编译原理

Compiler Construction Principles

朱 青

信息学院计算机系，

中国人民大学，

zqruc2012@aliyun.com

第5章 属性文法和语法制导翻译

- 5.1 语法制导翻译概说
- 5.2 基于属性文法的处理方法
- 5.3 S—属性文法的自下而上计算
- 5.4 L—属性文法的自顶向下翻译

中间代码生成是编译程序构造的第三阶段,普遍采用语法制导翻译方法,为每个产生式配一个翻译子程序(称语义动作或语义子程序),并且在语法分析的同时执行这些子程序.

语义动作指出:

(1)一个产生式所产生的符号的意义.

(2) 按照这种意义规定了生成某种中间代码应作哪些基本动作。

5.1 语法制导翻译概说

在语法分析过程中,随着分析的步步进展,根据每个产生式对应的语义程序(语义动作)进行翻译(产生中间代码)的办法叫做 语法制导翻译法.

有两种描述语法制导翻译的形式.一种称为语法制导定义,另一种称为翻译规程.

语法制导定义是关于翻译的高层次规格说明,其中隐去实现细节,不规定翻译顺序.

翻译规程规定翻译途径,指明语义动作的求值顺序.

语法制导定义

一个语法制导定义是一个上下文无关文法,其中每个文法符号都有一个相关的属性集合,属性分成两类,综合属性和继承属性.

属性:可以表示指定的任何信息.语法树结点中的属性要用语义规则来定义,语义规则和相应结点的产生式相关.

依赖关系图:按照语义规则可以建立各属性之间的依赖关系.这种依赖关系可以用一个有向图表示,称为依赖关系图.

例如:“算术表达式”E的“值”的语义(语法制导定义):

产生式	语义规则
(1)$E \longrightarrow E^{(1)} + E^{(2)}$	$\{E^{(1)}.VAL + E^{(2)}.VAL\}$
(2)$E \longrightarrow 0$	{E.VAL:=0 }
(3)$E \longrightarrow 1$	{E.VAL:=1 }

属性文法

- 属性(attribute)：
  - 编程语言结构的任意特性
  - 静态(static)：执行之前绑定的属性
  - 动态(dynamic)：执行期间绑定的属性
  - 属性的典型例子有：
    - 变量的数据类型：静态
    - 表达式的值：动态
    - 存储器中变量的位置：静态或动态
    - 程序的目标代码：静态

属性文法

- 属性文法：在上下文无关文法的基础上，为每个文法符号（终结符或非终结符）配备若干属性
  - 属性可以计算和传递
  - 属性加工过程即是语义处理过程
- 语义规则：文法的每个产生式都配备一组属性的计算规则

