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（3）

Syntax-Directed Translation

【第5.5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**

L属性的SDD的例子

- **SDD**

$S \rightarrow \textbf{while}\ (\ C\ )\ S_1$	$L1 = new();$ $L2 = new();$ $S_1.next = L1;$ $C.false = S.next;$ $C.true = L2;$ $S.code = \textbf{label} \parallel L1 \parallel C.code \parallel \textbf{label} \parallel L2 \parallel S_1.code$

- **继承属性：**
  - **next：语句结束后应该跳转到的标号**
  - **true、false：C为真/假时应该跳转到的标号**
- **综合属性code表示代码**

转换为SDT

$S \rightarrow$ **while** (	$\{\ L1 = new();\ L2 = new();\ C.false = S.next;\ C.true = L2;\ \}$
C)	$\{\ S_1.next = L1;\ \}$
S_1	$\{\ S.code = \textbf{label} \parallel L1 \parallel C.code \parallel \textbf{label} \parallel L2 \parallel S_1.code;\ \}$

- 语义动作
  - **L1=new( )和L2=new( )：计算临时值**
  - **C.false = S.next; C.true = L2：计算C的继承属性**
  - **S_1.next = L1：计算S_1的继承属性**
  - **S.code= …：计算S的综合属型**

L属性的SDD的实现方法

- L属性定义应用广泛，但是在实现中会遇到较大的挑战。
- 主要的实现方法有
  - 使用递归下降的语法分析器，为每个非终结符号建立一个函数，在函数中计算属性
  - 使用递归下降的语法分析，边扫描边生成代码（on-the-fly）
  - 与LL语法分析器结合，实现一个SDT
    - （此方法不讲，感兴趣的同学可以自学第5.5.3节）
  - 与LR语法分析器结合，实现一个SDT

（1）递归下降函数法

- 使用递归下降的语法分析器
  - 每个非终结符号对应一个函数
  - 函数的参数接受继承属性
  - 返回值包含了综合属性
- 在函数体中
  - 首先选择适当的产生式
  - 使用局部变量来保存属性
  - 对于产生式体中的终结符号，读入符号并获取其（经词法分析得到的）综合属性
  - 对于非终结符号，使用适当的方式调用相应函数，并记录返回值。

递归下降法实现L属性SDD的例子

```
string S(label next) {
        string Scode, Ccode; /* 存放代码片段的局部变量 */
        label L1, L2; /* 局部标号*/
        if ( 当前输入== 词法单元while ) {
                读取输入 ;
                检查 '(' 是下一个输入符号 , 并读取输入 ;
                L1 = new();
                L2 = new();
                Ccode = C(next, L2);
                检查 ')' 是下一个输入符号 , 并读取输入 ;
                Scode = S(L1);
                return("label" || L1 || Ccode || "label" || L2 || Scode);
        }
        else /* 其他语句类型 */
}
```

（2）On-the-fly (递归下降、边扫描边生成)

- 当属性值很大时，对属性值进行运算的效率很低
  - 比如code（代码）可能是一个上百K的串，对其进行并置等运算会比较低效
  - 可以逐步生成属性的各个部分，并增量式添加到最终的属性值中
- 条件：
  - 存在一个主属性，且主属性是综合属性
  - 在各产生式中，主属性是通过产生式体中各个非终结符号的主属性连接（并置）得到的。同时还会连接一些其它的元素。
  - 各非终结符号的主属性的连接顺序和它在产生式体中的顺序相同

边扫描边生成的基本思想

- 只需要在适当的时候“发出（emit）”非主属性的元素,即把这些元素拼接到适当的地方
- 举例说明
  - S→ while (C) S1
    - S.code = Label || L1 ||C.code ||Label || L2 || S1.code
  - 处理S时，先调用C，再调用S（对应于S1）
  - 如果各个函数把主属性code打印出来，我们处理while语句时，只需要先打印Label L1，再调用C（打印C的代码），再打印Label L2，再调用S（打印S1的代码）
  - 对于这个规则而言，只需要打印Label L1和Label L2
    - 要求C和S的语句在相应情况下跳转到L1和L2。

