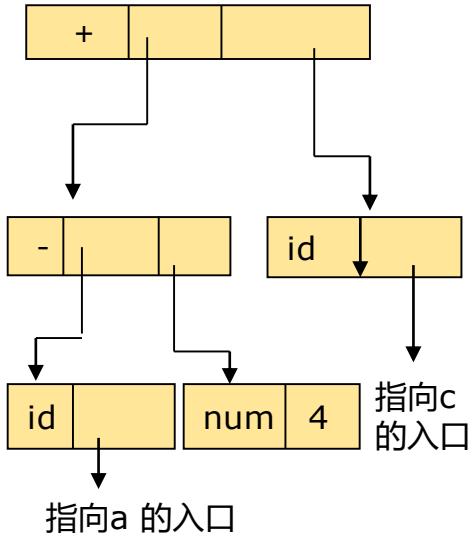
(4) p4:=mkleaf(id, entry c)

(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。



左递归的消除引起继承属性

❑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

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

$T + T + T + \ldots$

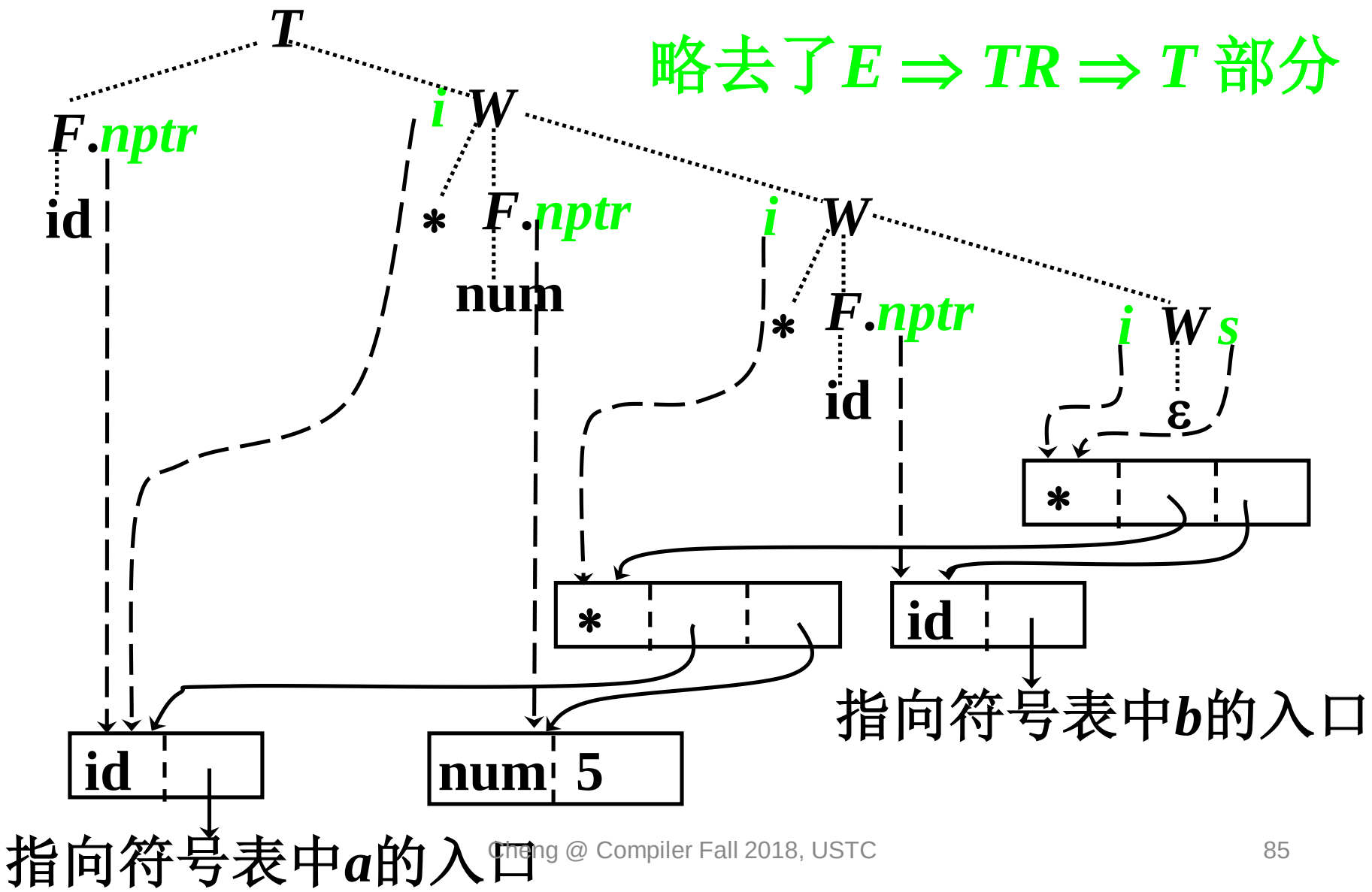
$E \rightarrow TR$
$R \rightarrow +TR_1$
$R \rightarrow \varepsilon$
$T \rightarrow FW$
$W \rightarrow *FW_1$
$W \rightarrow \varepsilon$