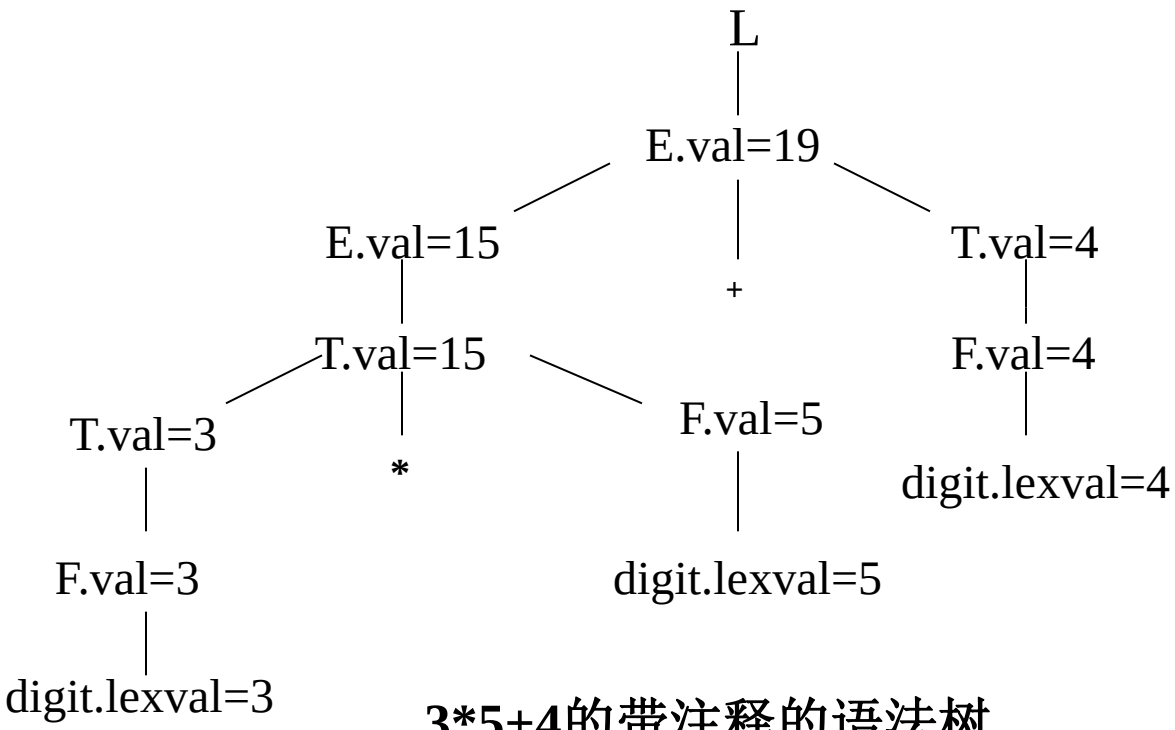
属性文法示例

台式计算器程序

产生式	语 义 规 则
$L \rightarrow E$	Print(E.val)
$E \rightarrow E_1+T$	$E.val:=E_1.val+T.val$
$E \rightarrow T$	E.val:=T.val
$T \rightarrow T_1 * F$	$T.val:=T_1.val \times F.val$
$T \rightarrow F$	T.val:=F.val
$F \rightarrow (E)$	F.val:=E.val
$F \rightarrow$ digit	F.val:=digit.lexval

综合属性

- 结点的综合属性由其子结点的属性值确定
- 通常采用自底向上的方法计算综合属性
- 例：设表达式为3＊5+4，则语义动作打印数值19



3*5+4的带注释的语法树

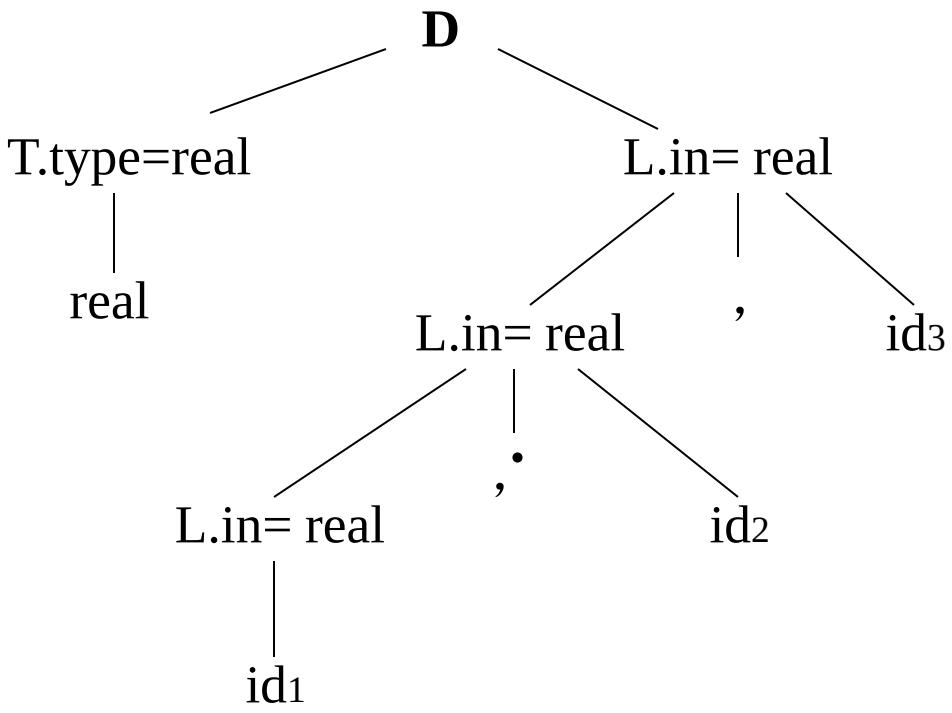
继承属性

- 结点的继承属性值是由此结点的父结点和/或兄弟结点的某些属性来决定的。
- 例2

产生式	语义规则
D →TL	L.in:=T.type
T→ int	T.type:=integer
T →real	T.type:=real
L → L1,id	L1.in:=L.in addtype(id.entry,L.in)
L→ id	addtype(id.entry,L.in)

