中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle

编 译 原 理

第五章 语法制导翻译（2）

郑馥丹
zhengfd5@mail.sysu.edu.cn

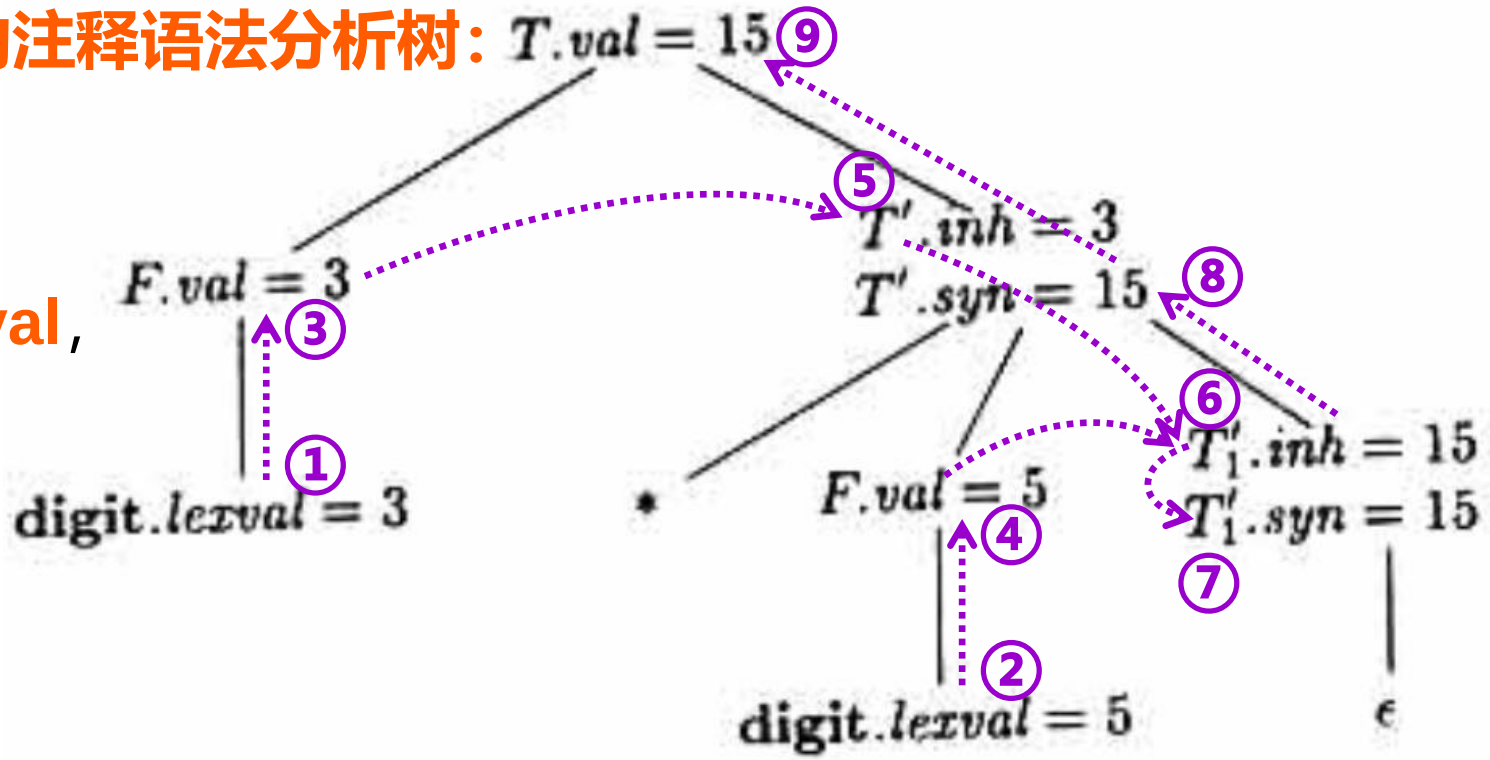
CONTENTS

目 录

1. 依赖图

- 依赖图[dependency graph]:
  - 确定一棵语法分析树中各个属性的求值顺序
  - 描述了某个语法分析树中的属性之间的信息流
  - 从一个属性到另一个的边表示计算第二个属性时需要第一个属性的值

3*5的注释语法分析树:



T→FT’ ⑤{ **T’.inh=F.val**,
⑨ T.val=T’.syn }

T’→*FT_1’ ⑥{ **T_1’.inh=T’.inh×F.val**,
⑧ T’.syn=T_1’.syn }

T’→ε ⑦{ T’.syn=T’.inh }

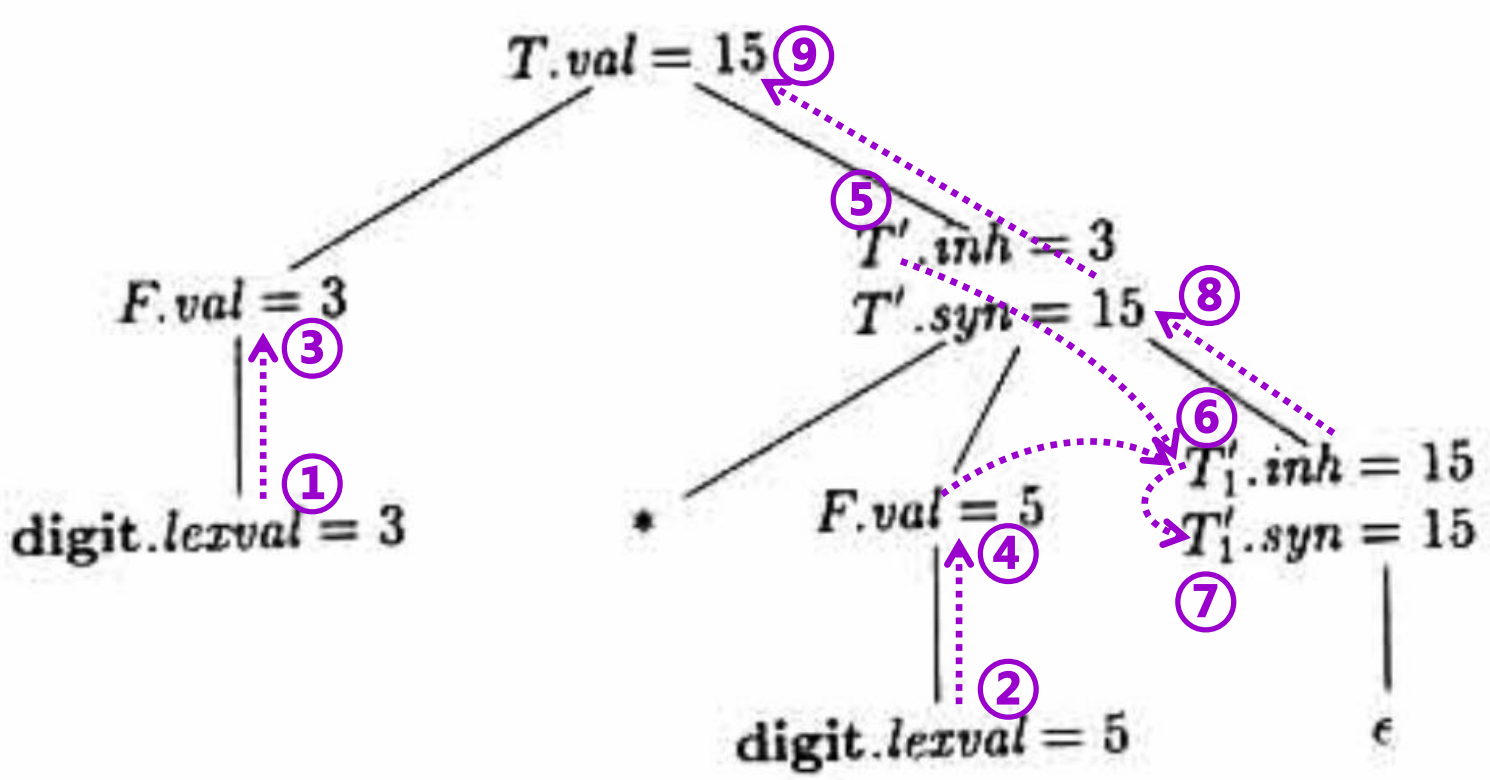
F→digit ③④{ F.val=digit.lexval }

2. 属性求值的顺序

- 依赖图刻画了对一棵语法分析树中不同结点上的属性求值时可能采取的顺序
- 如果依赖图中有一条从结点M到结点N的边，那么要先对M对应的属性求值，再对N对应的属性求值
- 可行的求值顺序就是满足下列条件的结点顺序$N_1,N_2,\ldots N_k$，如果有一条从结点N_i到N_j的依赖图的边，则i<j，这个排序称为这个图的**拓扑排序(topological sort)**
- **若图中有环，则不存在拓扑排序**
- **有向无环图(Directed Acyclic Graph，DAG)则至少存在一个拓扑排序**