F产生式部分不再给出

L属性定义的语法树构造

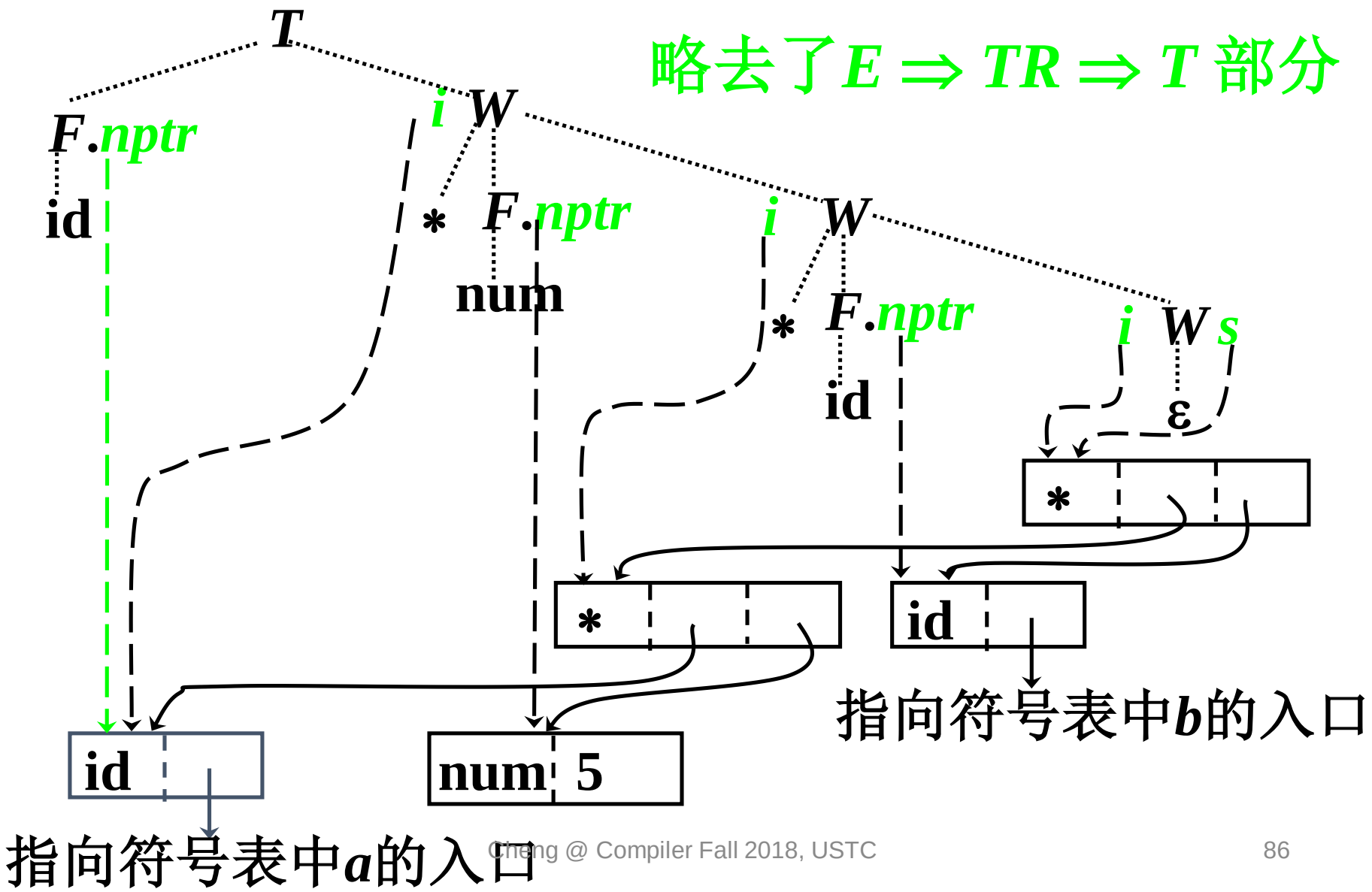
产 生 式	语 义 规 则
$T \rightarrow FW$	$W.i = F.nptr$ $T.nptr = W.s$
$W \rightarrow *FW_1$	$W_1.i = mkNode(\text{'*'}, W.i, F.nptr)$ $W.s = W_1.s$
$W \rightarrow \varepsilon$	$W.s = W.i$
$F \rightarrow \text{id}$	$F.nptr = mkLeaf(\text{id}, \text{id}.entry)$
$F \rightarrow \text{num}$	$F.nptr = mkLeaf(\text{num}, \text{num}.val)$

