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（3）

郑馥丹
zhengfd5@mail.sysu.edu.cn

CONTENTS

目 录

1. L-属性定义

- 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 **不依赖于父亲节点的综合属性**
    - ✓ 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） **不依赖于右兄弟的任何属性**
    - ✓ 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

不能有循环依赖

2. L-属性SDD的求值顺序

- L-属性SDD的求值顺序
  - 与**深度优先[Depth-First]**访问语法分析树相同
  - 与**自顶向下**解析的顺序相同

```
void dfvisit(n: Node) {
    for (each child m of n, from left to right) {
        evaluate inherited attributes of m;
        dfvisit(m);
    }
    evaluate synthesized attributes of n;
}
```

3. L-属性翻译方案

- 翻译方案与L-属性定义
  - 翻译方案中的**显式**求值顺序
  - 按照**从左到右深度优先**的顺序执行语义操作

3. L-属性翻译方案

- 语法制导翻译方案[Syntax-Directed Translation Schemes, **SDT**]
  - 后缀翻译方案[Postfix translation scheme]/**后缀SDT**：所有语义动作都在**产生式的最右端**

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 }

3. L-属性翻译方案

- 语法制导翻译方案[Syntax-Directed Translation Schemes, **SDT**]
  - 产生式内部带有语义动作的SDT：语义动作可以放置在**产生式右部的任何位置**上[Actions inside productions]
    - ✓ 若有产生式B→X**{a}**Y：
      - 如果X是终结符，则识别到X后，动作a就会执行
      - 如果X是非终结符，则识别到所有从X推导出的终结符后，动作a就会执行
      - 如果语法分析是自底向上的，则在X出现在分析栈的栈顶时，动作a就会执行
      - 如果语法分析是自顶向下的，则在推导展开Y（Y为非终结符）或者在输入中检测到Y（Y为终结符）之前，动作a就会执行