2. 属性求值的顺序

- 同一个依赖图可能存在多个拓扑排序
- 例：右图的拓扑排序：
  - ①②③④⑤⑥⑦⑧⑨
  - ①③⑤②④⑥⑦⑧⑨
  - ……



3. 抽象语法树[Abstract Syntax Tree, AST]

- 抽象语法树比语法分析树（Parse Tree）更简洁，直接反映了源代码的语法结构，同时剔除了无关的语法细节（如分号、括号等）
- 例：对于代码a=3+5*2;
  - 语法分析树：会包含全部结点（=，+，*，；）
  - 抽象语法树：仅保留关键结构：

3. 抽象语法树[Abstract Syntax Tree, AST]

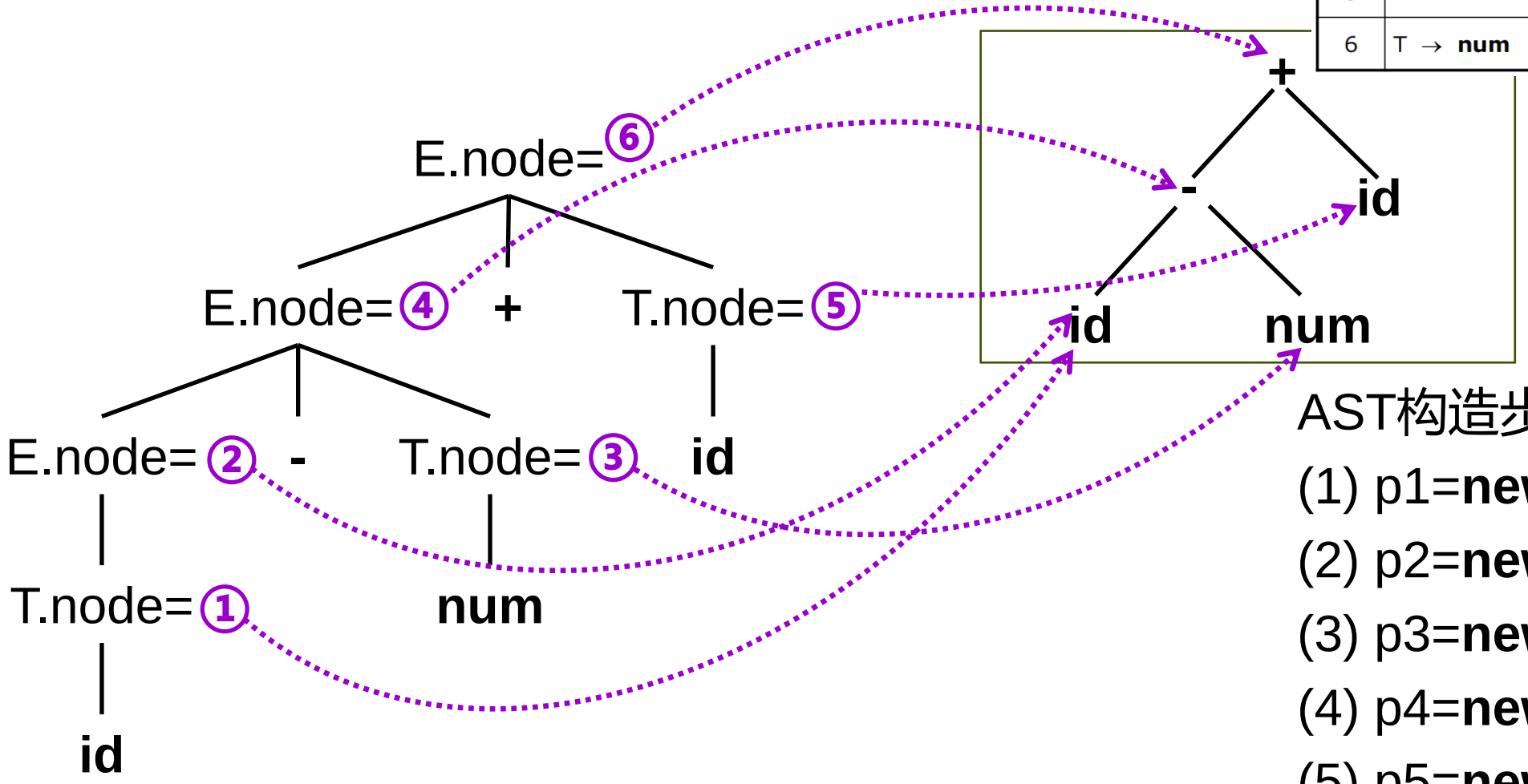
- 例：文法

No.	Productions	Semantic Rules
1	$E \rightarrow E_1$ **+** T	E.node = **new** Node('+', E_1.node, T.node)
2	$E \rightarrow E_1$ **–** T	E.node = **new** Node('–', E_1.node, T.node)
3	E → T	E.node = T.node
4	T → **(** E **)**	T.node = E.node
5	T → **id**	T.node = **new** Leaf(**id**, **id**.entry)
6	T → **num**	T.node = **new** Leaf(**num**, **num**.val)

3. 抽象语法树[Abstract Syntax Tree, AST

- **a-4+c的抽象语法树**

No.	Productions	Semantic Rules
1	E → E_1 + T	E.node = **new** Node('+', E_1.node, T.node)
2	E → E_1 – T	E.node = **new** Node('–', E_1.node, T.node)
3	E → T	E.node = T.node
4	T → (E)	T.node = E.node
5	T → **id**	T.node = **new** Leaf(**id**, **id**.entry)
6	T → **num**	T.node = **new** Leaf(**num**, **num**.val)



AST构造步骤：

(1) p1=**new** Leaf(**id**, entry-a)

(2) p2=**new** Leaf(**num**, 4)

(3) p3=**new** Node('**-**', p1, p2)

