第五章　语法制导的翻译

戴新宇

2021-11


字符流
词法分析器
符号流
语法分析
语法树
语义分析
语法树
符号表
中间代码生成器
中间表示形式
机器无关代码优化器
中间表示形式
代码生成器
目标机器语言
机器相关代码优化器
目标机器语言

问题：

- 语义分析包含哪些部分？
- 上下文无法能否完成？
  - 上下文无关文法不能表达标示符在程序中先声明后使用的现象
- 如何解决？

语法制导的翻译

- 使用上下文无关文法来引导对语言的“翻译”
- 可以用于类型检查和中间代码生成等任务
- 在产生式中引入：
  - 属性：附加在代表语言构造的文法符号上，把若干信息与语言构造联系起来。
    - 可以是多种类型的，比如数字、串、记录等
  - 语义规则：附加在文法产生式上，用来描述文法符号的属性值。
    - 能够产生中间代码
    - 能够将词法信息填入符号表
    - 能够进行类型检查
    - ……
- E.g.

产生式	语义规则
$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel '+'$

语法制导的翻译

- 对于语义规则和产生式的关联，有两种方法
  - 语法制导定义

产生式	语义规则
$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel '+'$

  - 翻译方案

$$E \rightarrow E_1 + T \quad \{ \text{print } '+' \}$$

$$L \rightarrow E\ \mathbf{n} \qquad \{ \text{print}(E.val); \}$$

  - 语法制导定义更易读。翻译方案更加高效，更适合翻译的实现。

语法制导定义

- Syntax-Directed Definition, SDD
  - 上下文无关文法和属性及规则的结合
  - 属性和文法符号相关联
  - 语义规则和产生式相关联
- 属性分类
  - 综合属性
  - 继承属性

1、 综合属性。在分析树结点 N 上的非终结符号 A 的综合属性是由 N 上的产生式所关联的语义规则来定义的。请注意，这个产生式的头一定是 A。结点 N 上的一个综合属性只能通过 N 的子结点或 N 本身的属性值来定义。

2、 继承属性。在分析树结点 N 上的一个非终结符号 B 的一个继承属性是由 N 的父结点上的产生式所关联的语义规则来定义的。请注意，这个产生式的体中必然包含符号 B。结点 N 上的继承属性只能通过 N 的父结点、N 本身、和 N 的兄弟结点上的属性值来定义。

- 在一个SDD中，一个产生式$A \rightarrow \alpha$和一组语义规则相关联：$b=f(c_1,c_2,\ldots,c_n)$ ，$b,c_1,c_2,\ldots,c_n$均为A或α中文法符号的属性值
  - 若b是A的属性，且$c_1,c_2,\ldots,c_n$是A或α中文法符号的属性，则b称为A的综合属性
  - 若b是α中某个文法符号X的属性，且$c_1,c_2,\ldots,c_n$是A或α中文法符号的属性，则b称为X的继承属性
- 书上的定义基于分析树，且针对非终结符号
- 终结符号可以具有综合属性，其值来源于词法分析器。不能有继承属性。

一个SDD例

- 每个非终结符号具有唯一的名为val的综合属性。
- 终结符号digit的综合属性lexval由词法分析器提供

	产生式	语义规则
1)	$L \to E$ **n**	$L.val = E.val$
2)	$E \to E_1 + T$	$E.val = E_1.val + T.val$
3)	$E \to T$	$E.val = T.val$
4)	$T \to T_1 * F$	$T.val = T_1.val \times F.val$
5)	$T \to F$	$T.val = F.val$
6)	$F \to (E)$	$F.val = E.val$
7)	$F \to$ **digit**	$F.val =$ **digit**.lexval

图 5-1　一个简单的桌上计算器的语法制导定义

S属性的SDD

- 只包含综合属性的SDD称为S属性的SDD
- 在S属性的SDD中，每个规则都根据相应产生式体中的属性值来计算产生式头部非终结符号的属性值。

属性求值和注释语法分析树

- 基于语法分析树，可以通过语义规则对语法分析树上的各个结点的所有属性进行求值。
- 显示了它的各个属性的值的语法分析树称为注释语法分析树。
- 树上那么多的结点，按照什么顺序对各个节点的属性进行求值呢？
- 显然，在对某个节点的一个属性进行求值之前，必须首先求出这个属性值所依赖的所有属性值。
- 一个产生式$A \rightarrow \alpha$的一条语义规则$b=f(c_1,c_2,\ldots,c_n)$，要求b的值，首先得计算出$c_1,c_2,\ldots,c_n$的值，计算顺序反应它们之间存在依赖关系。