3. L-属性翻译方案

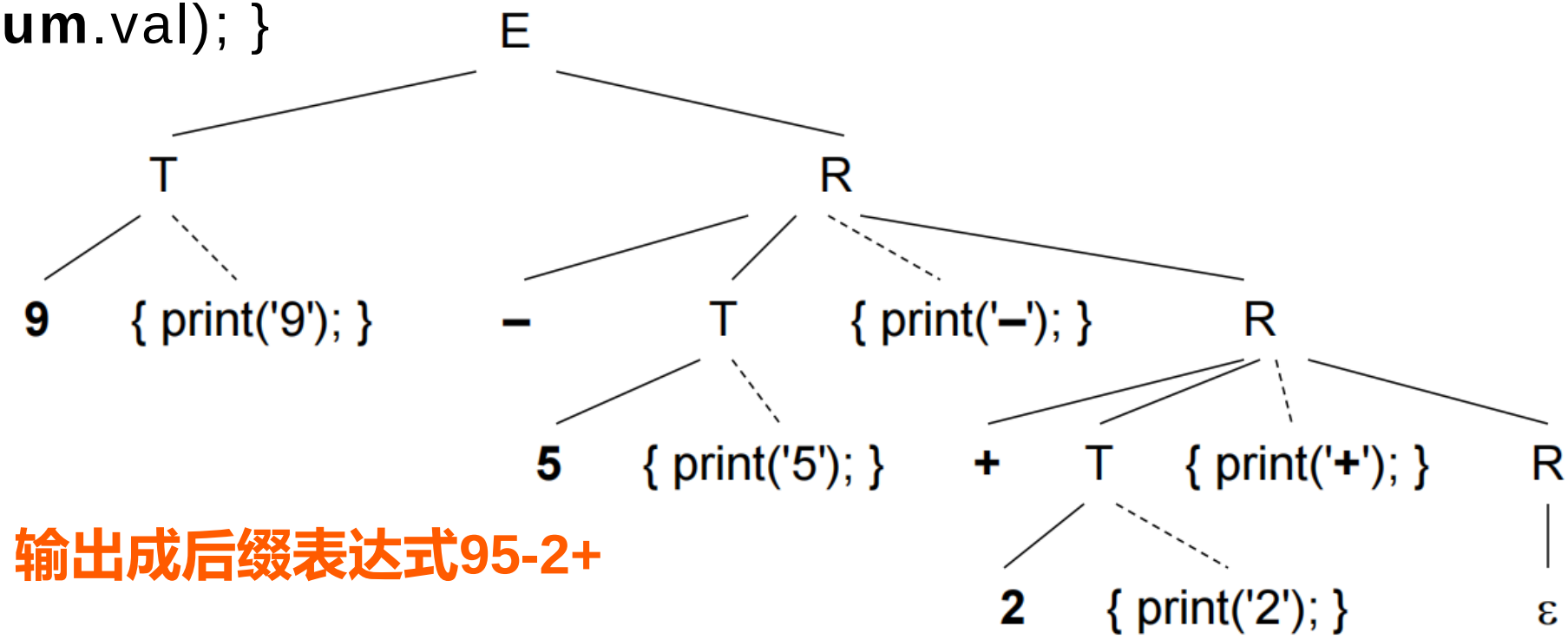
- 例：将中缀表达式转成后缀表达式的SDT

$E \rightarrow TR$

$R \rightarrow$ **addop** T **{ print(addop.lexeme); }** R_1　　**产生式内部带有语义动作**

$R \rightarrow \varepsilon$

$T \rightarrow$ **num** { print(**num**.val); }



将中缀表达式9-5+2，输出成后缀表达式95-2+

3. L-属性翻译方案

- 注意：**不是所有的SDT都可以在语法分析过程中实现**
- 例：将中缀表达式转成前缀表达式的SDT

L → E **n**

E → **{ print('+'); }** E1 + T

