北京大学计算机学院

2024年秋季学期　　《编译原理》

第5章 语法制导翻译（1）

Syntax-Directed Translation

【第2.3节、第5.1, 5.2节】

主要内容

- 语法制导翻译概述
- 语法制导定义（SDD）
  - SDD的求值顺序
- 语法制导翻译的应用
  - 抽象语法树的构造
  - 类型结构
- 语法制导的翻译方案（SDT)
- L属性的SDD的实现方法

语法制导翻译概述

- 语法分析器可以判别输入在语法上的正确性
  - 在语法分析完成之后，一般还需要把输入的源代码翻译为一种目标表示形式。
- 语法制导翻译（syntax-directed translation）指的是编译器在分析过程中执行的工作
  - 首先是语义分析和正确性检查，若正确，则翻译成中间代码或目标代码
  - 文法符号的属性可以描述文法符号的语义
    - 例如，一个变量文法符号的属性可以有变量的类型，层次，存储地址等。表达式的属性有类型，值等。
  - 通过对属性值的计算完成翻译任务

语法制导定义

- **语法制导定义(syntax-directed definition, SDD)**
  - 将文法符号和某些属性相关联
  - 通过语义规则来描述如何计算属性的值

例：将中缀表达式翻译为后缀表达式的SDD

PRODUCTION	SEMANTIC RULES
$expr \rightarrow expr_1 + term$	$expr.t = expr_1.t \mid\mid term.t \mid\mid '+'$
$expr \rightarrow expr_1 - term$	$expr.t = expr_1.t \mid\mid term.t \mid\mid '-'$
$expr \rightarrow term$	$expr.t = term.t$
$term \rightarrow 0$	$term.t = '0'$
$term \rightarrow 1$	$term.t = '1'$
...	...
$term \rightarrow 9$	$term.t = '9'$

语法制导翻译方案

- 语法制导翻译(syntax-directed translation, SDT)
  - 在产生式体中加入语义动作，并在适当的时候执行这些语义动作

例：将中缀表达式翻译为后缀表达式的SDT方案

$expr$	$\rightarrow$	$expr_1$ + $term$	{print('+')}
$expr$	$\rightarrow$	$expr_1$ - $term$	{print('-')}
$expr$	$\rightarrow$	$term$	
$term$	$\rightarrow$	0	{print('0')}
$term$	$\rightarrow$	1	{print('1')}
		...	
$term$	$\rightarrow$	9	{print('9')}

主要内容

- 语法制导翻译概述
- 语法制导定义（SDD）
  - SDD的求值顺序
- 语法制导翻译的应用
  - 抽象语法树的构造
  - 类型结构
- 语法制导的翻译方案（SDT)
- L属性的SDD的实现方法