属性求值和注释语法分析树（续）

- 所有属性都是综合属性，比较好办，自底向上，后根遍历即可。

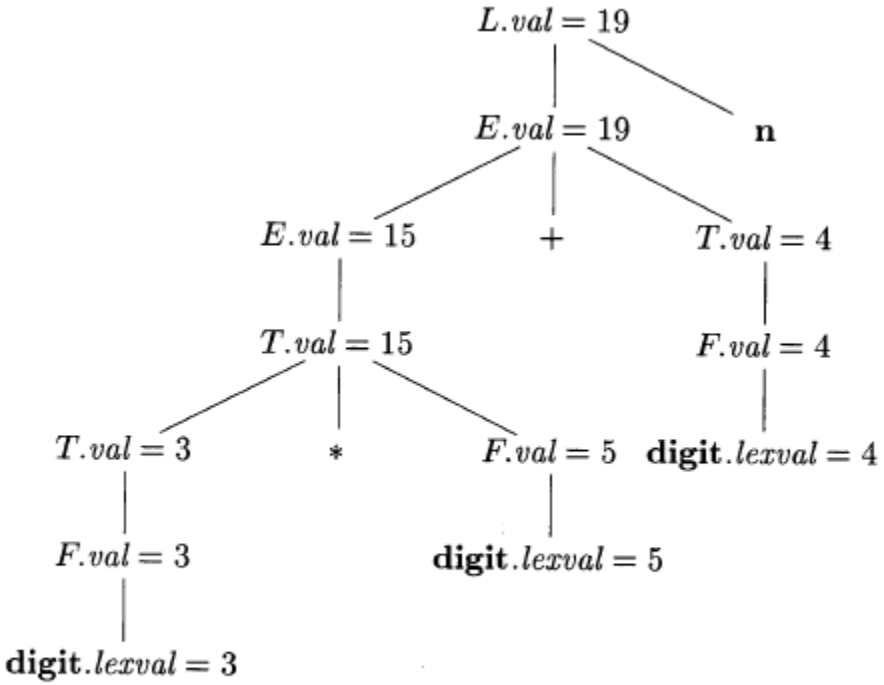


图 5-3　3 * 5 + 4**n** 的注释语法分析树

属性求值和注释语法分析树（续）

- 对存在继承属性的语法分析树进行注释
- 继承属性有时是必要的

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) $F \to \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

图 5-4　一个基于适用于自顶向下语法分析的文法的 SDD

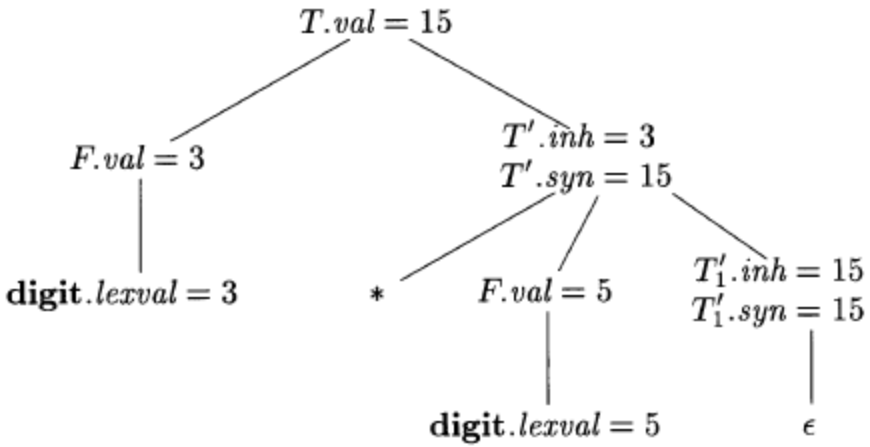


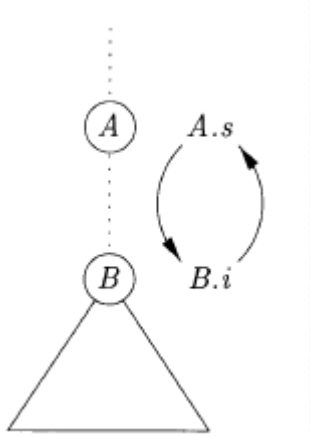
图 5-5　3 * 5 的注释语法分析树

- 如何给出正确的求值顺序？
  - 依赖图

属性求值和注释语法分析树（续）

- 对于同时具有综合属性和继承属性的SDD，不能保证有一个顺序来对各节点上的属性进行求值，可能会有循环定义。

产生式	语义规则
$A \rightarrow B$	$A.s = B.i;$ $B.i = A.s + 1$



- 有一类SDD，能够保证对每棵语法树都存在一个求值顺序。
  - S属性SDD
  - L属性SDD

依赖图

- 可以确定一棵给定语法分析树中各个属性求值顺序的一种有效工具。
- 依赖图
  - 结点：对应于语法分析树上的每个结点，其上的文法符号的每个属性，都是依赖图中的一个结点。
  - 边：在一个产生式对应的语义规则中，一个文法符号X的属性a的值的计算需要另一个文法符号B的属性b的值，那么在依赖图中有一条b指向a的有向边。
- 显然一个SDD中的属性和语义规则可以帮助我们构造其对应的依赖图

依赖图示例一

产生式	语义规则
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$

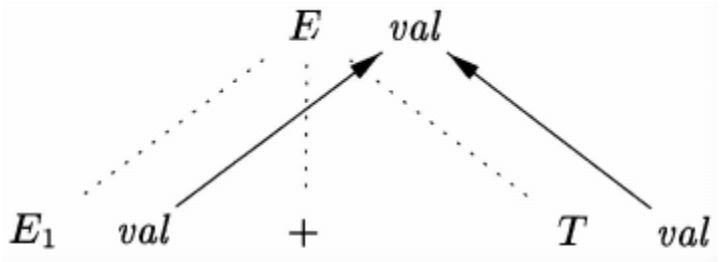


图 5-6　$E.val$ 由 $E_1.val$ 和 $T.val$ 综合得到

依赖图示例二

- 一个完整的依赖图

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) $F \to \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