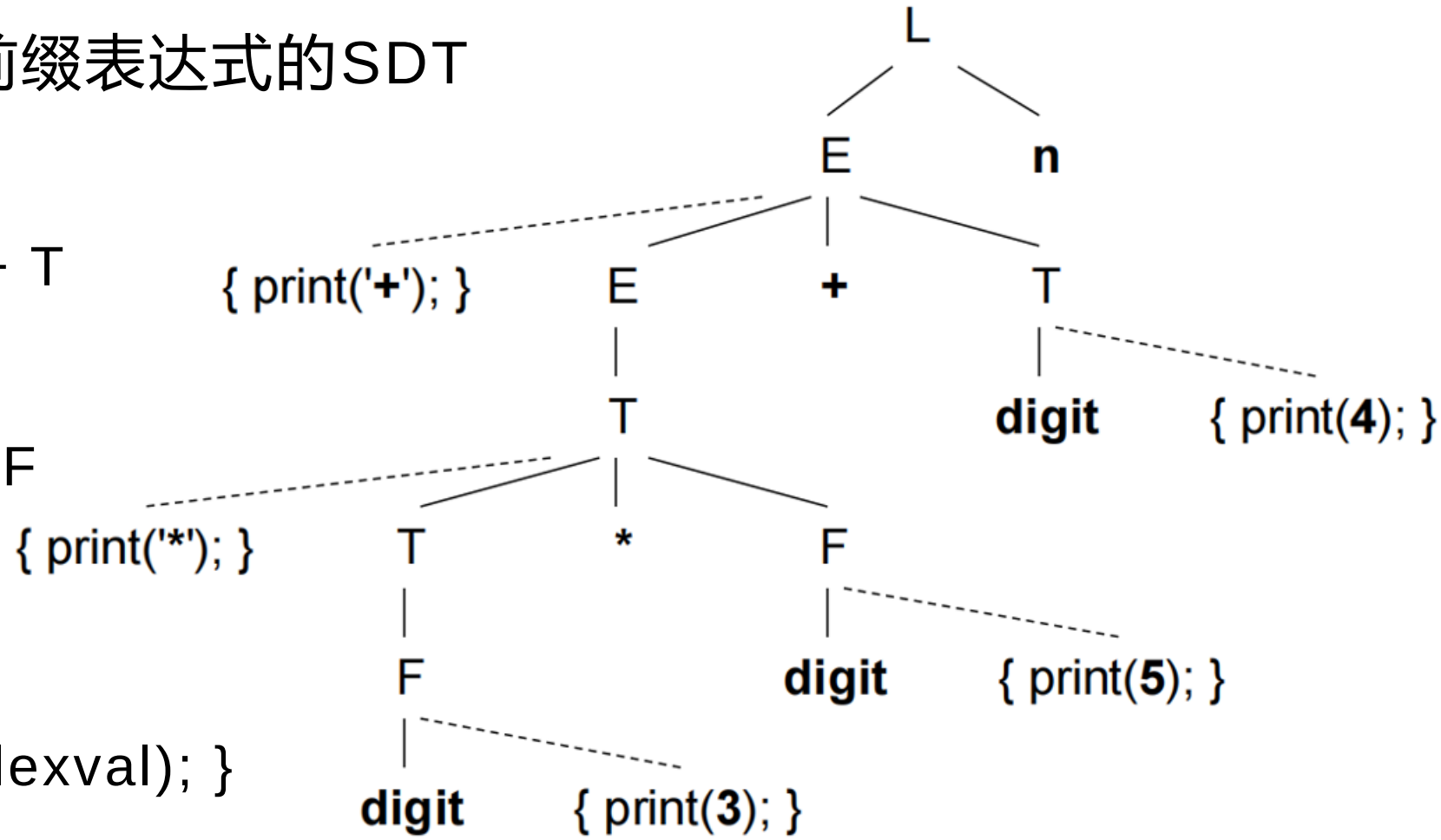
E → T

T → **{ print('*'); }** T1 * F

T → F

F → (E)

F → **digit** { print(**digit**.lexval); }

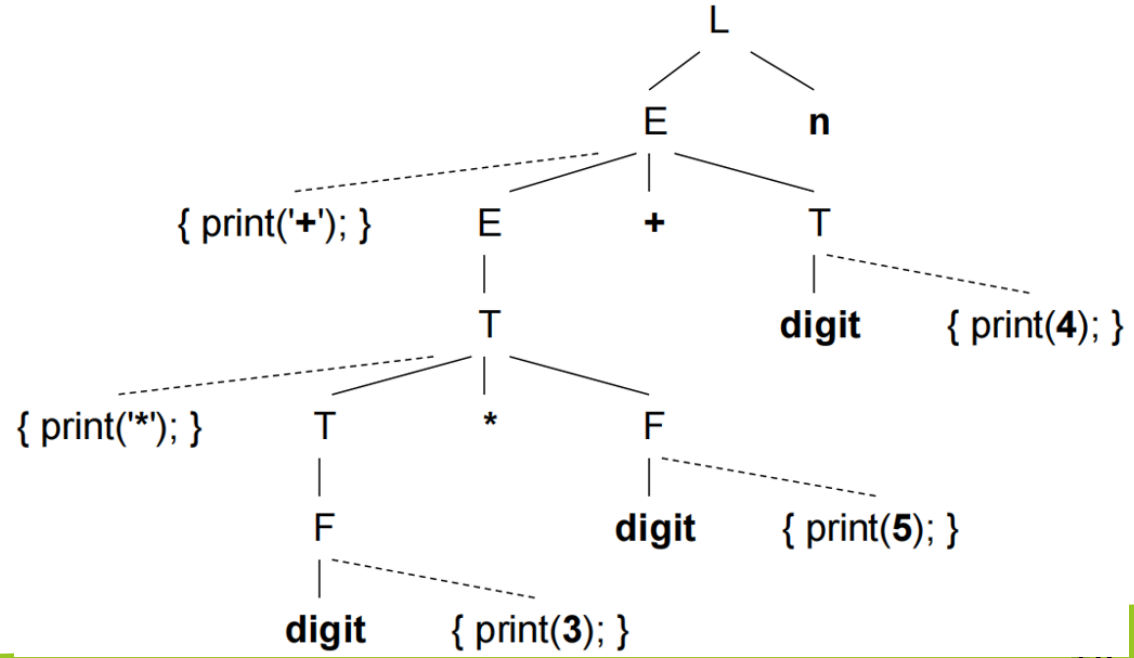


将中缀表达式3*5+4，输出成前缀表达式+*354

3. L-属性翻译方案

- 注意：**不是所有的SDT都可以在语法分析过程中实现**
- 例：将中缀表达式转成前缀表达式的SDT
  - 该文法无法在语法分析过程中实现
  - 存在问题：
    - ✓ **动作时机过早**：位于产生式最开始，语法分析器必须在看到整个表达式之前就决定打印哪个运算符
    - ✓ 自底向上：在预测使用哪个产生式之前就需要执行打印动作
    - ✓ 自顶向下：在知道运算符是什么之前就需要执行打印动作