(4) p4=**new** Leaf(**id**, entry-c)

(5) p5=**new** Node('**+**', p3, p4)

4. S-属性的定义

- 一个仅包含综合属性[Synthesized attribute]的SDD称为**S-属性[S-attribute]**的SDD，或称S-属性文法
  - **每个属性都必须是综合属性**
  - 每个节点的属性值**仅由其子节点的属性值计算**而来（自底向上传递信息）
  - 可以保证**求值顺序与LR分析的输出顺序相同**
  - 可按照语法分析树结点的任何自底向上顺序来计算属性值：
    - ✓ **后序**遍历

```
postorder(N){
    for(从左边开始，对N的每个子结点C) postorder(C);
    对N关联的各个属性求值;
}
```

    - ✓ 当遍历最后一次离开某个结点N时计算出N的各个属性值

4. S-属性的定义

- 例：S-属性SDD
  - E→E1+T{E.val=E1.val+T.val}
  - E→T{E.val=T.val}
  - T→num{T.val=num.val}

算术表达式3+5：	
LR分析过程	**S属性求值顺序**
扫描3，归约T→3	计算T.val = 3
归约E→T	计算E.val=T.val=3
扫描+，移进	
扫描5，归约T→5	计算T.val = 5
归约E→E1+T	计算E.val = 3 + 5 = 8

- 例：非S-属性SDD
  - 若SDD中包含继承属性，如： A→BC { **B.in=A.in**, **C.in=B.s**, A.s=C.s }
  - 则B.in依赖A.in（父节点），C.in依赖B.s（左边的兄弟节点）
  - 这种计算顺序不能直接用LR分析，因为LR分析是严格自底向上的，无法在归约时访问父节点或右边兄弟节点的属性

5. L-属性的定义

- 一个SDD的产生式右部所关联的各个属性之间，依赖图的边总是从左到右[Left-to-right]，则称该SDD为**L-属性[L-attribute]**的SDD
- L-属性[L-attribute]的SDD中，每个属性：
  - 要么是一个综合属性
  - 要么是一个继承属性，但有以下约束：假设存在产生式$A \rightarrow X_1 X_2 \ldots X_n$，其$X_i$的继承属性$X_i.a$的计算规则只能依赖：
    - ✓ 产生式左部A的继承属性
    - ✓ X_i左边的文法符号X_1、X_2、…、X_{i-1}的继承属性或综合属性（即已经计算过的兄弟节点）
    - ✓ X_i本身的属性，且由X_i的属性组成的依赖图中不存在环