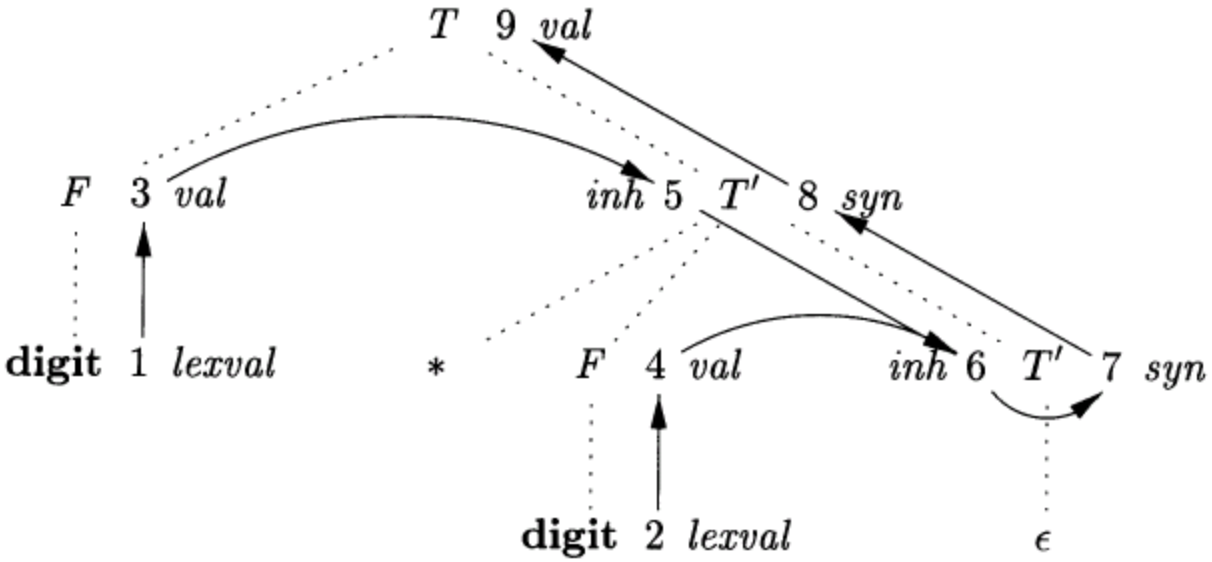


图 5-7　对应于图 5-5 中的注释语法分析树的依赖图

属性求值的顺序

- 依赖图指定了各个属性的计算顺序
- 依赖图中有一条从结点M到结点N的边，那么就先要对M对应的属性求值，才能对N对应的属性求值
- 拓扑排序：一种可行的依赖图中属性的求值顺序。给依赖图上的所有结点排序N_1、N_2、...、N_k，如果有一条从结点N_i到N_j的边，那么i<j
- 如果图中存在环，则不存在拓扑排序。如果图中没有环，则至少存在一个拓扑排序。

S属性定义和L属性定义

- 在一个给定的SDD中，很难判定是否存在一棵其依赖图中包含环的语法树。
- 两类特定的SDD一定有一个求值顺序，它们不允许产生带有环的依赖图
  - S属性定义
  - L属性定义
- 这两类SDD可以和自顶向下和自底向上的语法分析过程一起高效地实现。

S属性定义

- 定义：如果一个SDD的每个属性都是综合属性，它就是S属性的。
- 可以按照语法分析树结点的自底向上、后根遍历的顺序来计算它的各个属性值。
- 和自底向上的语法分析过程能够很好的结合起来，对应于与LR分析器将一个产生式体归约成它的头的过程。

```
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:
}
```

L属性定义

- 一个SDD称为L属性定义，对于一个产生式 $A \rightarrow X_1 X_2 ... X_n$ 所关联的语义规则，其中的每个属性：
  - 或者是综合属性
  - 或者是这样的继承属性
    - $X_i.a$ 依赖于 A 的继承属性。
    - $X_i.a$ 依赖于 X_i 左边的符号的属性。
    - X_i 的其它继承或综合属性，但是 X_i 所有的属性组成的依赖图中不存在环。
- S_属性定义也是L_属性定义。

L属性定义

L_属性定义其属性总可按如下方式计算

```
L_dfvisit(n)
{
   for m=从左到右的n的每个子节点 do
   {
         计算m的继承属性;
         L_dfvisit(m);
   }
   计算n的综合属性。
}
```

L属性定义示例

- 正例

产生式	语义规则
$T \to F\ T'$	$T'.inh = F.val$
$T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$

- 反例

产生式	语义规则
$A \to B\ C$	$A.s = B.b;$ $B.i = f(C.c, A.s)$

SDD中的函数

- 一个没有副作用（函数）的SDD有时候也称为属性文法
- 翻译过程有时候需要副作用
  - 打印结果
  - 符号表中加入标识符类型
- 如果能不影响各属性的求值结果，则允许具有受控副作用的语义规则
  - 语义规则的函数被看作是相应产生式头的哑综合属性。

1）$L \rightarrow E\mathbf{n}$	$print(E.val)$

带有副作用的SDD

	产生式	语义规则
1)	$D \rightarrow T\ L$	$L.inh = T.type$
2)	$T \rightarrow \mathbf{int}$	$T.type = \text{integer}$
3)	$T \rightarrow \mathbf{float}$	$T.type = \text{float}$
4)	$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.inh = L.inh$ $addType(\mathbf{id}.entry, L.inh)$
5)	$L \rightarrow \mathbf{id}$	$addType(\mathbf{id}.entry, L.inh)$

