

语义分析

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 10 日





一对概念



两类属性定义



三种实现方式



四大应用

Definition (语法制导的翻译方案 (Syntax-Directed Translation Scheme; SDT))

SDT 是在其产生式体中嵌入**语义动作**的上下文无关文法。

语义动作可以嵌入在产生式体中的任何位置

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

L	\rightarrow	$E n$	$\{ \text{print}(E.val); \}$
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	$\{ F.val = \text{digit.lexval}; \}$

语义动作

语义动作可以嵌入在产生式体中的任何位置

- 1) $L \rightarrow E \mathbf{n}$
- 2) $E \rightarrow \{ \text{print}('+'); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}('*'); \} T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

前缀表达式 SDT

语义动作可以嵌入在产生式体中的任何位置

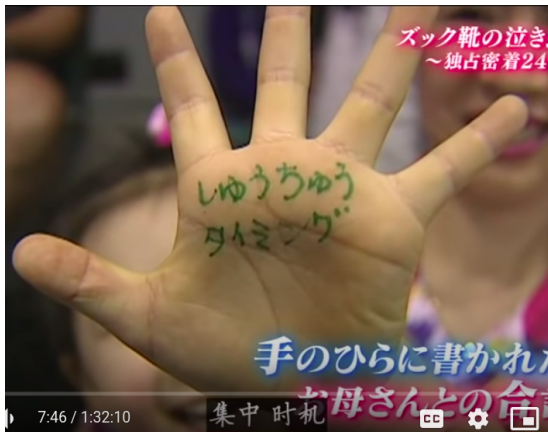
- 1) $L \rightarrow E n$
- 2) $E \rightarrow \{ \text{print}(' + '); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

前缀表达式 SDT

语义动作嵌入的位置决定了**何时**执行该动作

基本思想: 一个动作在它**左边的**所有文法符号都**处理**过之后立刻执行

时机 (Timing; タイミング)



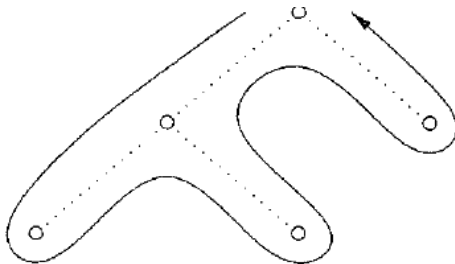
语义动作嵌入在什么地方？何时执行语义动作？

Q: 如何将 SDD 中的**语义规则**转换为带有**语义动作**的 SDT

	S 属性定义	L 属性定义
Offline		
LR		
LL		

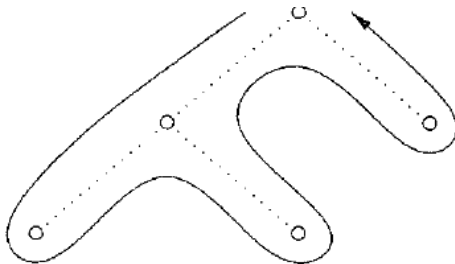
Q: 如何以**三种方式**实现 SDT?

Offline 方式: 已有语法分析树



按照**从左到右**的**深度优先**顺序遍历语法分析树

Offline 方式: 已有语法分析树

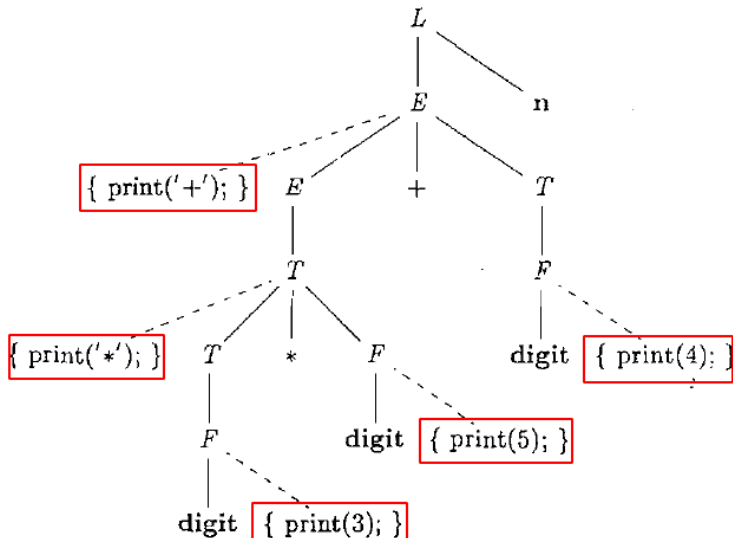


按照**从左到右**的**深度优先**顺序遍历语法分析树

基本思想: 一个动作在它**左边的**所有文法符号都**处理**过之后立刻执行

- 1) $L \rightarrow E \text{ n}$
- 2) $E \rightarrow \{ \text{print}(' + '); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