5. L-属性的定义

- 即：L-属性定义的目的是**确保属性计算可以按从左到右的顺序进行**：
  - **继承属性不能依赖右边的符号**，否则计算顺序会混乱（因为右边的符号可能还未被处理）
  - 不能有循环依赖，否则无法确定计算顺序
- 适用于：
  - **递归下降、LL(1)等自顶向下分析方法**
  - **LR(1)等自底向上的分析方法**

5. L-属性的定义

- 例：L-属性SDD
  - D→Tid{id.in_type=T.type，D.type=T.type}
  - id的属性in_type是继承属性，只依赖左边的T的属性
  - D的属性type是综合属性
  - 没有循环依赖

- 例：非L-属性SDD
  - A→BC{B.in=C.s}
  - B的属性in是继承属性，且依赖右边的C

随堂练习（2）

- 假设我们有一个产生式A→BCD，其中，A、B、C、D这四个非终结符都有两个属性，s是一个综合属性，i是一个继承属性，对于下面每组规则，指出这些规则是否满足S-属性定义的要求，是否满足L-属性定义的要求？

  (1) A.s=B.i+C.s

  (2) A.s=B.i+C.s，D.i=A.i+B.s

  (3) A.s=B.s+D.s

规则	S-属性定义	L-属性定义	原因
(1) A.s=B.i+C.s	❌ 否	✅ 是	依赖继承属性B.i，但L-属性允许，因B.i来源合法（非由右边C而来）
(2) A.s=B.i+C.s, D.i=A.i+B.s	❌ 否	✅ 是	继承属性D.i依赖父节点和左边兄弟节点，符合L-属性
(3) A.s=B.s+D.s	✅ 是	✅ 是	仅依赖综合属性，符合S-属性和L-属性

CONTENTS

目 录

1. 语法制导的翻译方案

- 语法制导翻译的工作步骤
  - ① 向语法符号引入属性
  - ② 为每个产生式定义**语义规则**
  - ③ 根据注释语法分析树绘制依赖图
  - ④ 依赖图的拓扑排序确定评估顺序
  - ⑤ 按照求值顺序执行**语义规则**

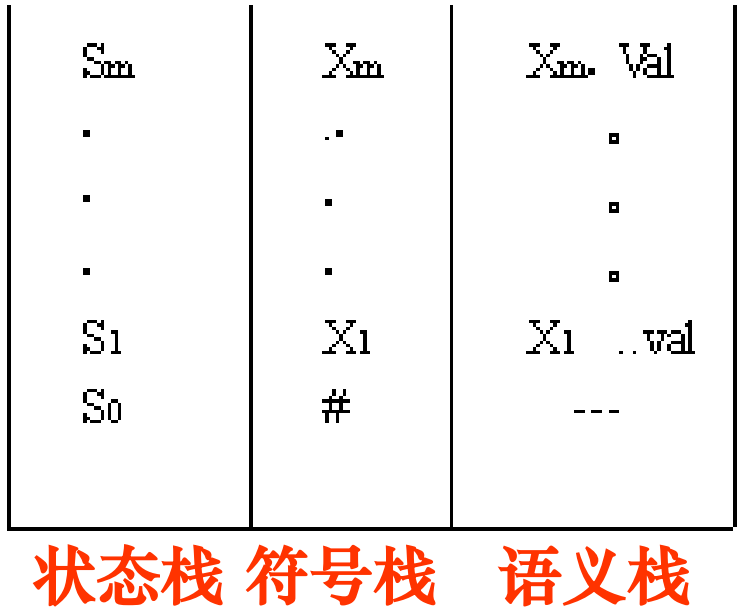
1. 语法制导的翻译方案

- 语法制导的翻译方案[Syntax-Directed Translation Schemes, SDT]
  - 在产生式右部嵌入了程序片段的上下文无关文法
  - 这些程序片段称为**语义规则**（语义动作）

- **自底向上分析**对应**S-属性翻译方案**

2．S-属性翻译方案

- 以LR分析为例：
  - 将LR分析器能力扩大，增加在归约后**调用语义规则**的功能
  - **语义动作是在每一步归约之后执行的**
  - **增加语义栈**，语义值放到与符号栈同步操作的语义栈中，多项语义值可设多个语义栈，栈结构为：