L → E **n**
E → { print('+'); } E1 + T
E → T
T → { print('*'); } T1 * F
T → F
F → (E)
F → **digit** { print(**digit**.lexval); }



4. L-属性的SDD转换为SDT

- 将一个L-属性的SDD转换为一个SDT的规则：
  - 把计算某个**非终结符A的继承属性的动作**插入到产生式体中紧靠在**A**的本次**出现之前**的位置上；如果A的多个继承属性以无环的方式相互依赖，就需要对这些属性的求值动作进行排序，以便先计算需要的属性
  - 将计算一个**产生式头的综合属性的动作**放置在这个**产生式体的最右端**

4. L-属性的SDD转换为SDT

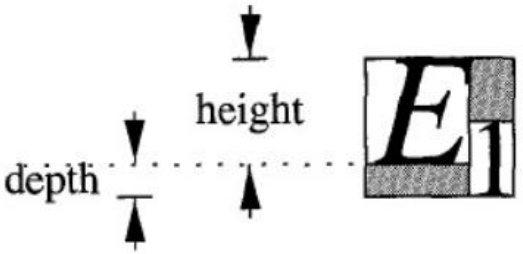
- 例：数学公式排版文法 $B \rightarrow B_1 B_2 | B_1 \textbf{sub} B_2 | (B_1) | \textbf{text}$　　B代表box
  - 两个并排的box： 
  - 两个box，一个是另一个的下标： 
  - 一个用括号括起来的box：()
  - 一串普通的文本串：**max**(...)
  - 约定：
    - **sub下标是右结合**
    - **并排关系是右结合**
    - **sub的优先级高于并排关系**

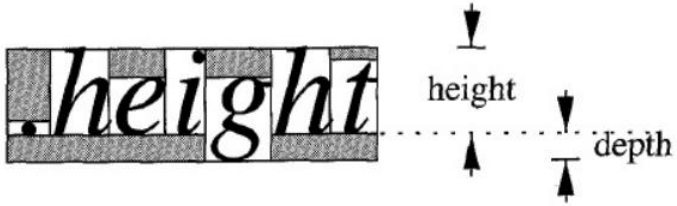
4. L-属性的SDD转换为SDT

- 例：数学公式排版文法 $B \rightarrow B_1 B_2 | B_1 \textbf{sub} B_2 | (B_1) | \textbf{text}$

Productions	Semantic Actions
1) $S \rightarrow B$	$B.ps = 10$
2) $B \rightarrow B_1 B_2$	$B_1.ps = B.ps$ $B_2.ps = B.ps$ $B.ht = \max(B_1.ht, B_2.ht)$ $B.dp = \max(B_1.dp, B_2.dp)$
3) $B \rightarrow B_1 \textbf{sub} B_2$	$B_1.ps = B.ps$ $B_2.ps = B.ps \times 70\%$ $B.ht = \max(B_1.ht, B_2.ht - B.ps \times 25\%)$ $B.dp = \max(B_1.dp, B_2.dp + B.ps \times 25\%)$
4) $B \rightarrow (B_1)$	$B_1.ps = B.ps$ $B.ht = B_1.ht$ $B.dp = B_1.dp$
5) $B \rightarrow \textbf{text}$	$B.ht = getHight(B.ps, \textbf{text}.lexval)$ $B.dp = getDepth(B.ps, \textbf{text}.lexval)$

ps = point size
ht = height
dp = depth





4. L-属性的SDD转换为SDT

- 例：数学公式排版文法 $B \rightarrow B_1 B_2 | B_1 \textbf{sub} B_2 | (B_1) | \textbf{text}$