简单类型声明的语法制导定义

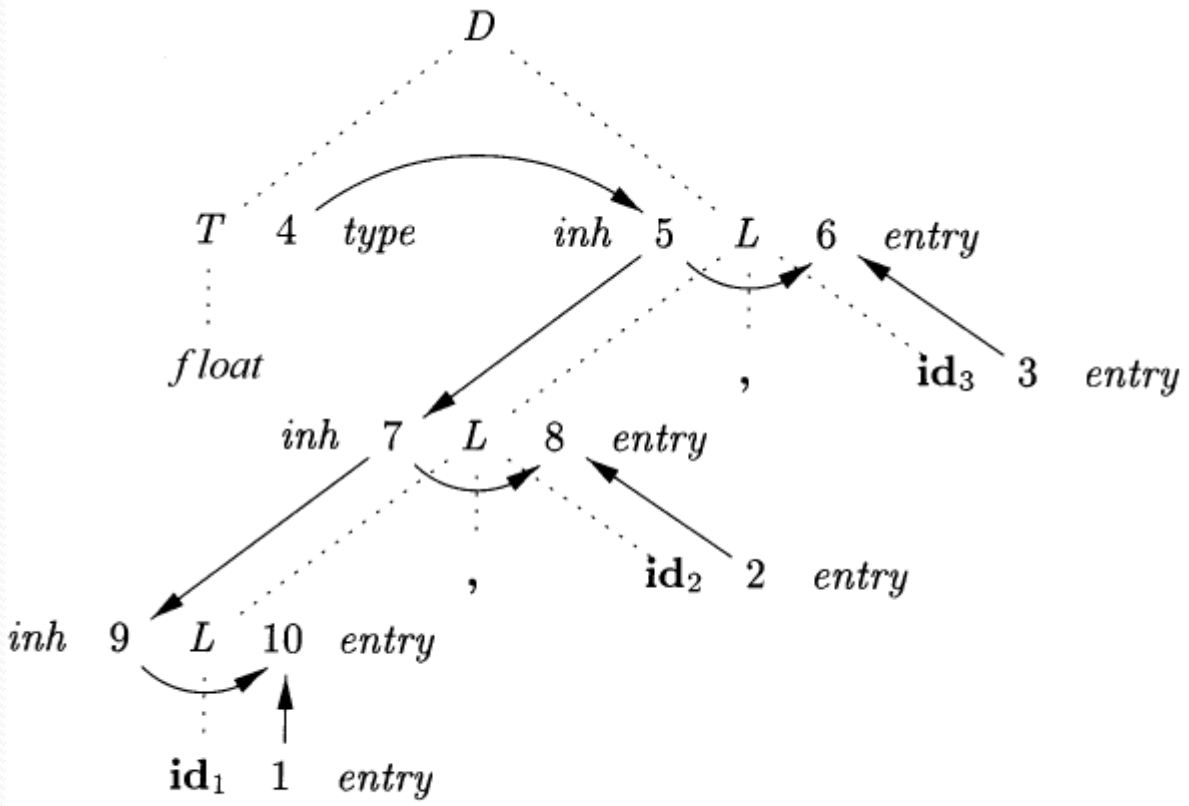


图 5-9　声明 **float** $\mathbf{id}_1$, $\mathbf{id}_2$, $\mathbf{id}_3$ 的依赖图

语法制导翻译的应用－抽象语法树的构造

- 抽象语法树：一种中间表示形式，树中每个结点代表一个程序构造，这个结点的子结点代表这个构造的有意义的组成部分。

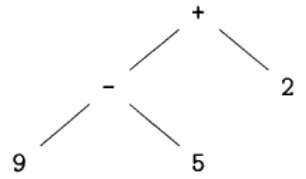


图 2-22　9－5＋2 的语法树

- 抽象语法树上的各个结点可以用具有适当数量的字段的记录对象来实现
  - 叶子结点，构造函数Leaf(id, id.entry),返回指向与叶子结点对应的新纪录的指针。
  - 内部结点，构造函数Node(op, c1,c2,...ck)，第一个字段op表示程序构造，ci指向新建结点的子结点。

抽象语法树的构造

- 作用于抽象语法树上的转换规则更容易完成到中间代码的翻译，因此我们考虑抽象语法树的构造，两个为表达式构造语法树的SDD
  - S属性定义
  - L属性定义

S属性定义为简单表达式文法构造抽象语法树

产生式	语义规则
1) $E \to E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2) $E \to E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3) $E \to T$	$E.node = T.node$
4) $T \to (\ E\ )$	$T.node = E.node$
5) $T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6) $T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

图 5-10　为简单表达式构造语法树

后序遍历，或者在自底向上分析过程中和归约动作一起进行求值。

S属性定义为简单表达式文法构造抽象语法树

产生式	语义规则
1) $E \to E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2) $E \to E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3) $E \to T$	$E.node = T.node$
4) $T \to (\ E\ )$	$T.node = E.node$
5) $T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6) $T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