2. S-属性翻译方案

- 例：简单算术表达式求值的属性文法

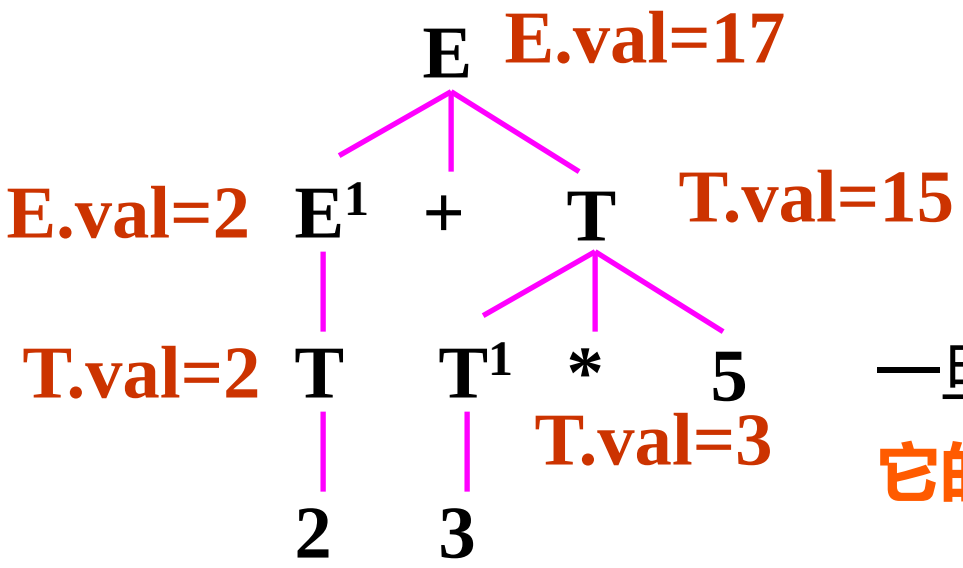
1）E→E1+T　　{ E.val = E1.val +T.val }

2）E→T　　{ E.val = T.val }

3）T→T1*digit　{ T.val = T1.val * digit.lexval }

4）T→digit　　{ T.val = digit.lexval }

2+3*5的语法树：　　**自下而上语法制导翻译过程：**



一旦语法分析确认输入符号串是一个句子，

它的值也同时由语义规则计算出来

2. S-属性翻译方案

- 例：简单算术表达式求值的属性文法

1）E→E1+T　　{ E.val = E1.val +T.val }

2）E→T　　{ E.val = T.val }

3）T→T1*digit　{ T.val = T1.val * digit.lexval }

4）T→digit　　{ T.val = digit.lexval }

状态	ACTION				GOTO	
	d	+	*	#	E	T
0	S_3				1	2
1		S_4		acc		
2		r_2	S_5	r_2		3
3		r_4	r_4	r_4		
4	S_3					7
5	S_6					
6		r_3	r_3	r_3		
7		r_1	S_5	r_1		

程序片段：

```
E→E1+T {stack[top-2].val=stack[top-2].val+stack[top].val;
        top=top-2}
E→T
T→T1*digit  {stack[top-2].val=stack[top-2].val*digit.lexval;
             top=top-2}
T→digit
```

2．S-属性翻译方案

1）E→E1+T　{ E.val = E1.val +T.val }
2）E→T　　{ E.val = T.val }
3）T→T1*digit { T.val = T1.val * digit.lexval}
4）T→digit　　{ T.val = digit.lexval }

E→E1+T {stack[top-2].val=stack[top-2].val+stack[top].val;
top=top-2}
E→T
T→T1*digit {stack[top-2].val=stack[top-2].val*digit.lexval;
top=top-2}
T→digit

状态	ACTION				GOTO	
	d	+	*	#	E	T
0	S_3				1	2
1		S_4		acc		
2		r_2	S_5	r_2		3
3		r_4	r_4	r_4		
4	S_3					7
5	S_6					
6		r_3	r_3	r_3		
7		r_1	S_5	r_1		

分析并计算2＋3*5的过程:

步骤	状态栈	语义栈	符号栈	剩余输入串	Action	GOTO
0	0	-	#	2＋3*5#	S_3	
1	03	- -	#2	＋3*5#	r_4	2
2	02	-2	#T	＋3*5#	r_2	1
3	01	-2	#E	＋3*5#	S_4	
4	014	-2-	#E+	3*5#	S_3	
5	0143	-2- -	#E+3	*5#	r_4	7
6	0147	-2-3	#E+T	*5#	S_5	
7	01475	-2-3-	#E+T*	5#	S_6	
8	014756	-2-3--	#E+T*5	#	r_3	7
9	0147	-2-15	#E+T	#	r_1	1
10	01	-17	#E	#	acc	