S →		{ B.ps = 10; }
	B	
B →		{ B_1.ps = B.ps; }
	B_1	{ B_2.ps = B.ps; }
	B_2	{ B.ht = max(B_1.ht, B_2.ht);
		B.dp = max(B_1.dp, B_2.dp); }
B →		{ B_1.ps = B.ps; }
	B_1 **sub**	{ B_2.ps = B.ps × 70%; }
	B_2	{ B.ht = max(B_1.ht, B_2.ht – B.ps × 25%);
		B.dp = max(B_1.dp, B_2.dp + B.ps × 25%); }
B →	**(**	{ B_1.ps = B.ps; }
	B_1 **)**	{ B.ht = B_1.ht;
		B.dp = B_1.dp; }
B →	**text**	{ B.ht = getHight(B.ps, **text**.lexval);
		B.dp = getDepth(B.ps, **text**.lexval); }

- 把计算某个非终结符A的继承属性的动作插入到产生式体中紧靠在A的本次出现之前的位置
- 将计算一个产生式头的综合属性的动作放置在这个产生式体的最右端

5. 预测分析中的L-属性定义

- 进行递归下降预测翻译的步骤
  1）为语法规则编写一个LL(1)语法
  2）通过附加语义规则定义L-属性定义
  3）将L-属性定义转换为翻译方案
  4）消除翻译方案中的左递归（因为**带有左递归的文法无法进行确定的自顶向下语法分析**）
  5）编写一个递归下降预测解析器（翻译器）