嵌入语义动作**虚拟节点**的语法分析树



$3 * 5 + 4 \Rightarrow + * 354$

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR		
LL		

基本思想: 一个动作在它**左边的**所有文法符号都**处理**过之后立刻执行

基本思想: 一个动作在它**左边的**所有文法符号都**处理**过之后立刻执行



Q: 是否所有的 SDT 都可以在 *LL/LR* 语法分析过程中实现?

该 SDT 无法在 $LL(1)/LR(1)$ 中实现

- 1) $L \rightarrow E n$
- 2) $E \rightarrow \{ \text{print}(' + '); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

前缀表达式 SDT

该 SDT **无法**在 $LL(1)/LR(1)$ 中实现

- 1) $L \rightarrow E n$
- 2) $E \rightarrow \{ \text{print}('+'); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}('*'); \} T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

前缀表达式 SDT

它需要在还不知道出现在输入中的运算符是 * 还是 + 时,
就执行打印这些运算符的操作

Q: 如何判断某 SDT 是否可以在 LL/LR 语法分析过程中实现?

Q: 如何判断某 SDT 是否可以在 LL/LR 语法分析过程中实现?

将每个内嵌的语义动作 A 替换为一个独有的**非终结符** M

添加新产生式 $M \rightarrow \epsilon$

判断新产生的文法是否可用 LL/LR 进行分析

前缀表达式 SDT

- 1) $L \rightarrow E \mathbf{n}$ **M2**
- 2) $E \rightarrow \{ \text{print}(' + '); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} T_1 * F$ **M4**
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.} \textit{lexval}); \}$ **M7**

$$M_2 \rightarrow \epsilon$$

$$M_4 \rightarrow \epsilon$$

$$M_7 \rightarrow \epsilon$$

- 1) $L \rightarrow E \text{ n}$ **M2**
- 2) $E \rightarrow \{ \text{print}(' + '); \} E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} T_1 * F$ **M4**
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$ **M7**

$$M_2 \rightarrow \epsilon$$

$$M_4 \rightarrow \epsilon$$

$$M_7 \rightarrow \epsilon$$

$$[A \rightarrow \alpha \cdot B\beta, a] \in I \implies \forall b \in \text{FIRST}(\beta a). [B \rightarrow \cdot \gamma, b] \in I$$

1)	L	\rightarrow	$E \text{ n}$	M2
2)	E	\rightarrow	$\{ \text{print}(' + '); \}$	$E_1 + T$
3)	E	\rightarrow	T	M4
4)	T	\rightarrow	$\{ \text{print}(' * '); \}$	$T_1 * F$
5)	T	\rightarrow	F	
6)	F	\rightarrow	(E)	
7)	F	\rightarrow	$\text{digit } \{ \text{print}(\text{digit.}lexval); \}$	M7

$$M_2 \rightarrow \epsilon$$

$$M_4 \rightarrow \epsilon$$

$$M_7 \rightarrow \epsilon$$

$$[A \rightarrow \alpha \cdot B\beta, a] \in I \implies \forall b \in \text{FIRST}(\beta a). [B \rightarrow \cdot \gamma, b] \in I$$

- | | | | | |
|----|-----|---------------|---|-----------|
| 1) | L | \rightarrow | $E \mathbf{n}$ | M2 |
| 2) | E | \rightarrow | $\{ \text{print}(' + '); \}$ | $E_1 + T$ |
| 3) | E | \rightarrow | T | M4 |
| 4) | T | \rightarrow | $\{ \text{print}(' * '); \}$ | $T_1 * F$ |
| 5) | T | \rightarrow | F | |
| 6) | F | \rightarrow | (E) | |
| 7) | F | \rightarrow | $\text{digit} \{ \text{print}(\text{digit.lexval}); \}$ | M7 |