继承属性

句子Real id1,id2,id3的语法树



第5章 属性文法和语法制导翻译

- 5.1 语法制导翻译概说
- 5.2 基于属性文法的处理方法
- 5.3 S—属性文法的自下而上计算
- 5.4 L—属性文法的自顶向下翻译

5.2 基于属性文法的处理方法

- 基于属性文法的处理过程：即语法制导翻译
  - 通常为：

    输入串→ 语法树 →属性 依赖图 → 语义规则计算顺序
  - 具体实现有时也可用一遍扫描实现属性文法的语义规则计算

依赖图

- 依赖图：
  - 有向图
  - 描述语法分析树中的继承属性和综合属性之间的相互依赖关系。

- 依赖图的构造算法：

for分析树中每一个结点n　do

for 结点n的文法符号的每一个属性a　do

为a在依赖图中建立一个结点；

for 分析树中每一个结点n　do

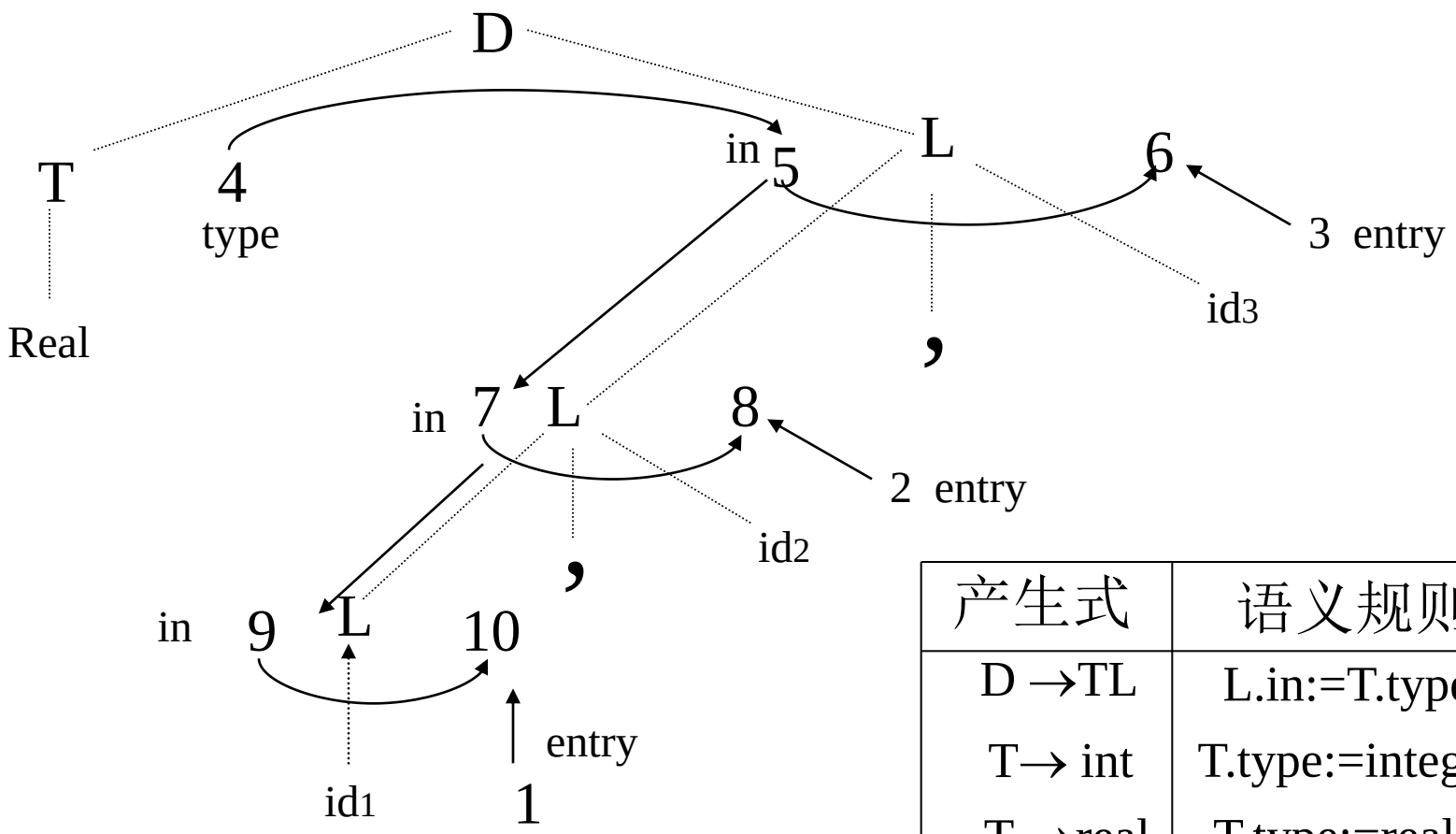
for结点n所用产生式对应的每一个语义规则

b:=f(c_1,c_2,…c_k)　　do

for　i :=1 to k do

从c_i结点到b结点构造一条有向边

例 Real id1,id2,id3分析树的依赖图(图中的结点用数字表示)



产生式	语义规则