L属性定义的语法树构造



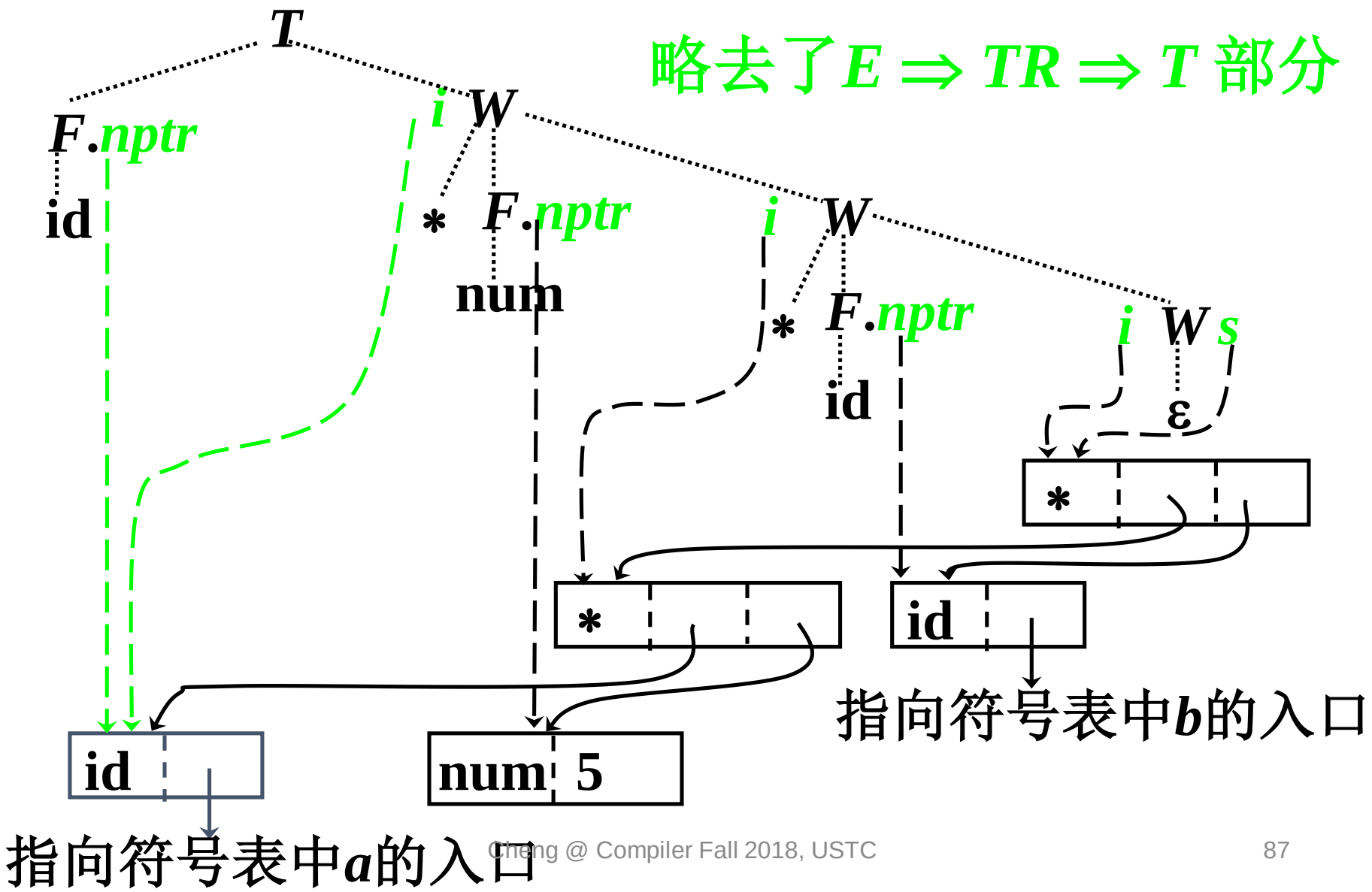
略去了 $E \Rightarrow TR \Rightarrow T$ 部分

L属性定义的语法树构造

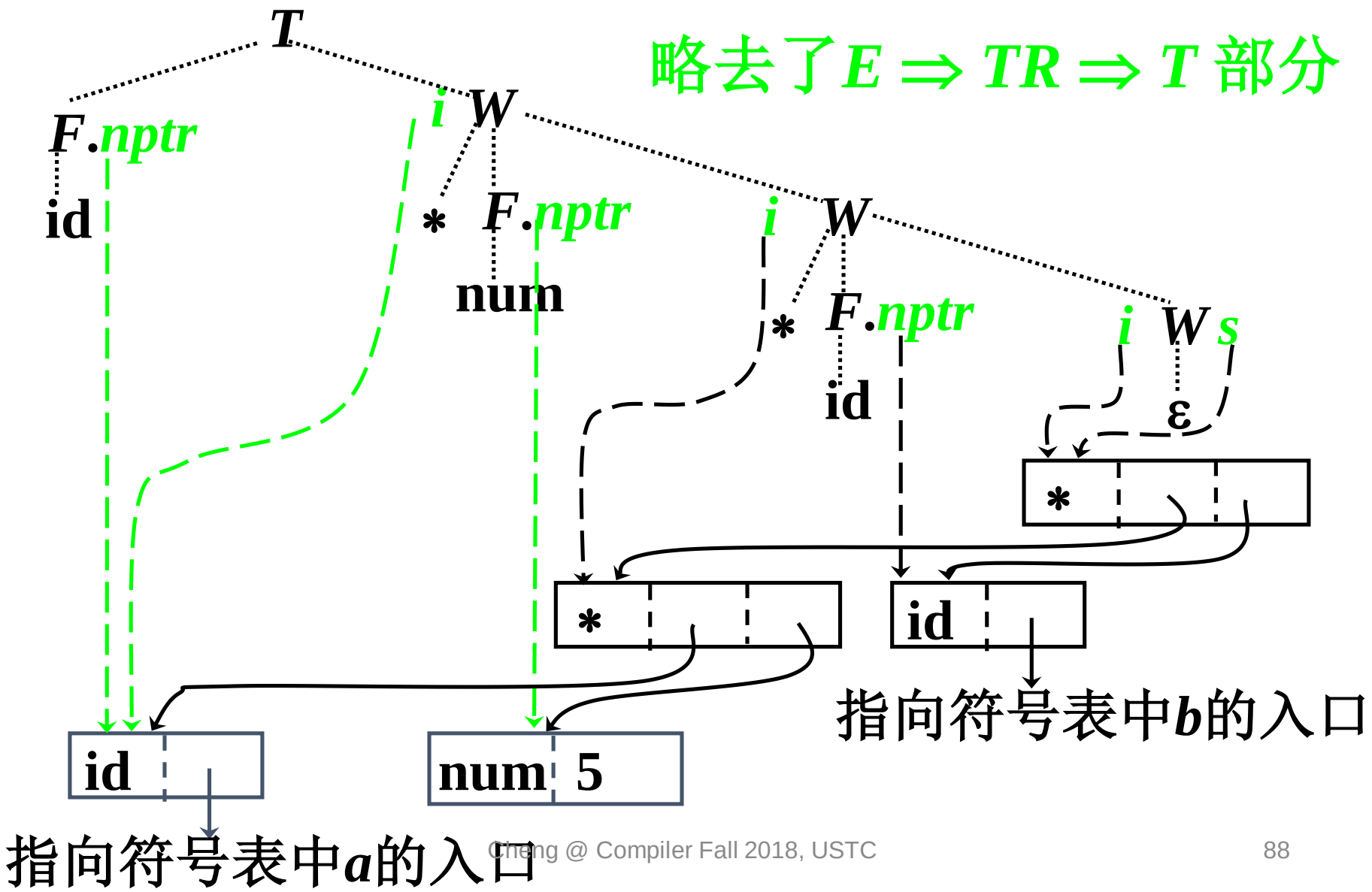