$$M_2 \rightarrow \epsilon$$

$$M_4 \rightarrow \epsilon$$

$$M_7 \rightarrow \epsilon$$

$$L \rightarrow \cdot E \mathbf{n}, \quad \$$$

$$E \rightarrow \cdot \mathbf{M_2} E + T, \quad \mathbf{n}$$

$$E \rightarrow \cdot T, \quad \mathbf{n}$$

$$\mathbf{M_2} \rightarrow \cdot, \quad (/ \text{digit}$$

$$T \rightarrow \cdot \mathbf{M_4} T * F, \quad \mathbf{n}$$

$$T \rightarrow \cdot F, \quad \mathbf{n}$$

$$\mathbf{M_4} \rightarrow \cdot, \quad (/ \text{digit}$$

$$F \rightarrow \cdot (E), \quad \mathbf{n}$$

$$F \rightarrow \cdot \text{digit } \mathbf{M_7}, \quad \mathbf{n}$$

遇到 **digit**, 产生移入/归约冲突

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	✓	
LL		✓

S 属性定义

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

后缀翻译方案

L	\rightarrow	$E n$	$\{ \text{print}(E.val); \}$
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	$\{ F.val = \text{digit.lexval}; \}$

语义动作

S 属性定义

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

后缀翻译方案

L	\rightarrow	$E n$	$\{ \text{print}(E.val); \}$
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	$\{ F.val = \text{digit.lexval}; \}$

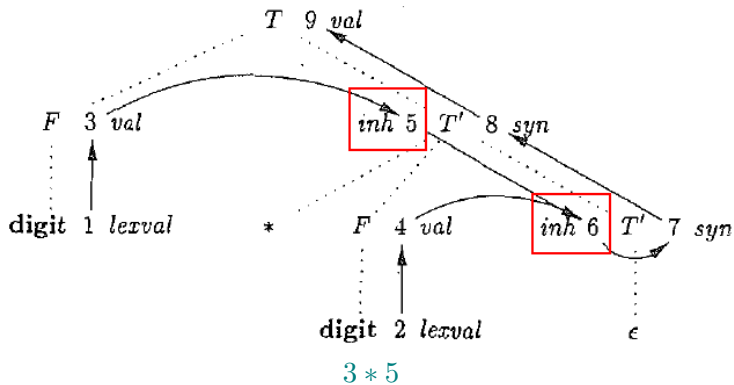
语义动作

后缀翻译方案: 所有动作都在产生式的最后

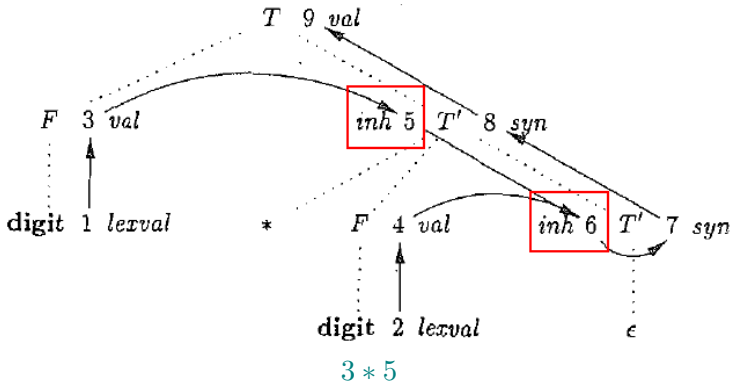
在 LR 中, 按某个产生式**归约**时, 执行相应动作

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	后缀 翻译方案	
LL		✓

L 属性定义 与 LL 语法分析



L 属性定义 与 LL 语法分析



$$A \rightarrow X_1 \cdots X_i \cdots X_n$$

原则: 从左到右处理各个 X_i 符号

对每个 X_i , 先计算继承属性, 后计算综合属性

递归下降子过程 $A \rightarrow X_1 \cdots X_i \cdots X_n$

- ▶ 在调用 X_i 子过程之前, 计算 X_i 的**继承属性**
- ▶ 以 X_i 的继承属性为**参数**调用 X_i 子过程
- ▶ 在 X_i 子过程返回之前, 计算 X_i 的**综合属性**
 - ▶ 在 X_i 子过程中**返回** X_i 的综合属性