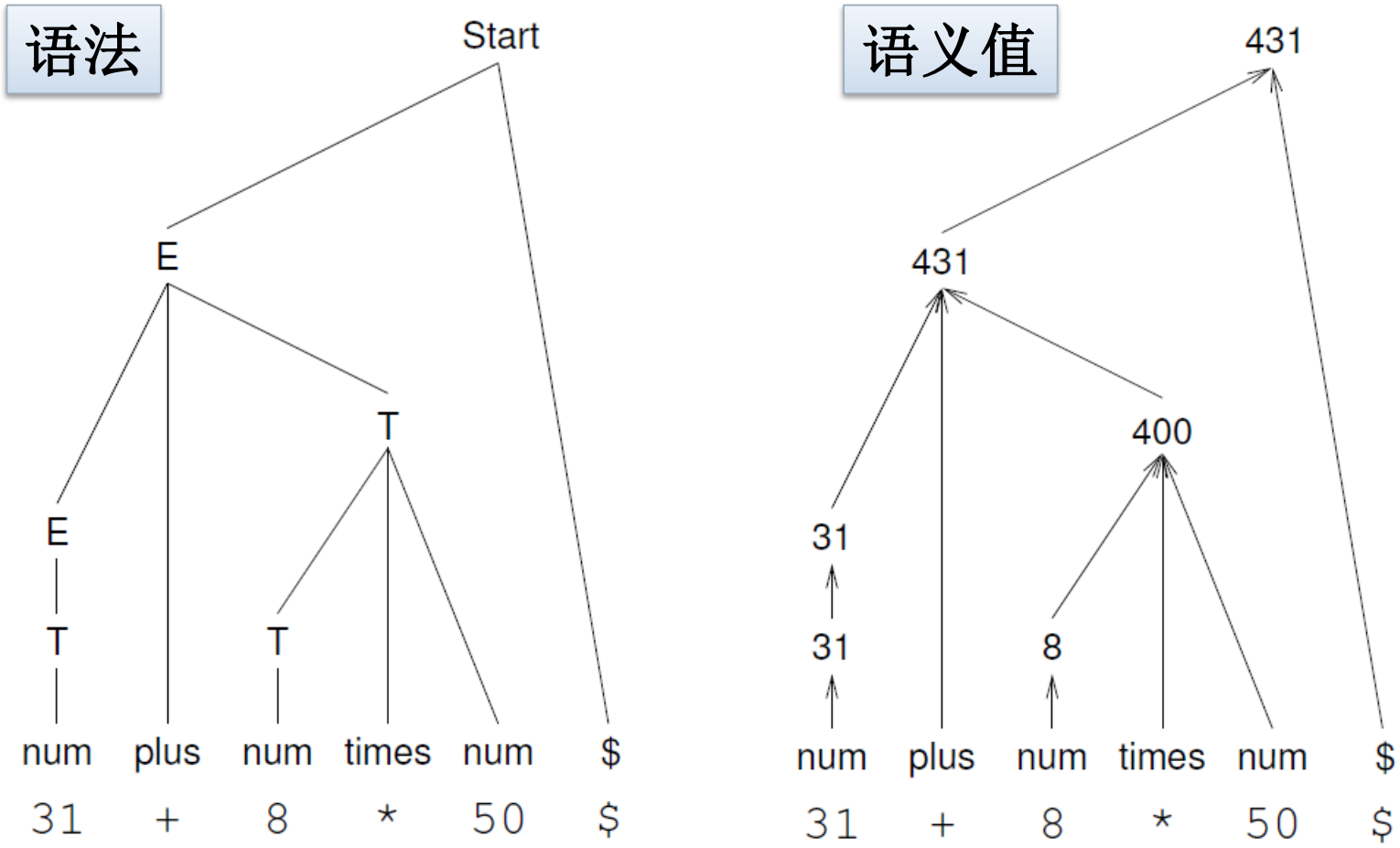
语法制导定义（SDD）

- SDD是上下文无关文法和属性/规则的结合；
  - 属性和文法符号相关联，按照需要来确定各个文法符号需要哪些属性
  - 规则和产生式相关联
- 对于文法符号X和属性a，我们用X.a表示分析树中的某个标号为X的结点的值
- 一个分析树结点和它的分支对应于一个产生式规则，而对应的语义规则确定了这些结点上的属性的取值

属性分类

- 综合属性(Synthesized Attribute)：在分析树结点N上的非终结符号A的属性值由N对应的产生式所关联的语义规则来定义
  - 通过N的子结点或N本身的属性值来定义
- 继承属性(Inherited Attribute)：结点N的属性值由N的父结点所关联的语义规则来定义
  - 依赖于N的父结点、N本身和N的兄弟结点上的属性值
- 不允许N的继承属性通过N的子结点上的属性来定义，但允许N的综合属性依赖于N本身的继承属性
- 终结符号有综合属性（由词法分析获得），但是没有继承属性