略去了$E \Rightarrow TR \Rightarrow T$部分

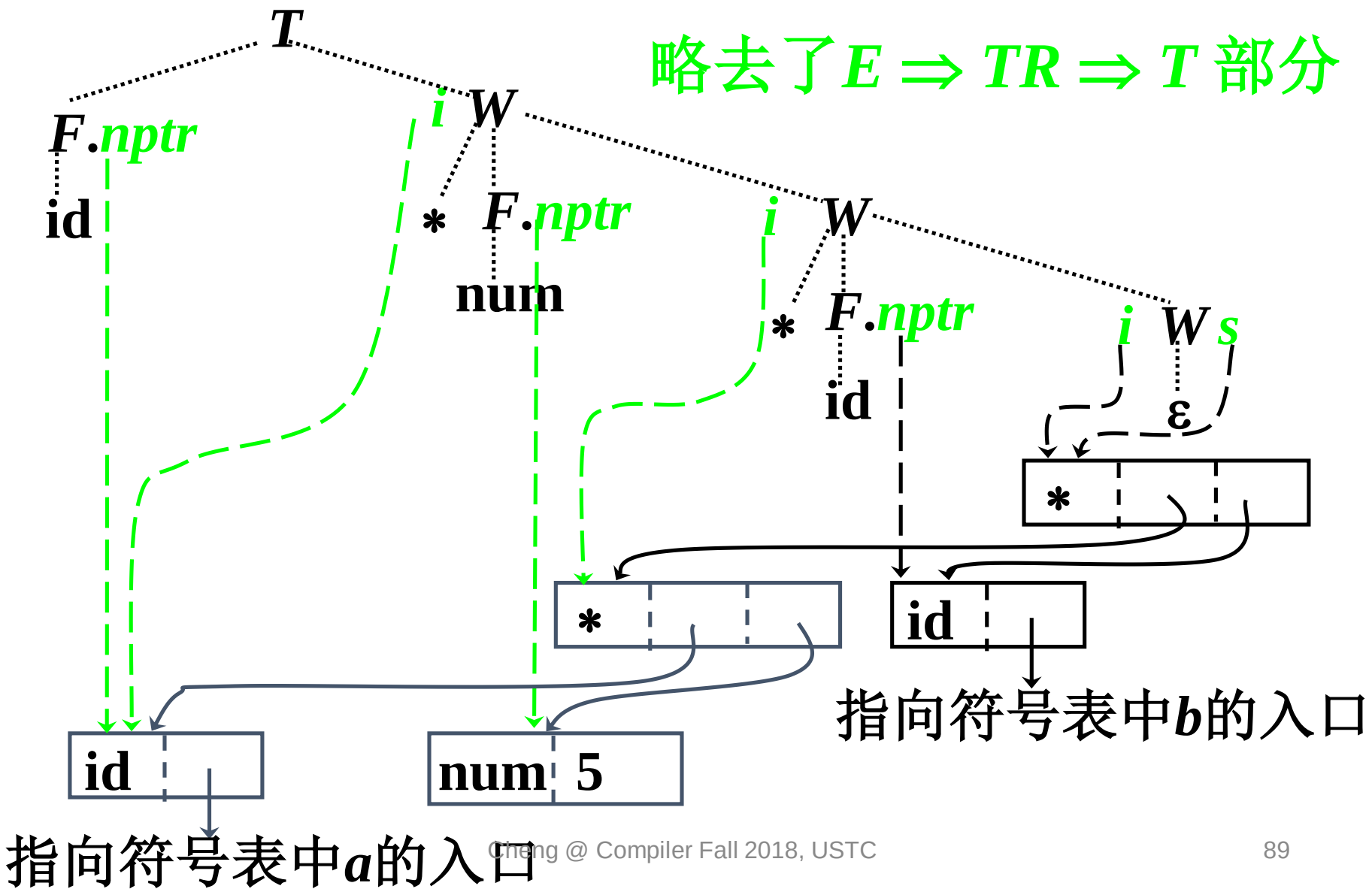
L属性定义的语法树构造