(左递归) S 属性定义

$$A \rightarrow A_1 Y \quad A.a = g(A_1.a, Y.y)$$

$$A \rightarrow X \quad A.a = f(X.x)$$

$$XY^*$$

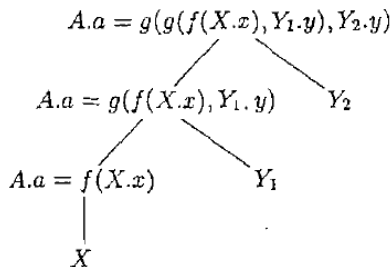
(右递归) L 属性定义

$$A \rightarrow X R \quad R.i = f(X.x); \quad A.a = R.s$$

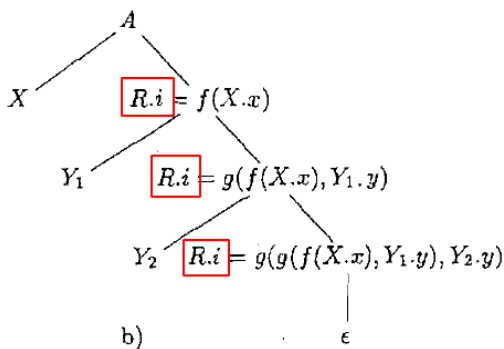
$$R \rightarrow Y R_1 \quad R_1.i = g(R.i, Y.y); \quad R.s = R_1.s$$

$$R \rightarrow \epsilon \quad R.s = R.i$$

继承属性 $R.i$ 用于计算并传递中间结果



a)



b)

c

先计算继承属性, 再计算综合属性

(右递归) L 属性定义

$$A \rightarrow XR \quad R.i = f(X.x); \quad A.a = R.s$$

$$R \rightarrow YR_1 \quad R_1.i = g(R.i, Y.y); \quad R.s = R_1.s$$

$$R \rightarrow \epsilon \quad R.s = R.i$$

原则：继承属性在处理文法符号之前，综合属性在处理文法符号之后

(右递归) L 属性定义

$$A \rightarrow XR \quad R.i = f(X.x); \quad A.a = R.s$$

$$R \rightarrow YR_1 \quad R_1.i = g(R.i, Y.y); \quad R.s = R_1.s$$

$$R \rightarrow \epsilon \quad R.s = R.i$$

原则：继承属性在处理文法符号之前，综合属性在处理文法符号之后

L 属性定义的 SDT

$$A \rightarrow X \quad \{R.i = f(X.x)\} \quad R \quad \{A.a = R.s\}$$

$$R \rightarrow Y \quad \{R_1.i = g(R.i, Y.y)\} \quad R_1 \quad \{R.s = R_1.s\}$$

$$R \rightarrow \epsilon \quad \{R.s = R.i\}$$

$$A \rightarrow X \quad \{\textcolor{red}{R}.i = f(X.x)\} \quad R \quad \{A.a = R.s\}$$
$$R \rightarrow Y \quad \{\textcolor{red}{R}_1.i = g(R.i, Y.y)\} \quad R_1 \quad \{R.s = R_1.s\}$$
$$R \rightarrow \epsilon \quad \{\textcolor{blue}{R}.s = \textcolor{blue}{R}.i\}$$

-
- | | |
|--|-----------------------------------|
| 1: procedure $A()$ | ▷ A 是开始符号, 无需继承属性做参数 |
| 2: if token = ? then | ▷ 假设选择 $A \rightarrow XR$ 产生式 |
| 3: $X.x \leftarrow \text{MATCH}(X)$ | ▷ 假设 X 是终结符, 返回综合属性 |
| 4: $\textcolor{red}{R}.i \leftarrow f(X.x)$ | ▷ 先计算 $\textcolor{red}{R}.i$ 继承属性 |
| 5: $\textcolor{blue}{R}.s \leftarrow R(R.i)$ | ▷ 递归调用子过程 $R(R.i)$ |
| 6: return $\textcolor{red}{R}.s$ | ▷ 返回 $A.a \leftarrow R.s$ 综合属性 |
-

$$A \rightarrow X \quad \{R.i = f(X.x)\} \quad R \quad \{A.a = R.s\}$$
$$R \rightarrow Y \quad \{R_1.i = g(R.i, Y.y)\} \quad R_1 \quad \{R.s = R_1.s\}$$
$$R \rightarrow \epsilon \quad \{R.s = R.i\}$$