D →TL	L.in:=T.type
T→ int	T.type:=integer
T →real	T.type:=real
L → L1,id	L1.in:=L.in addtype(id.entry,L.in)
L→ id	addtype(id.entry,L.in)

良定义的属性文法

- **良定义**：如果一属性文法不存在属性之间的循环依赖关系，那么称该文法为良定义的。
  - 一个属性对另一个属性的循环依赖关系。如，p、c_1、c_2都是属性，若p: = $f_1(c_1)$、c_1: = $f_2(c_2)$、c_2: = $f_3(p)$时，就无法对p求值。
- 为了设计编译程序，我们只处理良定义的属性文法。

抽象语法树

- 抽象语法树：去除语法树中对翻译不必要的信息
- 操作符和关键字不作为叶结点出现
- 建立表达式的抽象语法树
  - mknode(op,left,right)：建立运算符号结点
  - mkleaf(id,entry)：建立标识符结点
  - mkleaf(num,ral)：建立数结点

例：a-4+c抽象语语法树的建立

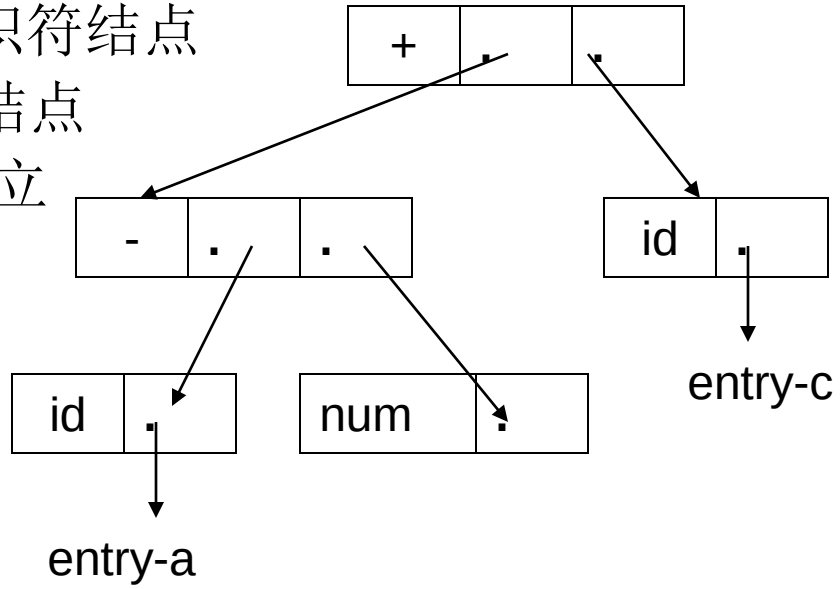
p1:=mkleaf(id,entry-a);