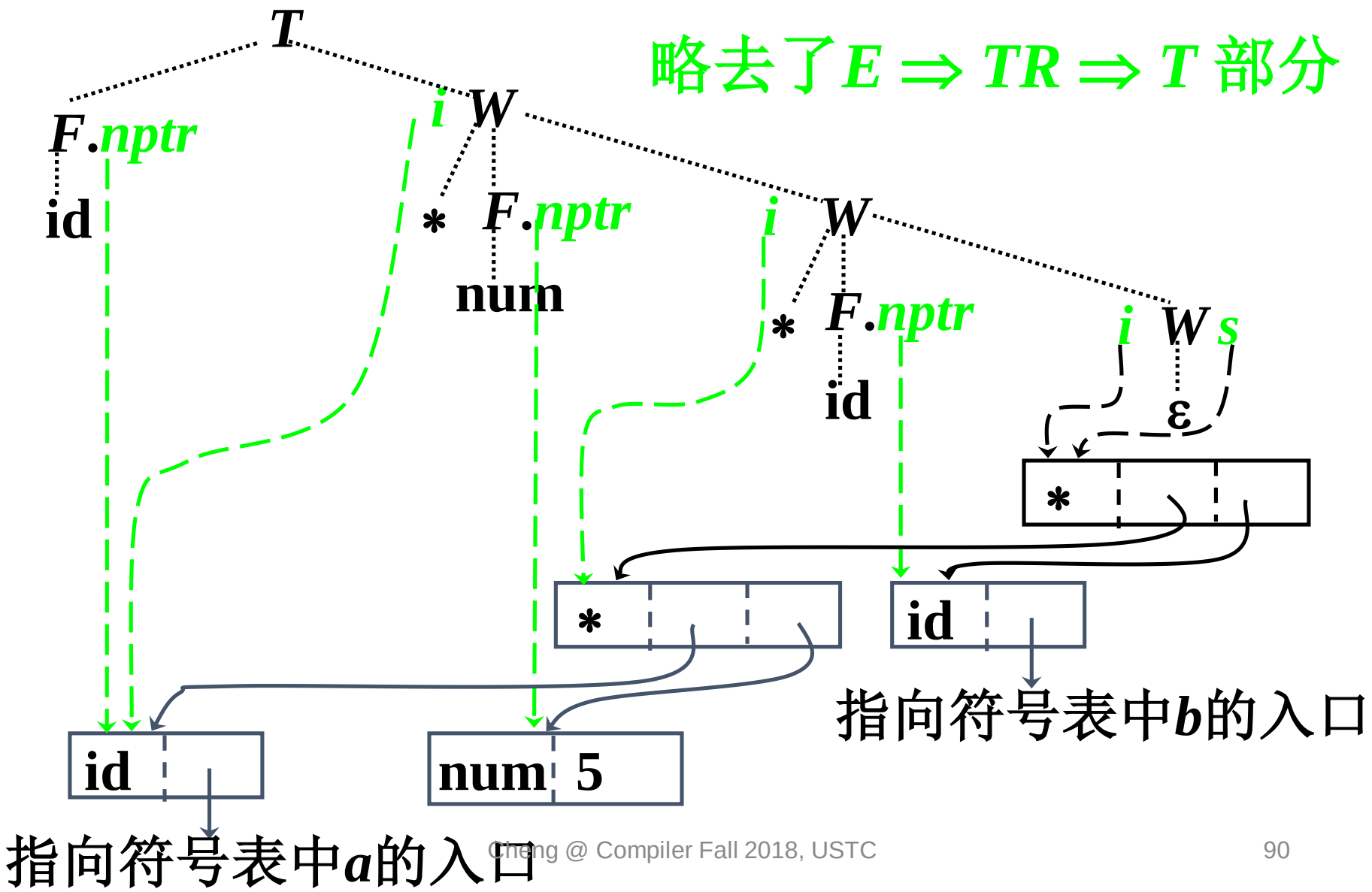
L属性定义的语法树构造



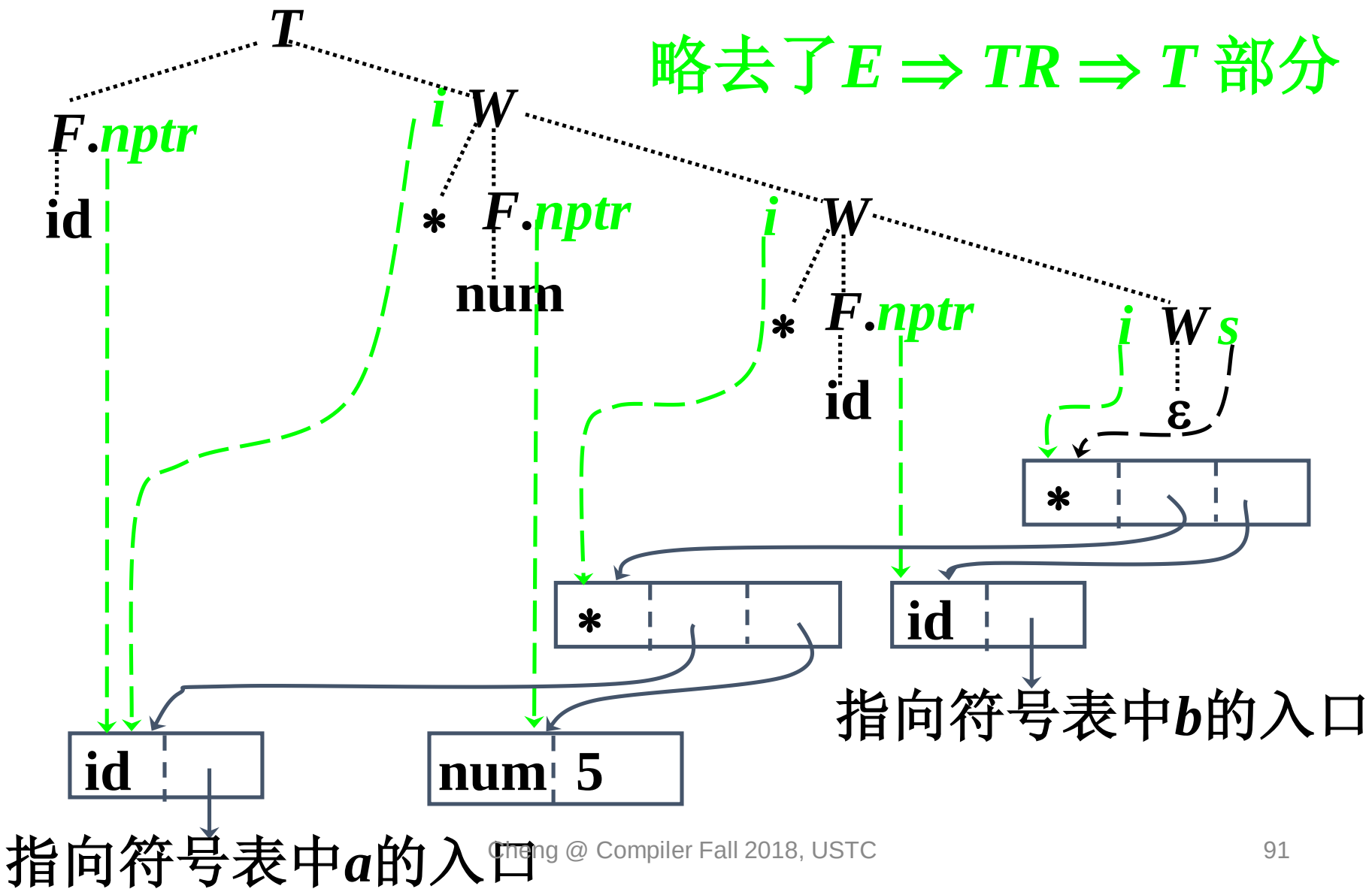
L属性定义的语法树构造



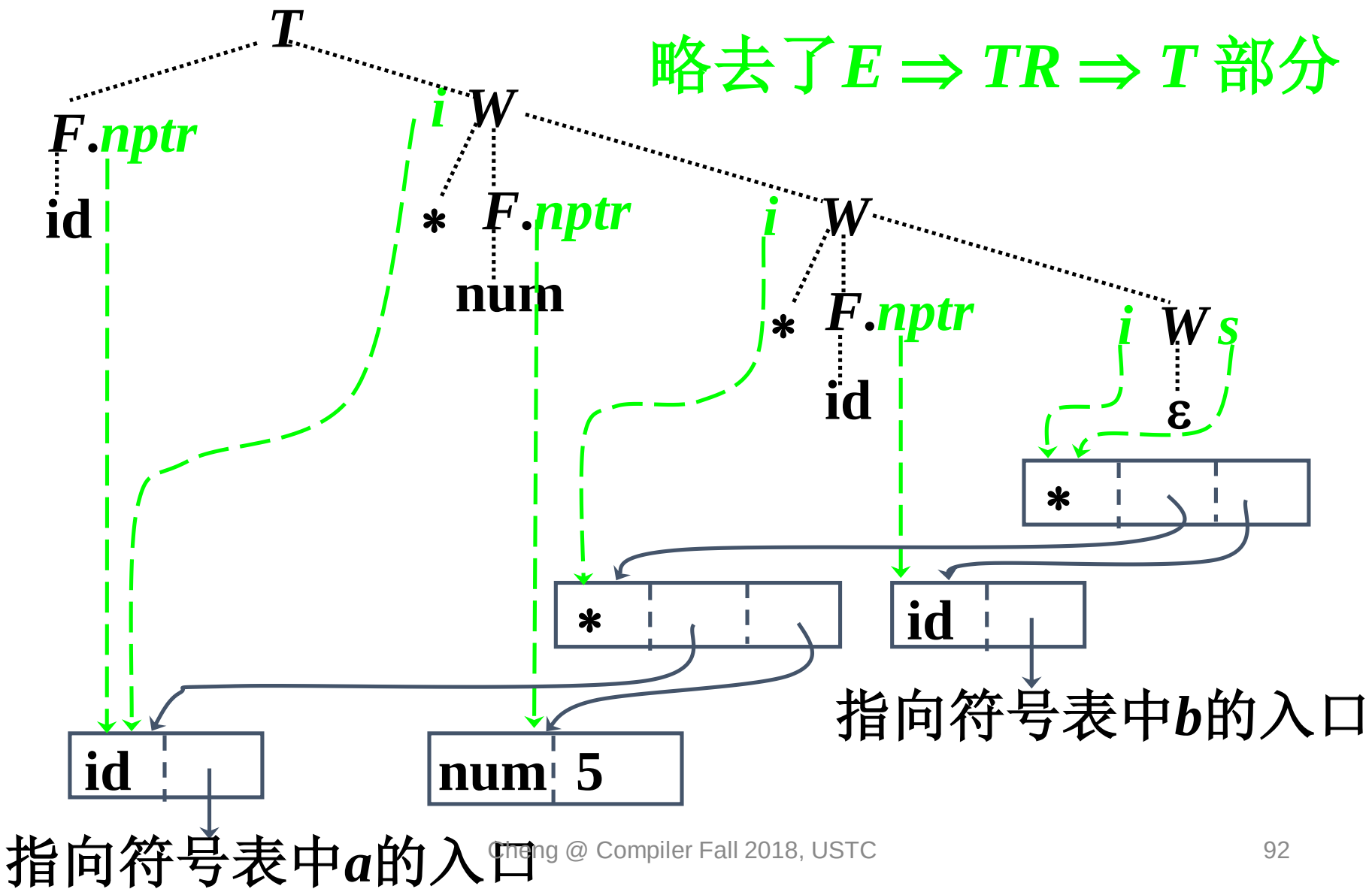
L属性定义的语法树构造



L属性定义的语法树构造



L属性定义的语法树构造



- A Retargetable System-Level DBT Hypervisor
  - USENIX ATC 2019 Best paper
  - 论文地址：https://www.usenix.org/system/files/atc19-spink.pdf
  - 本实验室同学的分享和解读 http://210.45.114.146/wiki/doku.php?id=public:rg:readinggroup

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

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

—— *Elon Musk*