5. 预测分析中的L-属性定义

4）消除翻译方案中的左递归

① 最简单的情况：SDD中的语义动作只涉及打印输出，而不涉及计算

– 此时，可将该动作当成终结符处理，然后使用消除左递归的通用方法

– 例：

$E \rightarrow E1 + T$ **{ print('+'); }**

$E \rightarrow T$

$E \rightarrow T\ R$

$R \rightarrow + T$ **{ print('+'); }** R

$R \rightarrow \varepsilon$

✓ **引入新的非终结符A’**，将产生式**A**→**A**α|β改写成：A→βA'，A'→αA'|ε

5. 预测分析中的L-属性定义

4）消除翻译方案中的左递归

② 若SDD中的语义动作涉及计算，且SDD是S-属性的

– 此时，可以通过将计算属性值的动作放在新产生式中的适当位置上来构造出一个SDT

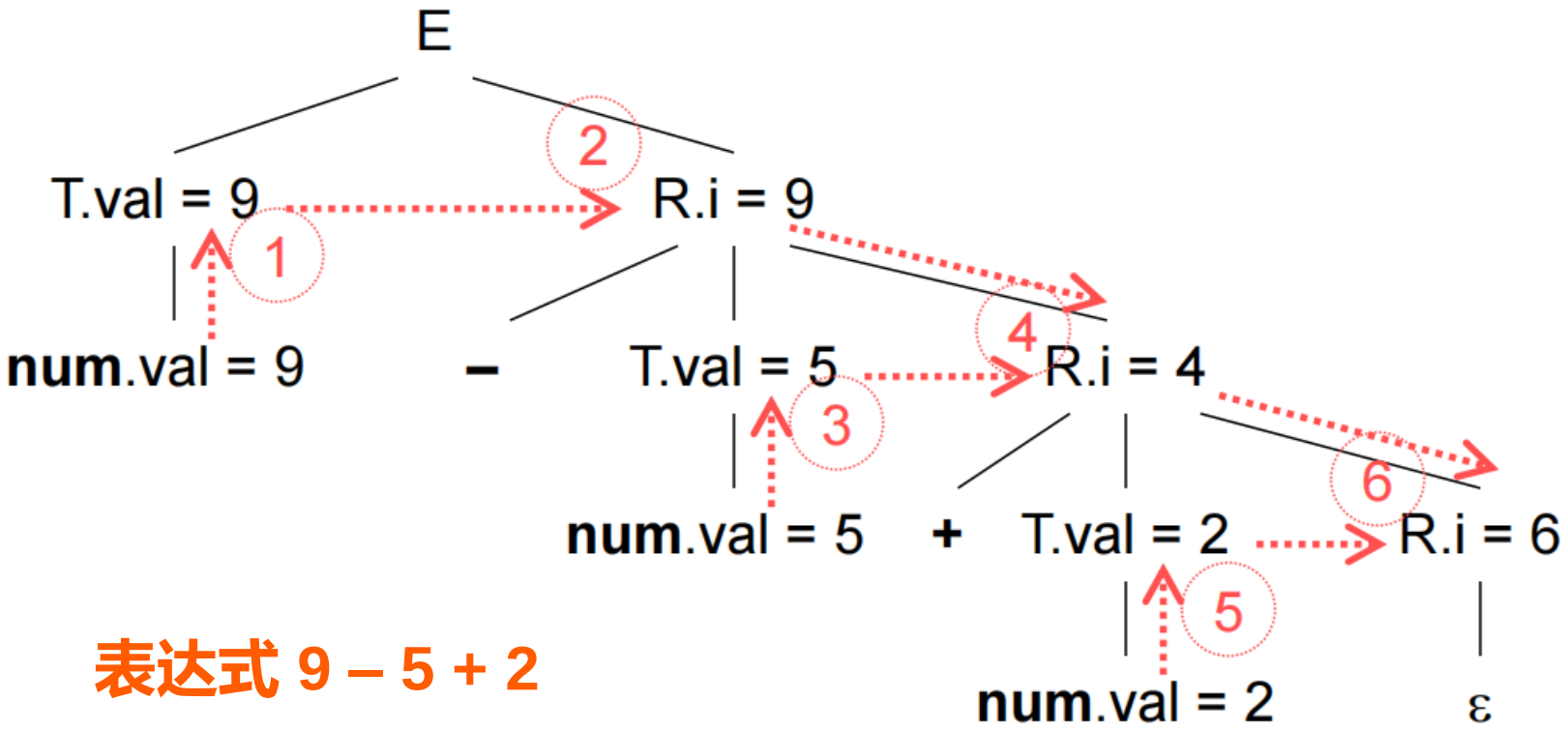
E → E1 **+** T **{ E.val = E1 .val + T.val; }**
E → E1 **–** T **{ E.val = E1 .val – T.val; }**
E → T **{ E.val = T.val; }**
T → **(** E **)** **{ T.val = E.val; }**
T → **num** **{ T.val = num.val; }**

E → T { R.i = T.val; }
R { E.val = R.s; }
R → **+** T { R1.i = R.i + T.val; }
R1 { R.s = R1.s; }
R → **–** T { R1.i = R.i – T.val; }
R1 { R.s = R1.s; }
R → ε { R.s = R.i; }
T → **(** E **)** { T.val = E.val; }
T → **num** { T.val = **num**.val; }

5. 预测分析中的L-属性定义

E → E1 + T { E.val = E1 .val + T.val; }
E → E1 – T { E.val = E1 .val – T.val; }
E → T { E.val = T.val; }
T → (E) { T.val = E.val; }
T → num { T.val = num.val; }

E → T { R.i = T.val; } R { E.val = R.s; }
R → + T { R1.i = R.i + T.val; } R1 { R.s = R1.s; }
R → – T { R1.i = R.i – T.val; } R1 { R.s = R1.s; }
R → ε { R.s = R.i; }
T → (E) { T.val = E.val; }
T → num { T.val = num.val; }



表达式 9 – 5 + 2

解析9-5+2：

1. T → **num**: T.val = **num**.val=9;
2. E → T R: R.i = T.val=9;
3. T → **num**: T.val = **num**.val=5;
4. R → – T R1: R1.i = R.i – T.val =9-5=4;
5. T → **num**: T.val = **num**.val=2;
6. R → + T R1: R1.i = R.i + T.val =4+2=6;
7. R → R.s = R.i=6;
8. E.val = R.s=6;