图 5-10　为简单表达式构造语法树

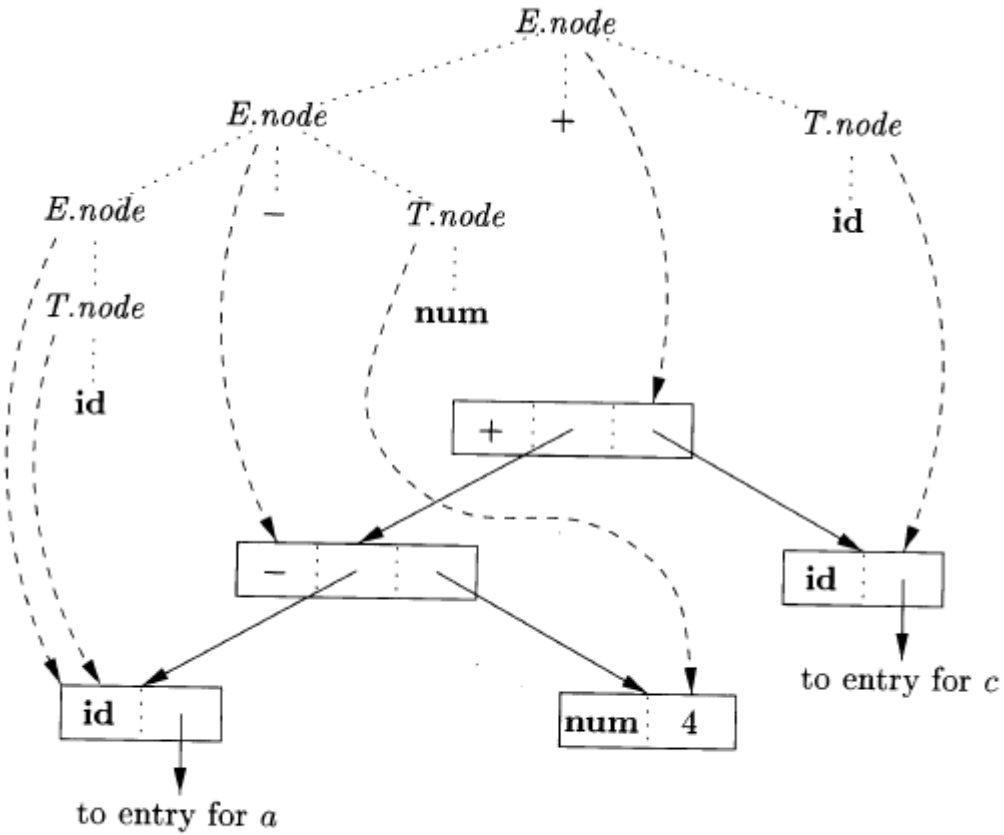


图 5-11　$a-4+c$ 的抽象语法树

后序遍历，或者在自底向上分析过程中和归约动作一起进行求值。

S属性定义为简单表达式文法构造抽象语法树

产生式	语义规则
1) $E \to E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2) $E \to E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3) $E \to T$	$E.node = T.node$
4) $T \to (\ E\ )$	$T.node = E.node$
5) $T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6) $T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

图 5-10　为简单表达式构造语法树

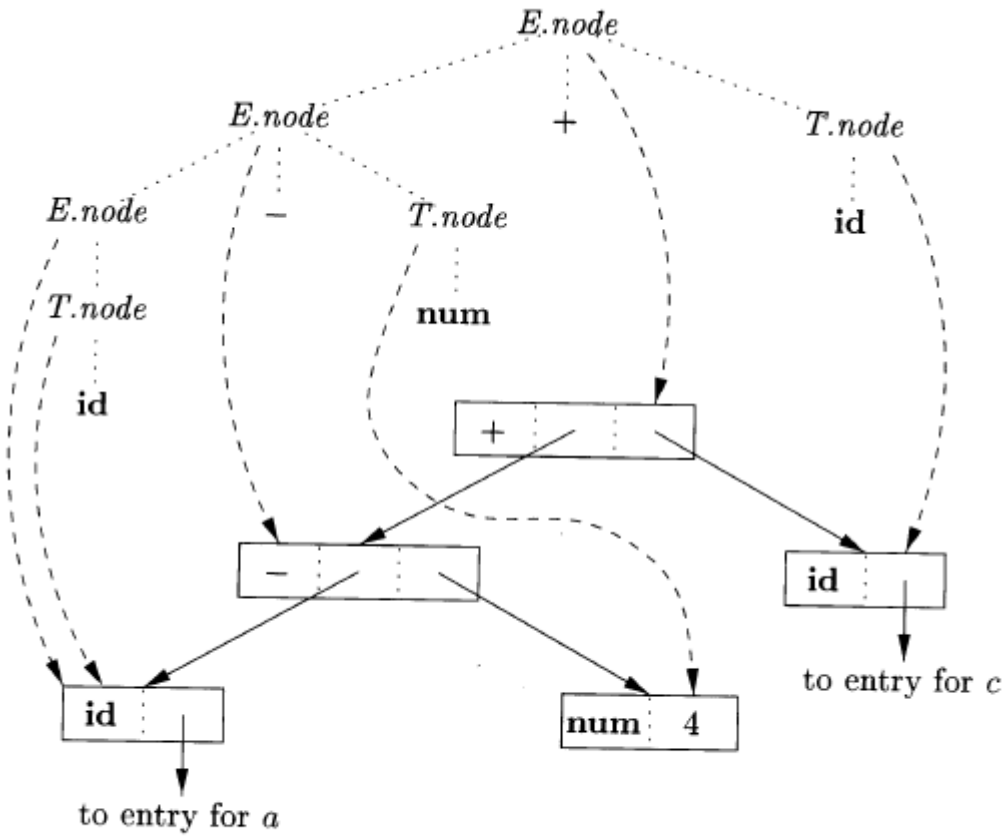


图 5-11　$a-4+c$ 的抽象语法树

1) $p_1 = \textbf{new}\ Leaf(\textbf{id}, entry\text{-}a);$
2) $p_2 = \textbf{new}\ Leaf(\textbf{num}, 4);$
3) $p_3 = \textbf{new}\ Node('-', p_1, p_2);$
4) $p_4 = \textbf{new}\ Leaf(\textbf{id}, entry\text{-}c);$
5) $p_5 = \textbf{new}\ Node('+', p_3, p_4);$