P2:=mkleaf(num,4);

P3:=mknode(‘-’,p1,p2);

P4:=mkleaf(id,entry-c);

P5:=mknode(‘+’,p3,p4);



第5章 属性文法和语法制导翻译

- 5.1 语法制导翻译概说
- 5.2 基于属性文法的处理方法
- 5.3 S—属性文法的自下而上计算
- 5.4 L—属性文法的自顶向下翻译

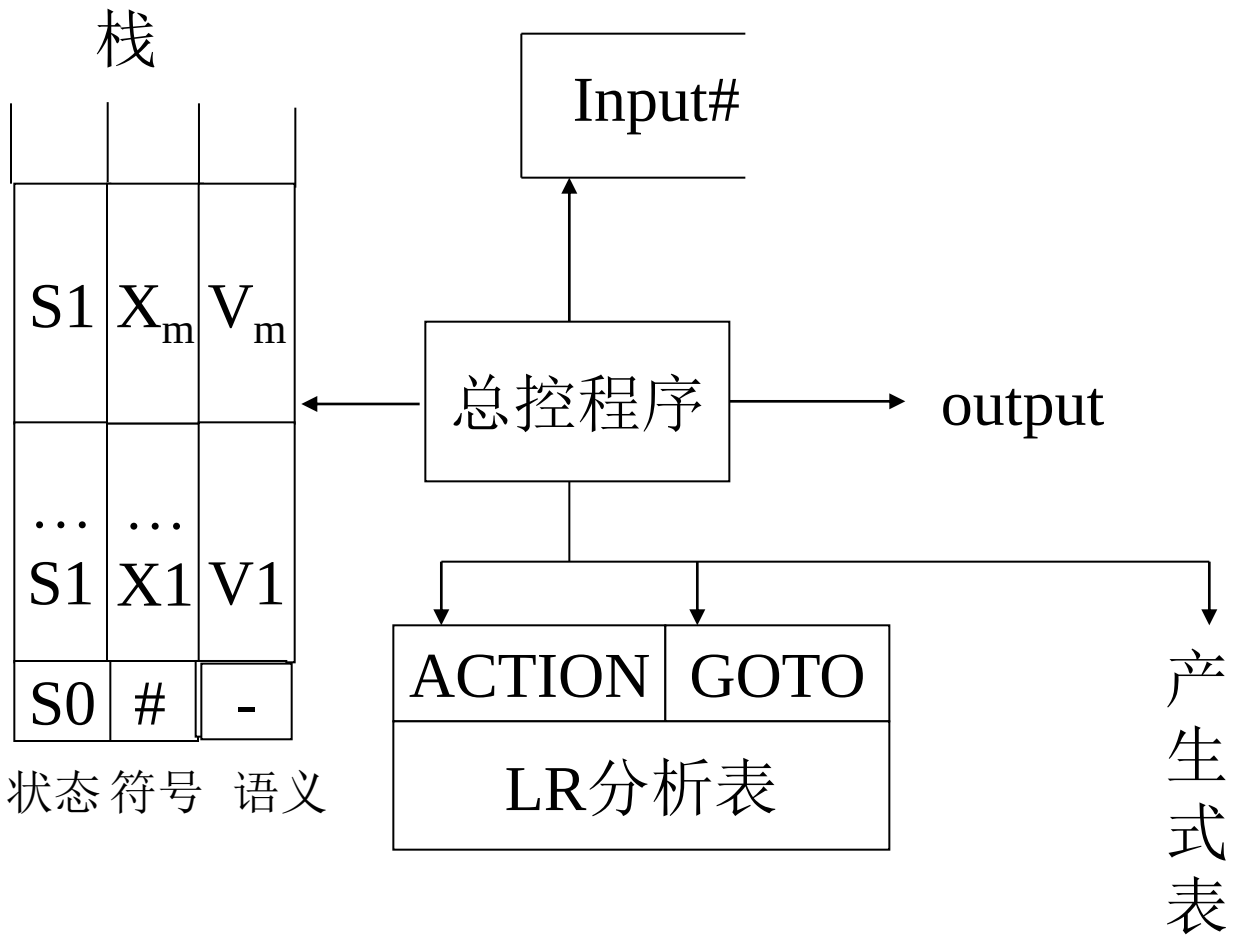
5.3 S－属性文法的自下而上计算

- 特点：只含有综合属性
- 计算：可以在分析输入符号串的同时自下而上的计算
  - 分析器可以保存与栈中文法符号有关的综合属性值，每当进行归约时，新的属性值就由栈中正在归约的产生式右边符号的属性值来计算。
- 可借助于LR分析器实现
  - 在S－属性文法的基础上，LR分析器可以改造为一个翻译器，在对输入串进行语法分析的同时对属性进行计算

S-属性文法

产生式	语义规则
0）L→E	print（E.val）
1）E→E¹+T	E.val：=E¹.val+T.val
2）E→T	E.val：=T.val
3）T→T¹*F	T.val：=T¹.val×F.val
4）T→F	T.val：=F.val
5）F→（E）	F.val：=E.val
6）F→digit	F.val:=digit.lexval

LR 分析器模型


栈
Input#
S1 X_m V_m
… …
S1 X1 V1
S0 # -
状态 符号 语义
总控程序
output
ACTION GOTO
LR分析表
产生式表

2＋3*5 的分析和计值过程

步骤	动作	状态栈	语义栈(值栈)	符号栈	余留输入串
1)		0	—	#	2＋3*5#
2)		0 5	—2	#	＋3*5#