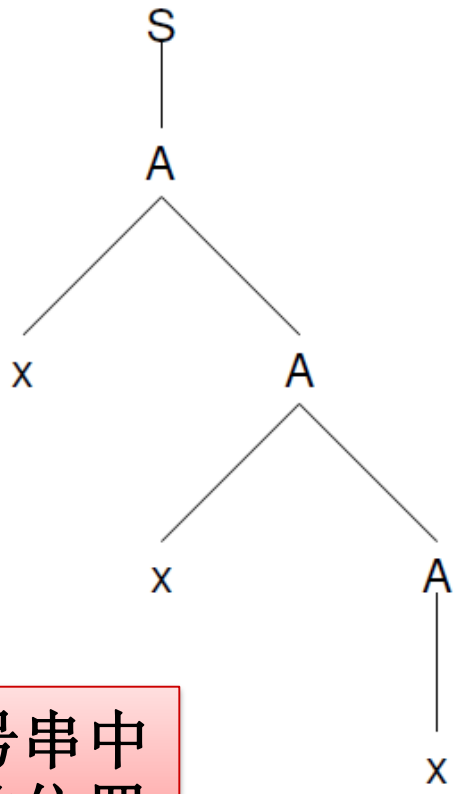
综合属性示例 （表达式文法）



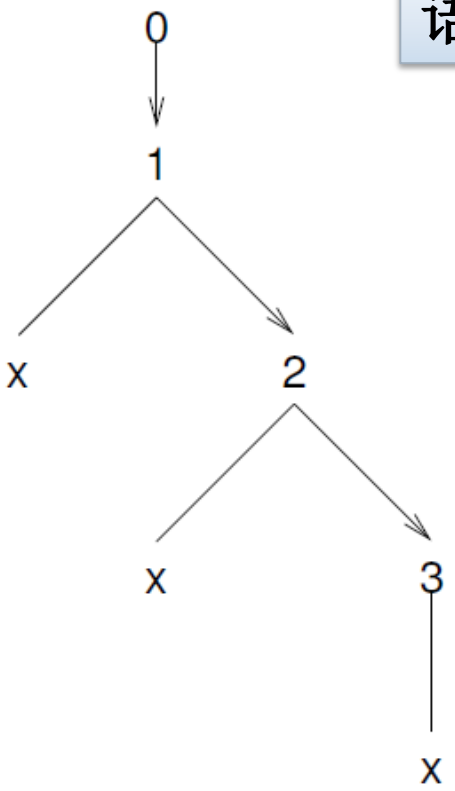
继承属性示例

$A \rightarrow xA \mid x$

语法



语义值



计算符号串中每个 x 的位置

语法制导定义（SDD）的规则

- 语法制导定义的**规则**：
  - $\forall A \rightarrow X_1 X_2 \ldots X_n \in P$，每个规则的一般形式为

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

  - **综合属性**：c是A的一个属性并且$c_1, c_2, \ldots, c_k$是A的继承属性或是某个X_i的属性；
  - **继承属性**：c是某个符号X_i的属性并且$c_1, c_2, \ldots, c_k$是A或X_j的属性。

台式计算器的语法制导定义

对结尾为n的表达式求值。

产生式	语义规则
$L \rightarrow E\ n$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow integer$	$F.val = integer.lexval$

S属性的SDD

- 只包含综合属性的SDD称为**S属性的SDD**。
  - 每个语义规则都根据产生式体中的属性值来计算头部非终结符号的属性值。
- S属性的SDD可以和LR语法分析器一起实现
  - 栈中的状态可以附加相应的属性值
  - 在进行归约时，按照语义规则计算归约得到的符号的属性值
- 语义规则不应该有复杂的副作用
  - 要求副作用不影响其它属性的求值
  - 没有副作用的SDD称为**属性文法(attribute grammar)**