On-the-fly 的 SDT

$S \rightarrow$ **while** (		$\{\ L1 = new();\ L2 = new();\ C.false = S.next;\ C.true = L2;\ \}$
C)		$\{\ S_1.next = L1;\ \}$
S_1		$\{\ S.code = \textbf{label} \parallel L1 \parallel C.code \parallel \textbf{label} \parallel L2 \parallel S_1.code;\ \}$

S	$\rightarrow$	**while** (	$\{\ L1 = new();\ L2 = new();\ C.false = S.next;$
			$C.true = L2;\ print(\texttt{"label"}, L1);\ \}$
		C)	$\{\ S_1.next = L1;\ print(\texttt{"label"}, L2);\ \}$
		S_1	

- 所有的非终结符号的SDT规则都要这么做

On-the-fly的递归下降代码生成

```
void S(label next) {
        label L1, L2; /* 局部标号 */
        if ( 当前输入 == 词法单元 while ) {
                读取输入 ;
                检查 '(' 是下一个输入符号 , 并读取输入 ;
                L1 = new();
                L2 = new();
                print("label", L1);
                C(next, L2);
                检查 ')' 是下一个输入符号 , 并读取输入 ;
                print("label", L2);
                S(L1);
        }
        else /* 其他语句类型 */
}
```

(3) L属性的自底向上语法分析

- 以LL文法为基础的L属性SDD可以在LR语法分析过程中实现
  - 首先构造出L属性SDD的SDT，即在非终结符号前计算其继承属性
  - 对于A的规则中的语义动作a，引入标记非终结符号M
  - M→ε{a'}，其中a'的构造方法如下：
    - 将a中需要的A或者其它属性作为M的继承属性进行拷贝
    - 按照a中的规则计算各个属性，作为M的综合属性
    - 但是：a'必须设法找到相应的属性，因为产生式M→ε中没有A的符号。

例：算术表达式中缀式->后缀式

算术表达式后缀式翻译方案的等价变换

L→En

E→TR

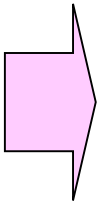
R→＋T {print(‘+’)} R1 |ε

T→FP

P→*F {print(‘*’)} P1 |ε

F→(E)

F→num { print(num.val) }



L→En

E→TR

R→＋TMR1|ε

M→ε {print(‘+’)}

T→FP

P→*FNP1|ε

N→ε {print(‘*’)}

F→(E)

F→num { print(num.val) }

自底向上实现L属性SDD的基本方法

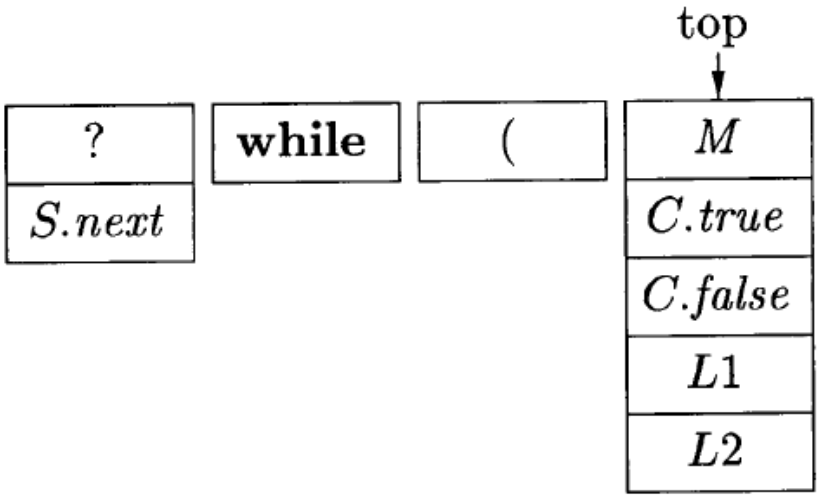
- A→{B.i=f(A.i);}B C
- 引入标记非终结符号M后
  - A → M B C
  - M → ε {M.i=A.i；M.s=f(M.i);}
- 如何找到A.i?
  - 设法使得在即将把BC归约到A时，A的继承属性存放在分析栈中BC的下方。
  - 当执行到M的归约时，A.i的值存放在M的下方。(如果产生式右部为KMBC，那么M的下方为K，K的下方存放A.i)
  - M.s即B.i，存放在M所在的位置，即将归约到B时，B.i存放在归约位置的下方。