3)	r6	0 3	—2	#F	＋3*5#
4)	r4	0 2	—2	#T	＋3*5#
5)	r2	0 1	—2	#E	＋3*5#
6)		0 1 6	—2—	#E＋	3*5#
7)		0 1 6 5	—2——	#E＋3	*5#
8)	r6	0 1 6 3	—2—3	#E＋F	*5#
9)	r4	0 1 6 9	—2—3	#E＋T	*5#
10)		0 1 6 9 7	—2—3—	#E＋T*	5#
11)		0 1 6 9 7 5	—2—3——	#E＋T*5	#
12)	r6	0 1 6 9 7 (10)	—2—3—5	#E＋T*F	#
13)	r3	0 1 6 9	—2—(15)	#E＋T	#
14)	r1	0 1	—(17)	#E	#
15) 接受					

第5章 属性文法和语法制导翻译

- 5.1 语法制导翻译概说
- 5.2 基于属性文法的处理方法
- 5.3 S—属性文法的自下而上计算
- 5.4 L—属性文法的自顶向下翻译

5.4 L-属性文法和自顶向下翻译

- 一个属性文法称为L－属性文法，如果对于每个产生式$A \rightarrow X_1X_2...X_n$，其每个语义规则中的每个属性或者是综合属性，或者是X_j（$1 \leqslant j \leqslant n$）的一个继承属性且这个继承属性仅依赖于：

（1）产生式X_j在左边符号X_1，X_2，…，X_{j-1}的属性；

（2）A的继承属性。

- L－属性文法允许一次遍历就计算出所有属性值。
- LL（1）：可以在自上而下语法分析的同时实现L属性文法的计算。

翻译模式

- 翻译模式：用于语法制导翻译的一种描述形式
  - 语义动作用花括号{ }括起来，插入到产生式右部的合适位置上（指示使用语义规则的计算次序）
  - 例如： 中缀表达式翻译为后缀表达式
    - LL分析:

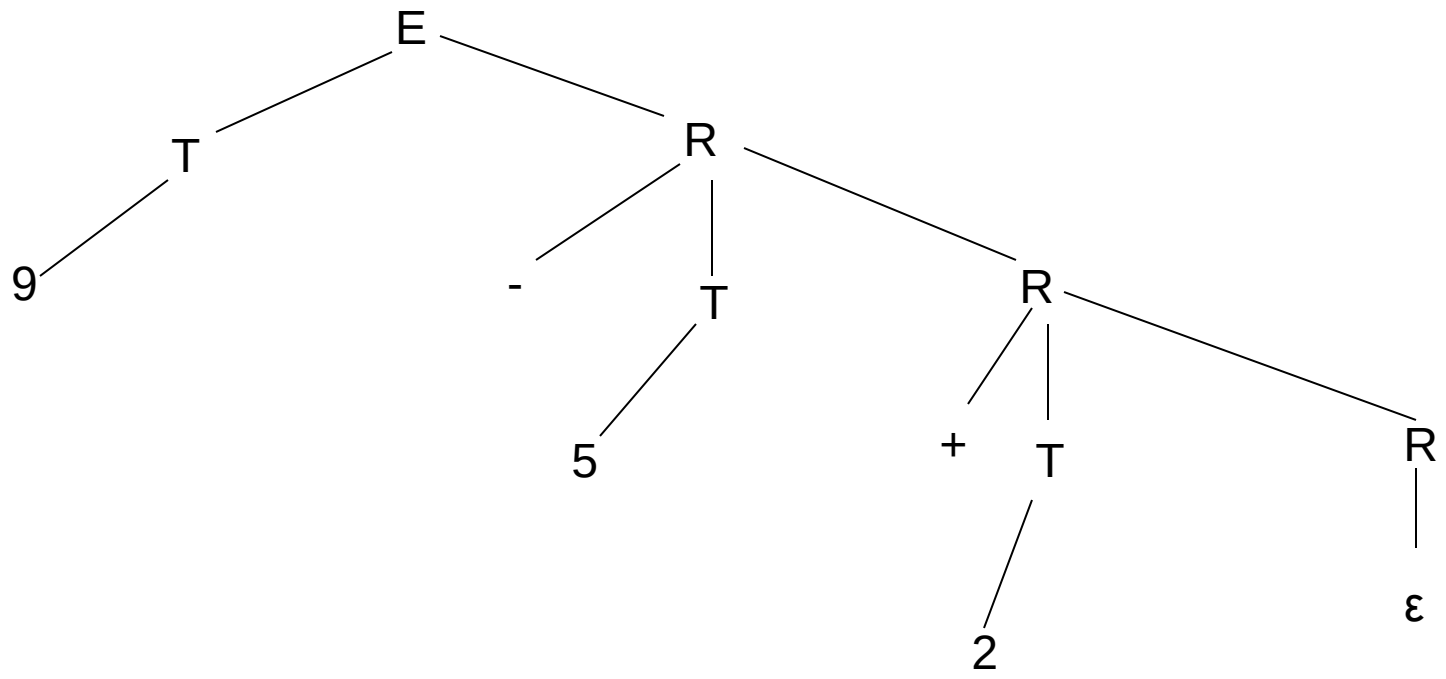
      E→TR

      R→addop T {print(addop. Lexeme)} R_1|ε

      T→num {print(num.val)}

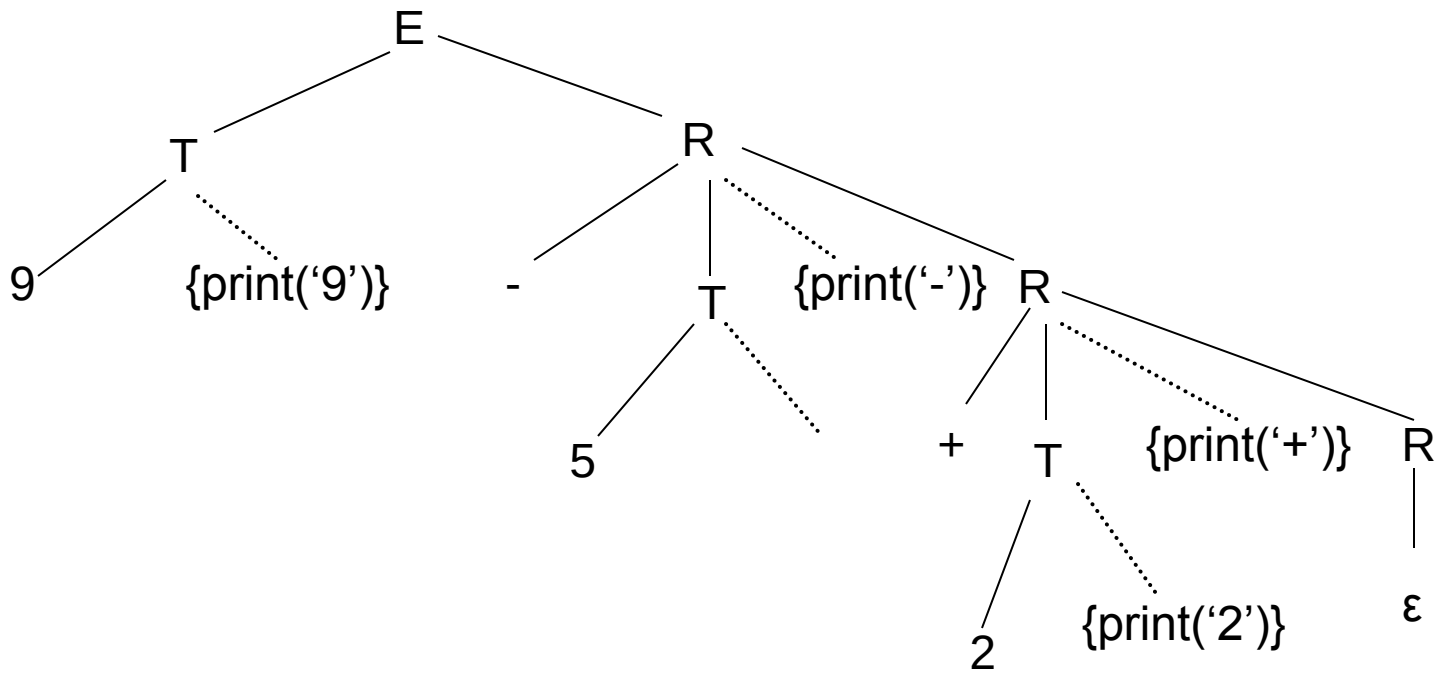
翻译模式

输入串9－5 + 2的LL语法分析树



翻译模式

把语义动作看作是终结符,每个语义动作作为相应产生式左部符号的结点的儿子



按深度优先次序执行图中的动作后，打印输出95－2+。

翻译模式

- 将L-属性文法改写为翻译模式的规则
  - 产生式右部的符号的继承属性必须在这个符号以前的动作中计算出来
  - 一个动作不能引用这个动作右部的符号的综合属性
  - 产生式左部非终结符的综合属性只有在它所引用的所有属性都计算出来以后才能计算，计算这种属性的动作通常放在产生式右端的末尾

S属性文法的翻译模式

例：对二目运算符的运算对象进行类型匹配审查

G[E]:

(1) E →T+T
(2) E →T or T
(3) T →n
(4) T →b

E→T^1 + T^2
{ if T^1.type = int and T^2.type= int
then E.type :=int
else error}

E →T^1 or T^2
{ if T^1.type = bool and T^2.type= bool
then E.type :=bool
else error}

T → n { T.type := int}

T → b { T.type := bool}

L－属性文法在自顶向下分析中的实现

- 带左递归的文法的翻译模式

$E \rightarrow E_1 + T$ {E.val: = E_1.val + T.val}

$E \rightarrow E_1 - T$ {E.val: = E_1.val－T.val}

$E \rightarrow T$ {E.val: = T.val}

$T \rightarrow (E)$ {T.val: = E.val}

$T \rightarrow num$ {T.val: = num.val}

L－属性文法在自顶向下分析中的实现

- 消除左递归的同时考虑属性，构造新的**翻译模式**

$E \rightarrow T$ {R.i: =T.val}

R {E.val: = R.s}

$R \rightarrow +$

T {R_1.i:=R.i + T.val}

R_1 {R.s: = R_1.s}

$R \rightarrow -$

T {R_1.i: = R.i -T.val}

R_1 {R.s: = R_1.s}

$R \rightarrow \varepsilon$ {R.s: = R.i}

$T \rightarrow (E)$ {T.val: =E.val}

$T \rightarrow num$ {T.val: = num.val}