图 5-12　$a-4+c$ 的抽象语法树的构造步骤

后序遍历，或者在自底向上分析过程中和归约动作一起进行求值。

L属性定义为简单表达式文法构造抽象语法树

	产生式	语义规则
1)	$E \to T\ E'$	$E.node = E'.syn$ $E'.inh = T.node$
2)	$E' \to +\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3)	$E' \to -\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4)	$E' \to \epsilon$	$E'.syn = E'.inh$
5)	$T \to (\ E\ )$	$T.node = E.node$
6)	$T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
7)	$T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

图 5-13　在自顶向下语法分析过程中构造抽象语法树

```
L_dfvisit(n)
{
    for m=从左到右的n的每个子节点 do
    {
            计算m的继承属性;
            L_dfvisit(m);
    }
    计算n的综合属性。
}
```

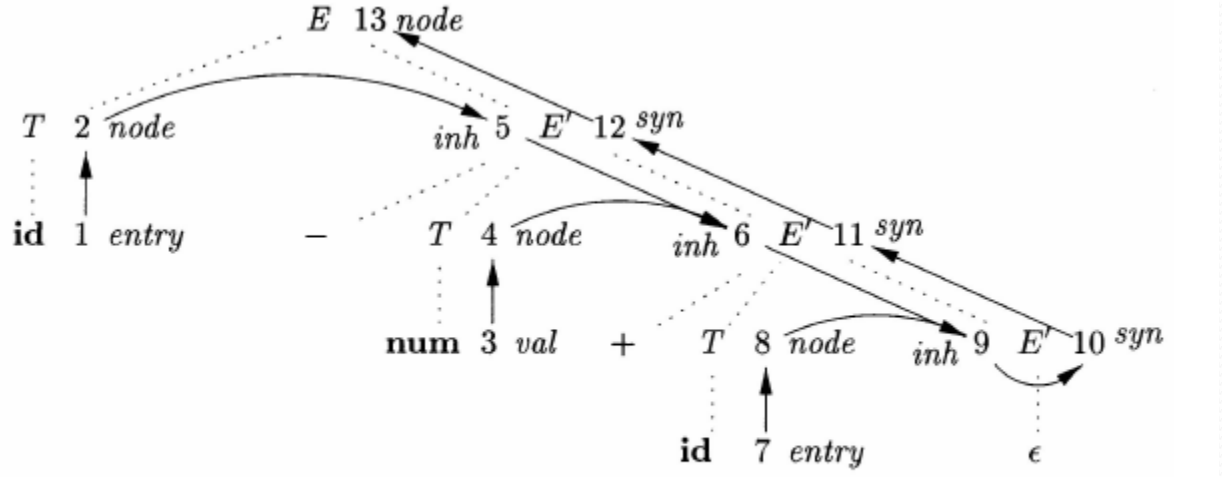
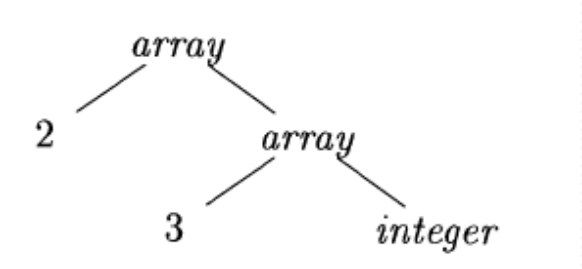


图 5-14　使用图 5-13 中的 SDD 时的 $a-4+c$ 的依赖图

语法制导翻译的应用2－类型的结构

- 数组类型int[2][3]，类型表达式array(2,array(3,integer))



产生式	语义规则
$T \rightarrow B\ C$	$T.t = C.t$
	$C.b = B.t$
$B \rightarrow$ **int**	$B.t = integer$
$B \rightarrow$ **float**	$B.t = float$
$C \rightarrow$ [**num**] C_1	$C.t = array(\mathbf{num}.val,\ C_1.t)$
	$C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$