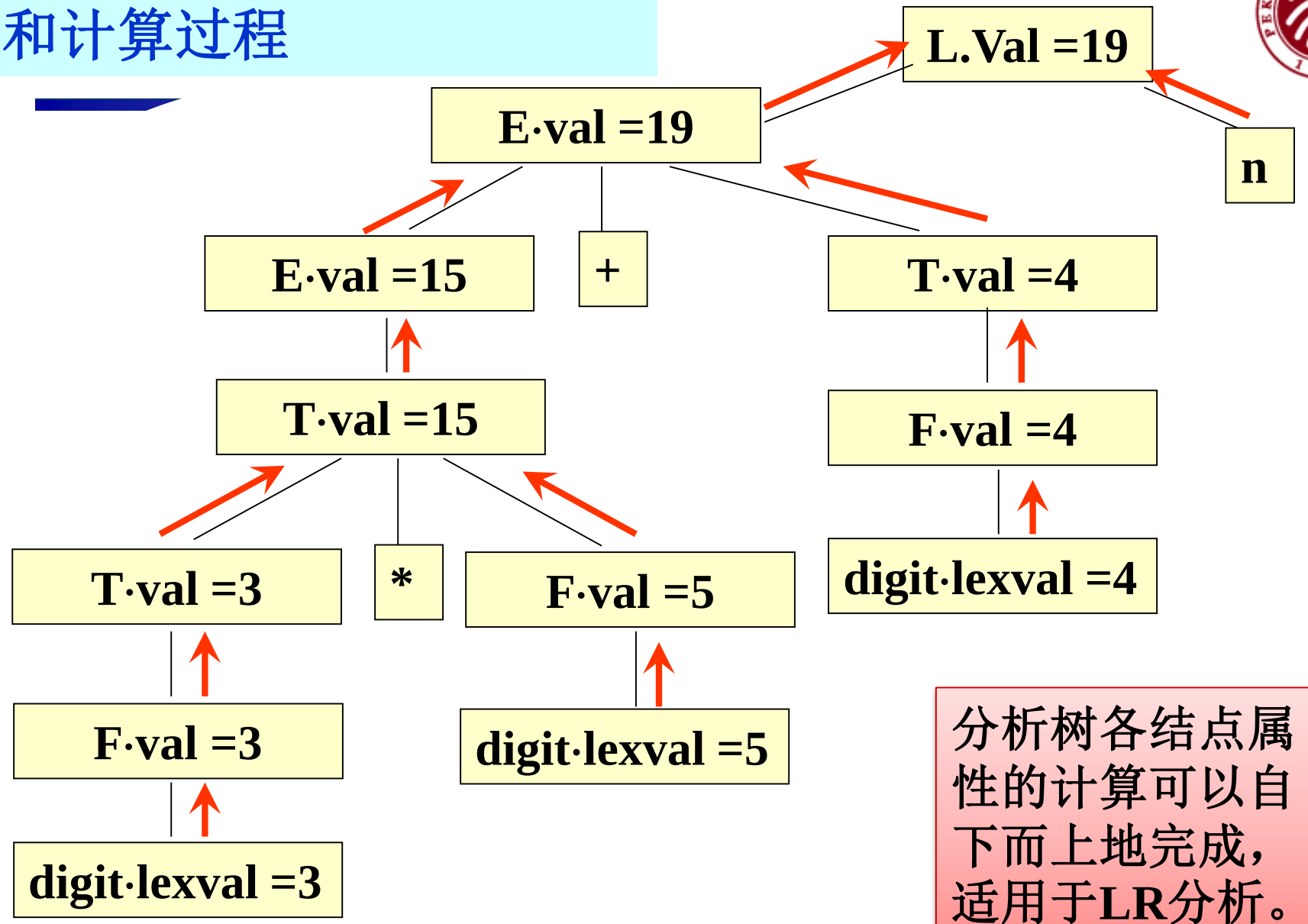
语法分析树上的SDD求值

- 注释语法分析树
  - 包含各个结点的各属性值的语法分析树
- 步骤：
  - 对于任意的输入串，首先构造出相应的分析树
  - 给各个结点（根据其文法符号）加上相应的属性值
  - 按照语义规则计算这些属性值即可
- 按照分析树中的分支对应的文法产生式，应用相应的语义规则计算属性值

SDD求值的计算顺序

- 计算顺序问题
  - 如果某个结点N的属性a为f(N1.b1,N2.b2,…,Nk.bk)，那么我们需要先算出N1.b1,N2.b2,…,Nk.bk的值
- 如果我们可以给各个属性值排出计算顺序，那么这个注释分析树就可以计算得到
  - S属性的SDD一定可以按照自底向上的方式求值
- 例如：下面的SDD不能计算
  - A→B　　A.s=B.i;　　B.i=A.s+1;