自底向上实现L属性的while语句翻译

$S \rightarrow$ **while** (	$\{\ L1 = new();\ L2 = new();\ C.false = S.next;\ C.true = L2;\ \}$
C)	$\{\ S_1.next = L1;\ \}$
S_1	$\{\ S.code = \textbf{label} \parallel L1 \parallel C.code \parallel \textbf{label} \parallel L2 \parallel S_1.code;\ \}$

- SDT转换后得到
  - S→while (M C) N S_1
  - M→ ε　N→ ε
- 按照此产生式归约时
  - S.next位于栈中右部的下方
  - C的继承属性true、false位于栈中紧靠C的下方
  - S_1的next紧靠S_1的下方，即N的栈记录中。
  - S和S_1的综合属性code出现在相应的栈记录中。

自底向上实现L属性的while语句翻译（2）

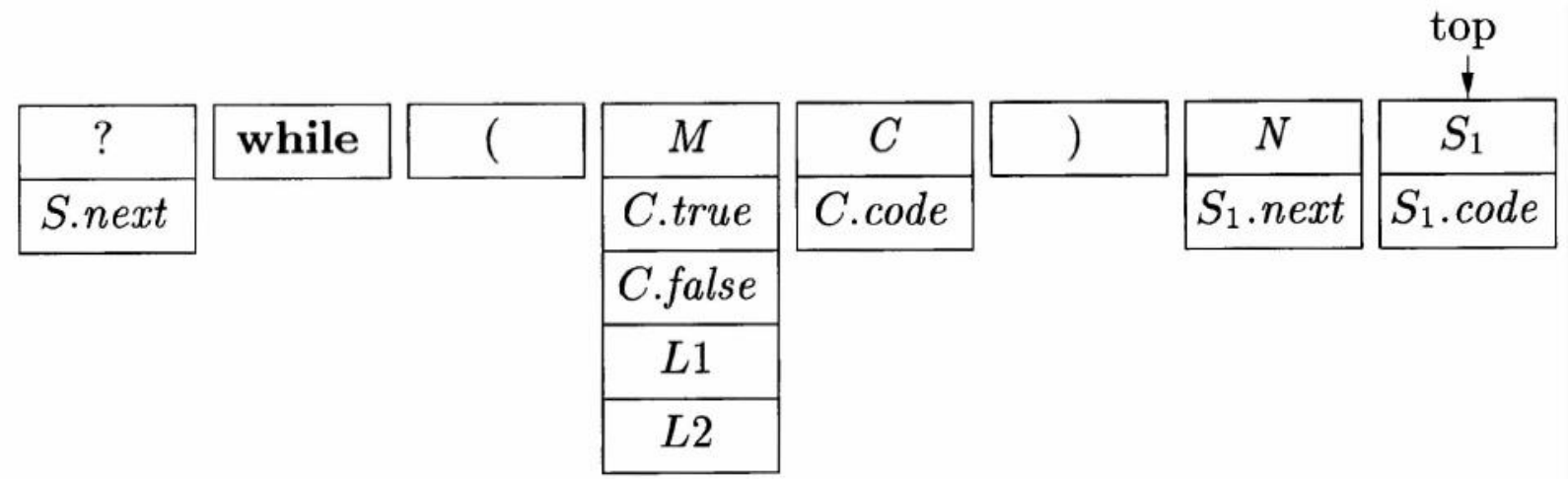


在将ϵ 归约到M的过程中执行的代码

$L1 = new();$
$L2 = new();$
$C.true = L2;$
$C.false = stack[top - 3].next;$

- **stack[top-3]访问了S.next**
- **C.true、C.false存放在M的记录中**

自底向上实现L属性的while语句翻译（3）



- S_1.next被存放在N的栈记录中，它恰好存放在S_1的栈记录之下
- S_1.next = stack[top-3].L1

生成while语句代码的自底向上SDT

$S \rightarrow$ **while** (	{ $L1 = new()$; $L2 = new()$; $C.false = S.next$; $C.true = L2$; }
C)	{ $S_1.next = L1$; }
S_1	{ $S.code =$ **label** $\|\|$ $L1$ $\|\|$ $C.code$ $\|\|$ **label** $\|\|$ $L2$ $\|\|$ $S_1.code$; }

S→while (M C) N S1

tempCode = label || stack[top-4].L1
|| stack [top-3].code || label || stack[top-4].L2
|| stack[top].code;

top = top – 6;

stack[top].code = tempCode;

M → ε

top=top+1; L1 = new(); L2 = new();

C.true = L2; C.false = stack[top-3].next;

N → ε

top=top+1; S_1.next = stack[top-3].L1;

确定继承属性在分析栈中的位置

- 综合属性值很容易在栈中(s[i].val)找到，因此在自底向上分析中处理L属性定义的关键是确定继承属性值在栈中的位置。

- 实际使用中，X的继承属性X.i通常和文法符号Y的综合属性值Y.s有关