图 5-16　T 生成一个基本类型或一个数组类型

数组类型语法制导翻译

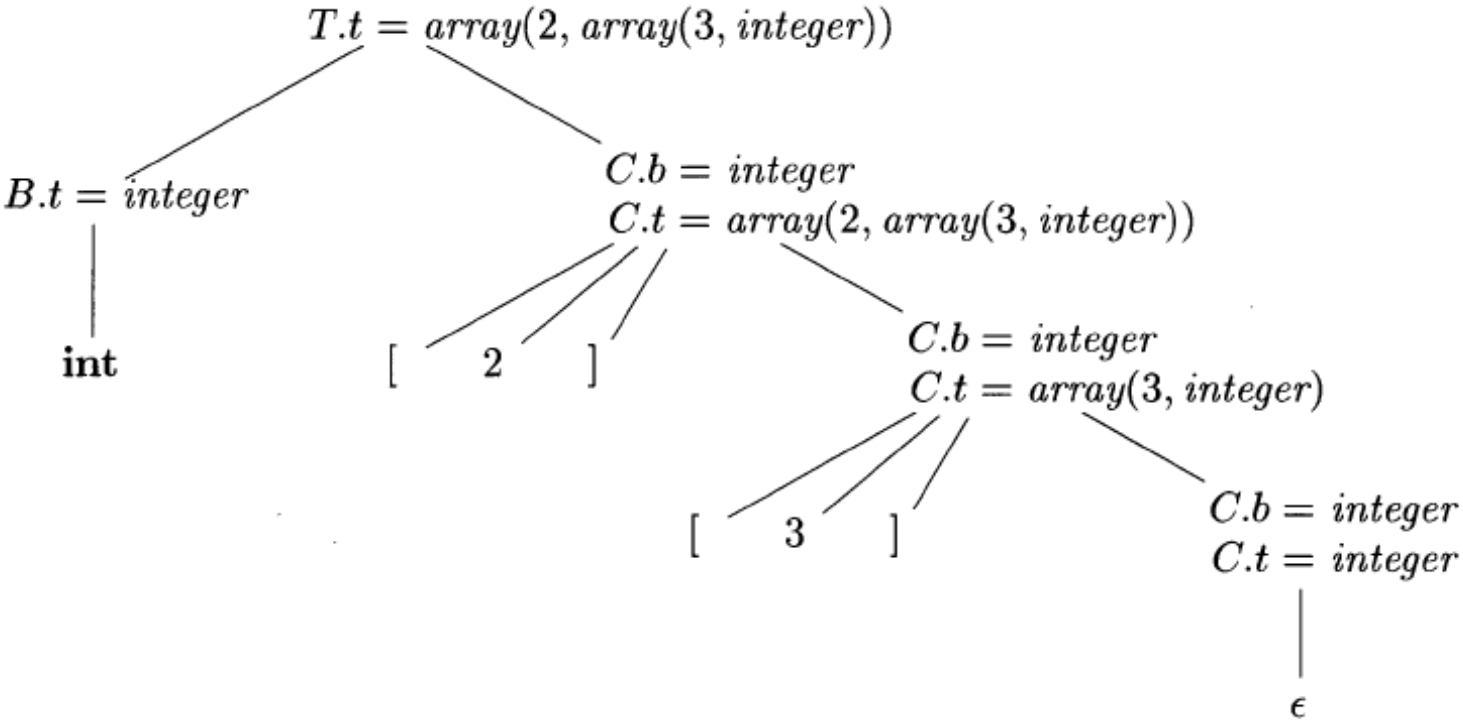


图 5-17　数组类型的语法制导的翻译

语法制导翻译的另一种方式

- 语法制导的翻译方案（Syntax-Directed translation scheme, SDT）
  - 在产生式体中嵌入程序片段的上下文无关文法
  - 程序片段是语义动作，可以出现在体中的任何位置

语法制导定义

- 给出翻译的抽象描述
- 隐藏了语义动作的实现细节
- 附在产生式上的语义规则没有明确告诉我们何时执行其中的语义动作

翻译方案

- 表示语义动作的程序片段可以在产生式体中的任何位置
- 位置信息告诉我们何时执行相应的语义动作
- 翻译方案给出了更多的实现细节信息

语法制导翻译的另一种方式（续）

- 从SDD到SDT
  - 语义规则是如何被转换成为一个带有语义动作的SDT的？语义动作应该放在产生式体中的什么位置？

后缀翻译方案

- 对S属性的SDD，可以构造一个SDT
  - 每个动作都放在产生式的最后
  - 在利用这个产生式进行规约时执行这个动作
- 所有动作在产生式最右端的SDT称为后缀翻译方案

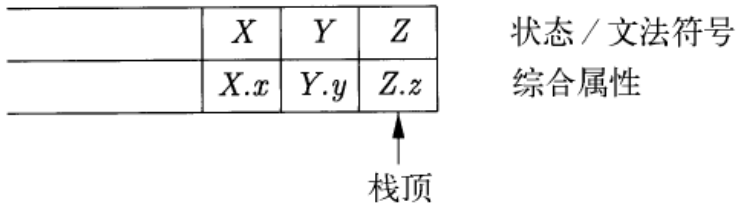
L	$\rightarrow$	E **n**	{ print($E.val$); }
E	$\rightarrow$	$E_1 + T$	{ $E.val = E_1.val + T.val$; }
E	$\rightarrow$	T	{ $E.val = T.val$; }
T	$\rightarrow$	$T_1 * F$	{ $T.val = T_1.val \times F.val$; }
T	$\rightarrow$	F	{ $T.val = F.val$; }
F	$\rightarrow$	(E)	{ $F.val = E.val$; }
F	$\rightarrow$	**digit**	{ $F.val =$ **digit**.$lexval$; }

图 5-18　实现桌上计算器的后缀 SDT

- 例：
  - 基本文法是LR的
  - SDD是S属性的
  - 动作可以和LR分析器的规约步骤同步执行

后缀SDT的分析栈实现

- 将属性和文法符号（状态）一起存放在栈中
- 使用A→XYZ进行规约
  规约后A及其属性在栈顶
- 后缀SDT分析栈实现示例



后缀SDT的分析栈实现

产生式	语义动作
$L \to E$ **n**	{ print($stack[top-1].val$); $top = top - 1$; }
$E \to E_1 + T$	{ $stack[top-2].val = stack[top-2].val + stack[top].val$; $top = top - 2$; }
$E \to T$	
$T \to T_1 * F$	{ $stack[top-2].val = stack[top-2].val \times stack[top].val$; $top = top - 2$; }
$T \to F$	
$F \to (\ E\ )$	{ $stack[top-2].val = stack[top-1].val$; $top = top - 2$; }
$F \to$ **digit**	