3*5+4n的注释语法分析树和计算过程



分析树各结点属性的计算可以自下而上地完成，适用于LR分析。

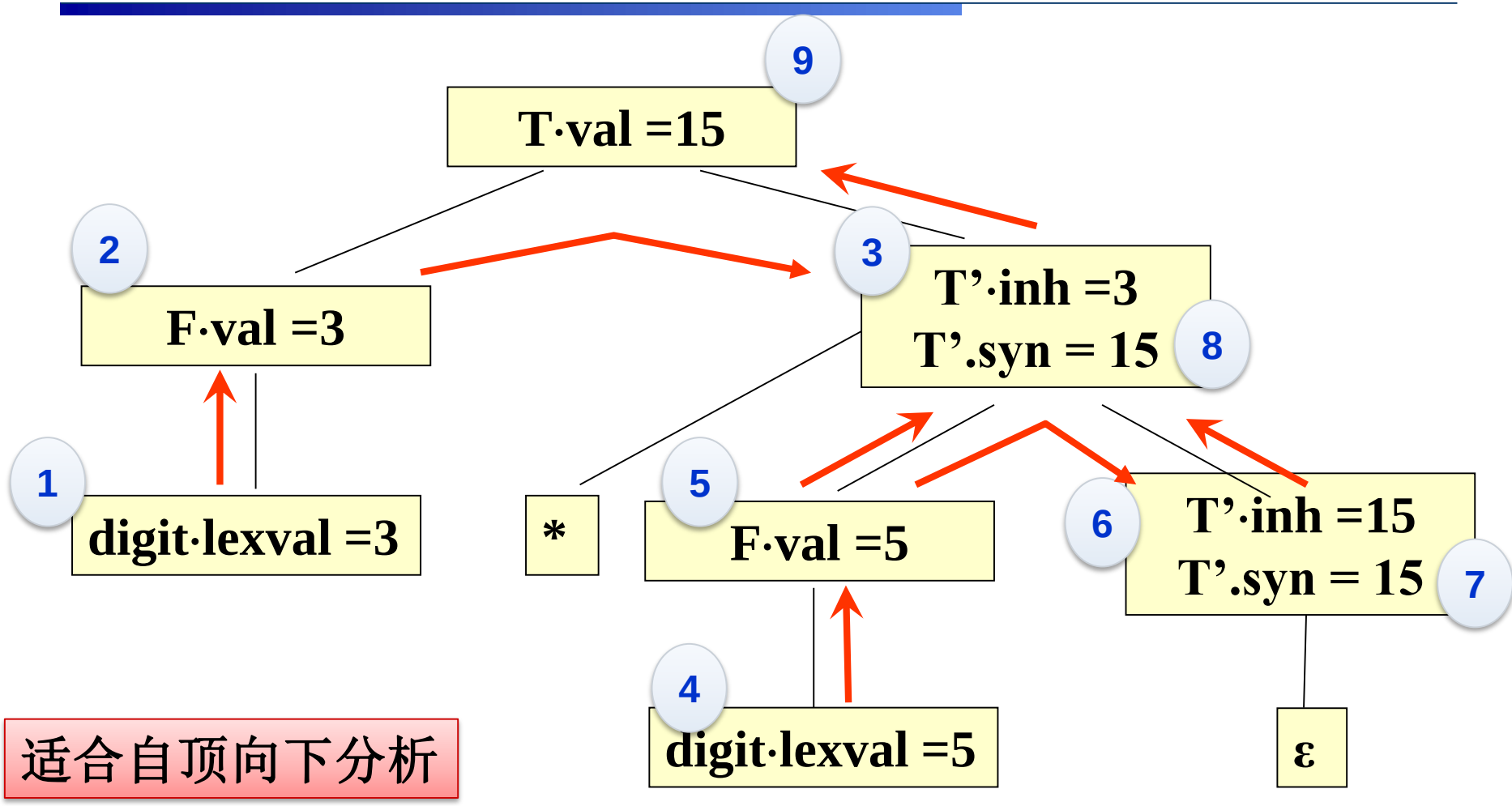
适用于自顶向下分析的SDD

- 前面的表达式文法存在直接左递归，因此无法直接用自顶向下的方法处理。
- 消除左递归之后，我们无法直接使用属性val进行处理：
  - 比如规则：T→FT’　　T’ → *FT’
  - T对应的项中，第一个因子对应于F，而运算符却在T’中。
  - 需要继承属性来完成这样的计算