1: procedure $R(R.i)$	▷ R 使用继承属性 $R.i$ 做参数
2: if token = ? then	▷ 假设选择 $R \rightarrow YR$ 产生式
3: $Y.y \leftarrow \text{MATCH}(Y)$	▷ 假设 Y 是终结符, 返回综合属性
4: $R.i \leftarrow g(R.i, Y.y)$	▷ 先计算 $R.i$ 继承属性
5: $R.s \leftarrow R(R.i)$	▷ 递归调用子过程 $R(R.i)$
6: return $R.s$	▷ 返回综合属性
7: else if token = ? then	▷ 假设选择 $R \rightarrow \epsilon$ 产生式
8: return $R.i$	▷ 返回 $R.s \leftarrow R.i$ 综合属性



L 属性定义转换为 SDT

$$A \rightarrow X_1 \cdots X_i \cdots X_n$$

计算 X_i **继承属性** 的动作放在产生式体中 X_i 的**左边**

计算产生式头部 A **综合属性** 的动作放在产生式体的**最右边**

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	后缀 翻译方案	
LL		先 继承 , 后 综合

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	后缀 翻译方案	
LL		先 继承 , 后 综合

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	后缀 翻译方案	
LL	无 继承 , 只 综合	先 继承 , 后 综合

$$A \rightarrow \{B.i = f(A.i)\}BC$$

	S 属性定义	L 属性定义
Offline	嵌入语义动作 虚拟节点	
LR	后缀 翻译方案	教材 5.5.4
LL	无 继承 , 只 综合	先 继承 , 后 综合

while 语句的翻译

类型声明与使用 (符号表)

类型检查

要有大局观!!!



认清“你”在语法分析树中所处的位置

```

 $S \rightarrow \text{while} ( C ) S_1 \quad \begin{array}{l} L1 = \text{new}(); \\ L2 = \text{new}(); \\ S_1.\text{next} = L1; \\ C.\text{false} = S.\text{next}; \\ C.\text{true} = L2; \\ S.\text{code} = \text{label} \parallel L1 \parallel C.\text{code} \parallel \text{label} \parallel L2 \parallel S_1.\text{code} \end{array}$ 

```

图 5-27 while 语句的 SDD

```

 $S \rightarrow \text{while} ( C ) S_1 \quad \begin{array}{l} L1 = \text{new}(); \\ L2 = \text{new}(); \\ S_1.\text{next} = L1; \\ C.\text{false} = S.\text{next}; \\ C.\text{true} = L2; \\ S.\text{code} = \text{label} \parallel L1 \parallel C.\text{code} \parallel \text{label} \parallel L2 \parallel S_1.\text{code} \end{array}$ 

```

图 5-27 while 语句的 SDD

$S.\text{code} = \dots \parallel \text{goto } L1$

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

图 5-28 while 语句的 SDT

```

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); C(c.false, c.true)
        检查 ')' 是下一个输入符号, 并读取输入;
        Scode = S(L1); S(S.next)
        return("label" || L1 || Ccode || "label" || L2 || Scode);
    }
    else /* 其他语句类型 */
}

```

图 5-29 用一个递归下降语法分析器实现 while 语句的翻译

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

图 5-28 while 语句的 SDT

$$\begin{array}{ll}
 S \rightarrow \text{while} (& \{ L1 = \text{new}(); L2 = \text{new}(); C.\text{false} = S.\text{next}; C.\text{true} = L2; \} \\
 C) & \{ S_1.\text{next} = L1; \} \\
 S_1 & \{ S.\text{code} = \text{label} \parallel L1 \parallel C.\text{code} \parallel \text{label} \parallel L2 \parallel S_1.\text{code}; \}
 \end{array}$$

图 5-28 while 语句的 SDT

$$\begin{array}{ll}
 S \rightarrow \text{while} (& \{ L1 = \text{new}(); L2 = \text{new}(); C.\text{false} = S.\text{next}; \\
 & C.\text{true} = L2; \text{print}(\text{"label"}, L1); \} \\
 C) & \{ S_1.\text{next} = L1; \text{print}(\text{"label"}, L2); \} \\
 S_1 &
 \end{array}$$

图 5-32 边扫描边生成 while 语句的代码的 SDT

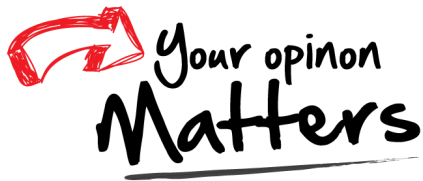
```

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 */
}

```

图 5-31 while 语句的 on-the-fly 的递归下降代码生成

Thank
You!



Office 926

hfwei@nju.edu.cn