状态 / 文法符号
综合属性

图 5-20　在一个自底向上语法分析栈中实现桌上计算器

从SDT中消除左递归

- 若文法存在左递归，则无法利用自顶向下技术进行语法分析。
- 在消除左递归的同时，要考虑如何处理其中的语义动作。
- 最简单的情况，如果每个动作仅仅是打印一个字符串，则把动作当作终结符号处理。

$$
\begin{array}{lcl}
E & \rightarrow & E_1 + T \quad \{\ \text{print}('+');\ \} \\
E & \rightarrow & T
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & T\ R \\
R & \rightarrow & +\ T\ \{\ \text{print}('+');\ \}\ R \\
R & \rightarrow & \epsilon
\end{array}
$$

从SDT中消除左递归（续）

- 对于S属性的SDD，消除左递归有一个通用的框架
- 假设产生式

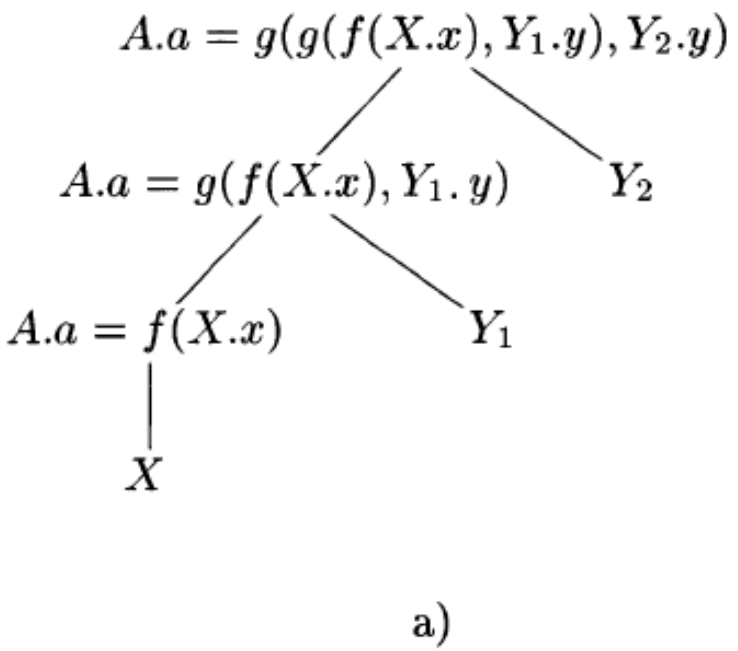
$$
\begin{aligned}
A &\rightarrow A_1\ Y\ \{A.a = g(A_1.a, Y.y)\} \\
A &\rightarrow X\ \{A.a = f(X.x)\}
\end{aligned}
$$

- 基本文法改成

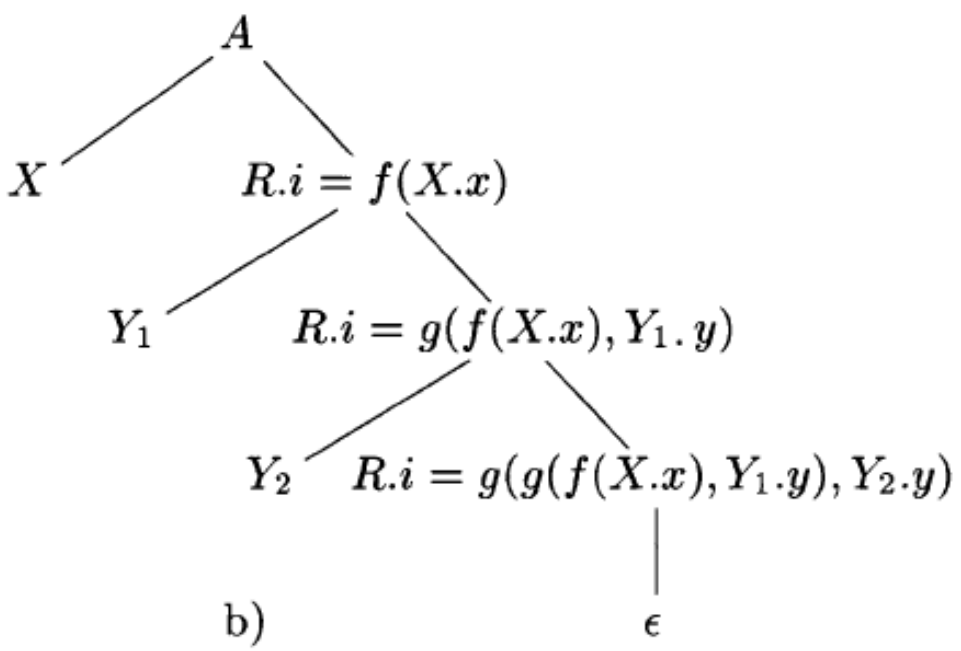
$$
\begin{aligned}
A &\rightarrow X\ R \\
R &\rightarrow Y\ R \mid \epsilon
\end{aligned}
$$

从SDT中消除左递归（续）

- 消除一个后缀SDT中的左递归



a)



b)

从SDT中消除左递归（续）

- 最终得到的消除左递归的SDT

$$
\begin{array}{lcl}
A & \rightarrow & X \ \{R.i = f(X.x)\} \ R \ \{A.a = R.s\} \\
R & \rightarrow & Y \ \{R_1.i = g(R.i, Y.y)\} \ R_1 \ \{R.s = R_1.s\} \\
R & \rightarrow & \epsilon \ \{R.s = R.i\}
\end{array}
$$