适合于自顶向下分析的带有继承属性的SDD

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

3*5的注释语法分析树和求值顺序



适合自顶向下分析

消除左递归时如何处理语义规则？

- 原文法
  - $A \rightarrow A_1 Y$　　$A.a = g(A_1.a, Y.y)$
  - $A \rightarrow X$　　$A.a = f(X.x)$

- 消除左递归
  - $A \rightarrow XR$　　$R.i = f(X.x);\ A.a = R.s$
  - $R \rightarrow YR_1$　　$R_1.i = g(R.i, Y.y);\ R.s = R_1.s$
  - $R \rightarrow \varepsilon$　　$R.s = R.i$

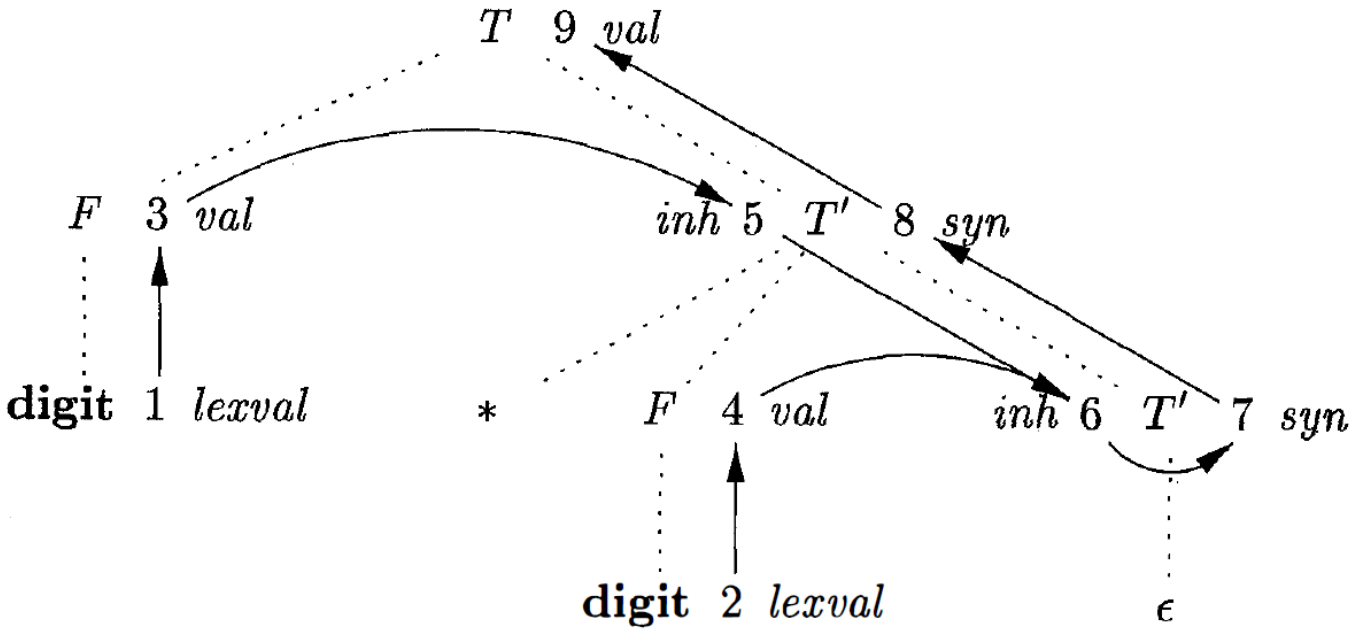
SDD的求值顺序

- 在对SDD的求值过程中，如果结点N的属性a依赖于结点M_1的属性a_1，M_2的属性a_2，….。那么我们必须先计算出M_i的属性，才能计算N的属性a
- 使用依赖图来表示计算顺序
  - 结点：属性值
  - 有向边：若结点m表示的属性值依赖于结点n的属性值，则有从n到m的有向边
- 若依赖图中无环，则存在一个拓扑排序，它就是属性值的计算顺序