5. 预测分析中的L-属性定义

4）消除翻译方案中的左递归

② 若SDD中的语义动作涉及计算，且SDD是S-属性的

– 此时，可以通过将计算属性值的动作放在新产生式中的适当位置上来构造出一个SDT

– 通用的解决方案

A → A1 Y **{ A.a = g(A1.a, Y.y); }**

A → X **{ A.a = f(X.x); }**

A.a，X.x和Y.y均为综合属性

A → X **{ R.i = f(X.x); }** R **{ A.a = R.s; }**

R → Y **{ R1.i = g(R.i, Y.y); }** R1 **{ R.s = R1.s; }**

R → ε **{ R.s = R.i; }**

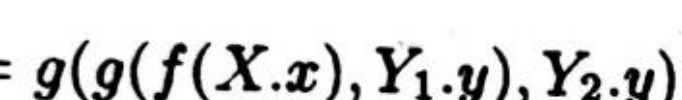
R.i为继承属性，R.s为综合属性

5. 预测分析中的L-属性定义

4）消除翻译方案中的左递归

A → A1 Y { A.a = g(A1.a, Y.y); }

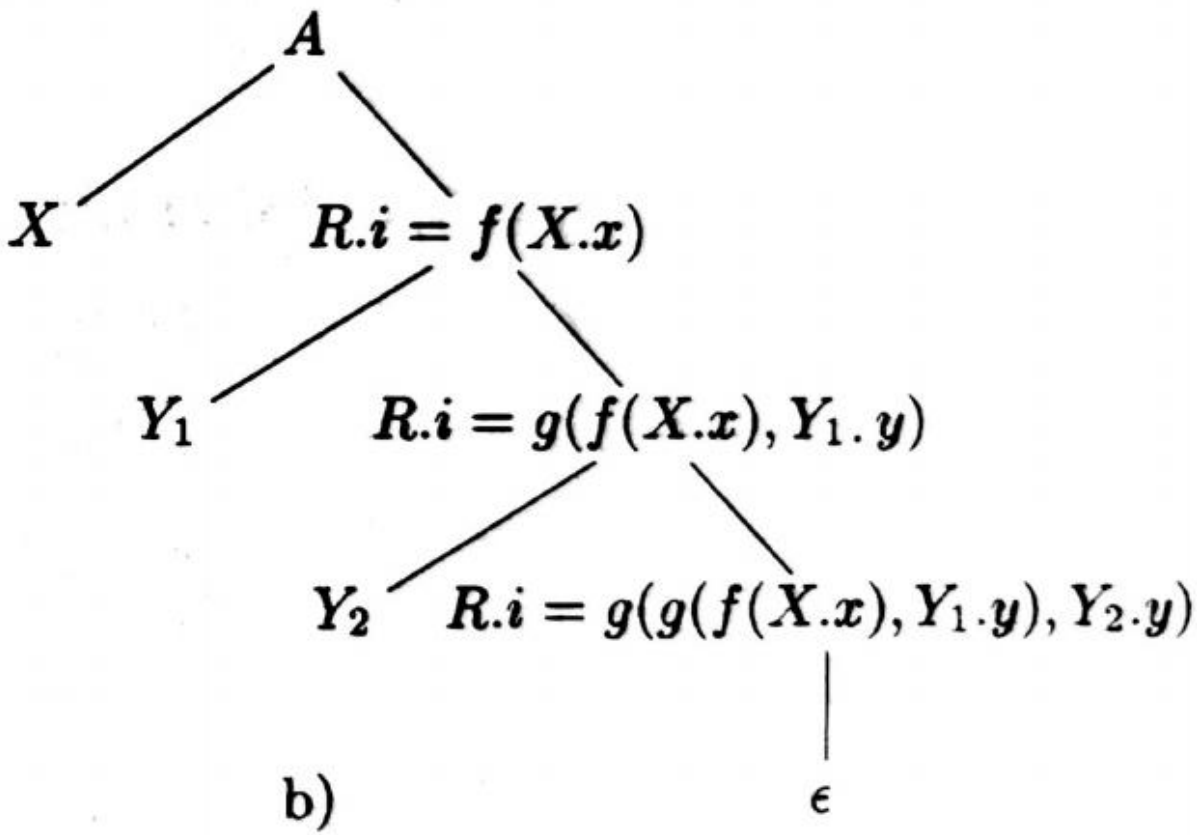
A → X { A.a = f(X.x); }



A → X { R.i = f(X.x); } R { A.a = R.s; }

R → Y { R1.i = g(R.i, Y.y); } R1 { R.s = R1.s; }

R → ε { R.s = R.i; }



5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 例： R → **addop** T { R1 .i = mknode(**addop**.lexeme, R.i, T.nptr); }