  - 或者是Y.s的直接拷贝
  - 或者是Y.s值的函数值

例1：
X的继承属性X.i的值是Y.s的直接拷贝

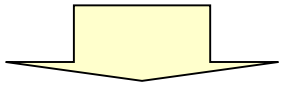
D→T { L.i=T.type } L

T→int　　{T.type=integer}

T→float　　{T.type=float}

L→{ L1.i=L.i } L1, id { addType(id.entry,L.i)}

L→id　　{ addType(id.entry,L.i)}



D→T L　　（无需代码）

T→int　　s[top].val = integer

T→float　　s[top].val = float

L→ L1, id　　addType(s[top].val, s[top-3].val)

L→id　　addType(s[top].val, s[top-1].val)

例2：
无法直接知道继承属性值在分析栈中的位置

S → aA {C.i = A.s} C
S → bAB {C.i = A.s} C
C → c {C.s = g(C.i)}

S → aA {C.i = A.s} C
S → bAB {M.i = A.s} M {C.i = M.s} C
M → ε {M.s = M.i}
C → c {C.s = g(C.i)}

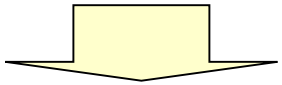
S → aAC
S → bABMC
M → ε　　s[newtop].val = s[top-1].val
C → c　　s[newtop].val = g(s[top-1].val)

- 首先需要对翻译方案进行等价改写
- 然后就可以在栈中找到继承属性的位置。

例3：
继承属性不是直接拷贝，需要通过计算得到

$S \rightarrow aA \{C.i = f(A.s)\}\ C$

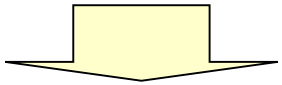
$C \rightarrow c \{C.s = g(C.i)\}$



$S \rightarrow aA \{M.i = A.s\}\ M \{C.i = M.s)\}\ C$

$M \rightarrow \varepsilon \{M.s = f(M.i)\}$

$C \rightarrow c \{C.s = g(C.i)\}$



$S \rightarrow aAMC$

$M \rightarrow \varepsilon$ s[newtop].val = f(s[top].val)

$C \rightarrow c$ s[newtop].val = g(s[top-1].val)

本章小结

- 语法制导定义（SDD）
  - 综合属性
  - 继承属性
  - 属性求值
- 语法制导的翻译方案（SDT）
  - 自顶向下翻译
    - 消除翻译方案中的左递归
  - 自底向上翻译
    - 消除嵌入在产生式中间的动作

作业

- 11月13日交
- 5.4.4-5.4.5
  - (都只做第1小题if语句)
- 5.5.1-5.5.2, 5.5.5
  - (都只做第1小题if语句)