依赖图的例子

- 3*5的注释分析树和依赖图
- 可能的计算顺序：
  - 1,2,3,4,5,6,7,8.9
  - 1,3,5,2,4,6,7,8,9

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) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow$ **digit**	$F.val =$ **digit**$.lexval$



属性值的计算顺序

- 按照依赖图的拓扑顺序计算各个属性的值
  - 如果依赖图中存在环，则属性计算无法进行
- 给定一个SDD，很难判定是否存在一棵分析树对应的依赖图是否包含环
- 但是特定类型的SDD一定不包含环，且有固定的排序模式
  - S属性的SDD
  - L属性的SDD
- 对于这些类型的SDD，我们可以确定属性的计算顺序，且可以把不需要的属性（及分析树结点）抛弃以提高效率

S属性的SDD

- 每个属性都是综合属性
  - 都是根据子构造的属性计算出父构造的属性
  - 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。
- 可用于自顶向下和自底向上的语法分析
  - 自顶向下：
    - 递归子程序法中，在过程A()的最后计算A的属性
    - 此时A调用的其他过程已经调用完毕
  - 自底向上：
    - 在构造分析树的结点的同时计算相关的属性
    - 此时其子结点的属性必然已经计算完毕

L-属性的SDD

- 语义规则中的每个属性可以是
  - 综合属性，
  - 继承属性，但是$\forall A \rightarrow X_1 X_2 \ldots X_n \in P$, X_j $(1 \leq j \leq n)$的继承属性仅依赖于 产生式中X_j的左边符号X_1, X_2, …, X_{j-1}的属性,和 A的继承属性。
- 依赖图的边：
  - 继承属性从左到右，从上到下。
  - 综合属性从下到上
- 每一个S-属性的SDD都是L-属性的SDD。
- L-属性的SDD可用于按深度优先顺序来计算

L属性SDD和自顶向下语法分析

- 在递归子程序中实现L属性
  - 对于每个非终结符号A，其对应的函数的参数为继承属性，返回值为综合属性
- 在处理规则$A \rightarrow X_1 X_2 \ldots X_n$时
  - 在调用$X_i()$之前计算X_i的继承属性值，然后以它们为参数调用$X_i()$；
  - 在产生式对应代码的最后计算A的综合属性
  - 注意：如果所有的文法符号的属性计算按上面的方式进行，计算顺序必然和依赖关系一致

L属性SDD的例子

产生式	语义规则
1) $T \to F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \to \epsilon$	$T'.syn = T'.inh$
4) $F \to \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

递归子程序法中实现L属性SDD

```
int T( )
{
    if(nextToken == digit)
        return T1( F() );
    else   error();
}
int T1 ( int T1inh)
{
    if(nextToken == '*' )
    {    match ('*');
         return T1( T1inh * F() );
    }
    else   { return T1inh; }
}
int F( )
{    if (nextToken == digit) return digit.lexval;   }
```

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) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow$ **digit**	$F.val =$ **digit**$.lexval$

非L-属性SDD的例子

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

上表的语法制导定义定义不是L-属性SDD。文法符号Q的继承属性依赖于它右边文法符号R的属性。

具有受控副作用的语义规则

- 副作用：在SDD中添加除求值之外的动作
- 例1：L→E n　　print(E.val)
  - 在桌面计算器的语义规则中添加打印语句
  - 通过副作用打印出E的值
  - 总是在最后执行，不会影响其它属性的求值
- 例2：变量声明的SDD中的副作用

变量声明的SDD

产生式	语义规则
D→ TL	L.inh = T.type
T → int	T.type = integer
T → float	T.type = float
L → L_1, id	L_1.inh =L.inh
	addType (id.entry, L.inh)
L → id	addType(id.entry, L.inh)

- addType函数将标识符的类型信息加入到标识符表中

作业

- 11月5日交
- 语法制导定义
  - 5.1.2, 5.1.3(1)
  - 5.2.3, 5.2.4, 5.2.5