R1 { R.s = R1 .s; }

R → ε { R.s = R.i; }

– 只解析：

```
void R() {
    if (lookahead == addop) {
        match(addop);
        T();
        R()
    } else {
        // do nothing
    }
}
```

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 解析并翻译：

- ✓ 每个继承属性对应于一个形式参数
- ✓ 所有综合属性对应返回值
  - 多条综合属性可以合并到一条记录返回中
- ✓ 子节点的每个属性对应一个局部变量
- ✓ 对于产生式右部：
  - 若为终结符：match()
  - 若为非终结符：过程调用
  - 若为动作[actions]：直接执行

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 解析并翻译：

```
SyntaxTreeNode R(SyntaxTreeNode i) {
    SyntaxTreeNode s;  // synthesized attributes
    SyntaxTreeNode t_nptr, r1_i, r1_s;  // for children
    char addopLexeme;  // temporary

    if (lookahead == addop) {
        addopLexeme = lookahead.lexval;
        match(addop);
        t_nptr = T();
        r1_i = mknode(addopLexeme, i, t_nptr);
        r1_s = R(r1_i);
        s = r1_s;
    } else {
        s = i;
    }
    return s;
}
```

R → **addop** T { R1.i = mknode(**addop**.lexeme, R.i, T.nptr); }

R1 { R.s = R1.s; }

R → ε { R.s = R.i; }

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 例：定点二进制小数转换为十进制小数

$N \rightarrow .\ \{ S.f := 1 \}\ S\ \{ print(S.v) \}$

$S \rightarrow \{ B.f := S.f \}\ B\ \{ S_1.f := S.f + 1 \}\ S_1\ \{ S.v := S_1.v + B.v \}$

$S \rightarrow \varepsilon\ \{ S.v := 0 \}$

$B \rightarrow 0\ \{ B.v := 0 \}$

$B \rightarrow 1\ \{ B.v := 2^{-B.f} \}$

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 例：定点二进制小数转换为十进制小数

✓ 根据产生式N→ . { S.f : =1} S { print(S.v) }

对非终结符N，构造如下函数：

```
void ParseN()
{
    MatchToken('.');          //匹配'.'
    Sf :=1;                   //变量 Sf 对应属性S.f
    Sv := ParseS(Sf);         //变量 Sv 对应属性S.v
    print(Sv);
}
```

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 例：定点二进制小数转换为十进制小数

✓ 根据产生式$S\rightarrow$ { B.f : =S.f } B { S_1.f := S.f +1 } S1 {S.v := S_1.v+B.v }

$S\rightarrow\varepsilon$ { S.v := 0 }

对非终结符S，构造如下函数：

```
float ParseS(int f){
    if (lookahead=='0' or lookahead=='1'){
          Bf := f;   Bv := ParseB(Bf); S1f := f+1;
     S1v := ParseS(S1f); Sv := S1v + Bv;
    }
    else if (lookahead=='#')  Sv := 0;
    else { printf("syntax error \n"); exit(0); }
    return Sv;
}
```

5. 预测分析中的L-属性定义

5）编写一个递归下降预测解析器（翻译器）

– 例：定点二进制小数转换为十进制小数

✓ 根据产生式$B \to 0$ { B.v := 0 }

$B \to 1$ { $B.v := 2^{-B.f}$ }

对非终结符B，构造如下函数：

```
float ParseB(int f){
    if (lookahead=='0'){ MatchToken('0'); Bv := 0 }
    else if(lookahead=='1'){
        MatchToken('1');    Bv := 2^(-f)
    }
    else{ printf("syntax error \n"); exit(0); }
    return Bv;
}
```