

第八章 语法制导翻译 和 中间代码生成

8.6 控制结构的翻译

8.7 说明语句的翻译

广东工业大学计算机学院

控制结构的翻译

- **8.6 控制结构的翻译**
 - **8.6.1 条件转移语句**
 - **8.6.2 开关语句**
 - **8.6.3 for循环语句**
 - **8.6.4 出口语句**
 - **8.6.5 goto语句**
 - **8.6.6 过程调用的四元式产生**

拉链回填技术的复习

■ **if** $a < b$ **or** $c < d$ **and** $e > f$ **then** **S1** **else** **S2** 四元式序列:

■ (1) **if** $a < b$ **goto** ~~0 [E.true]~~ (7)

(2) **goto** (3)



■ (3) **if** $c < d$ **goto** (5)

(4) **goto** ~~0 [E.false]~~ (p+1)

■ (5) **if** $e > f$ **goto** ~~(1)~~ (7)

(6) **goto** ~~(4)~~ (p+1)

■ (7) (关于**S1**的四元式)

...
(p) **goto** (q) /*跳过关于**S2**的四元式*/

■ (p+1) /*关于**S2**的四元式*/

...
(q - 1) ... /* **S2**的四元式末端*/

■ (q) ...

1. 条件转移语句文法

■ 使用文法**G[S]**定义条件转移语句:

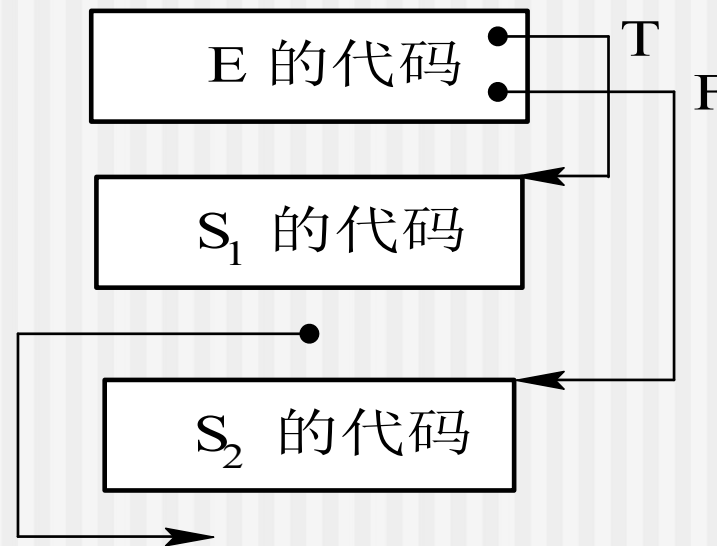
- (1) $S \rightarrow \text{if } E \text{ then } S$
- (2) $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- (3) $S \rightarrow \text{while } E \text{ do } S$
- (4) $S \rightarrow \text{begin } L \text{ end}$
- (5) $S \rightarrow A$
- (6) $L \rightarrow L; S$
- (7) $L \rightarrow S$

■ 其中个非终结符号的意义:

- | | |
|------------------|--------------------|
| ■ S ——语句 | ■ L ——语句串 |
| ■ A ——赋值句 | ■ E ——布尔表达式 |

条件语句 if 的翻译

- 按下面的条件语句 **if** 的模式进行讨论：
- **if E then S₁ else S₂**
- 布尔表达式 **E** 的作用仅在于控制对 **S₁** 和 **S₂** 的选择，因此可将作为转移条件的布尔式 **E** 赋予两种“出口”：
一是“真”出口，出向**S₁**；
一是“假”出口，出向**S₂**。



非终结符 **E** 具有两项语义值 **E.true** 和 **E.false**, 它们分别指出了尚待回填真假出口的四元式串。

E 的“真”出口只有在扫描完布尔表达式 **E** 后的 “**then**” 时才能知道,

E 的“假”出口则需要处理过 **S1** 之后并且到 ‘**else**’ 时才能明确。这就是说, 必须把 **E.false** 的值传下去, 以便到达相应的 **else** 时才进行回填。

S¹ 语句执行完就意味着整个 **if-then-else** 语句也已执行完毕, 因此, 在 **S¹** 的编码之后应产生一条无条件转移指令, 这条转移指令将导致程序控制离开整个 **if-then-else** 语句。

问题！

但是，在完成 **S2** 的翻译之前，这条无条件转移指令的转移目标是不知道的，甚至在翻译完 **S2** 之后仍无法确定！

⑩ 这种情形是由语句的嵌套性所引起的。


语句嵌套的情况

- **if E1 then**
 if E2 then S1 else S2
 else S3
- **S1**后的无条件转移目标地址在何时才能确定？
- 为了方便语句出口的控制，令非终结符**S**和**L**(泛指，除布尔表达式**E**外的其它非终结符)含有一项语义值**S.chain**和**L.chain**。
- **chain**属性把表示都“去往”某语句出口的四元式串在一起，这些四元式期待在翻译完**S**或**L**之后回填转移目标。
- 真正的回填工作在处理**S**的外层环境的某一适当时候完成。

chain属性举例

- 例如，对于语句串：

■ (p) if **E1** then
 S' { if **E2** then **S1** else **S2** } **S**
 (q-1) else **S3** } **W**
 (q) while **E** do **S4**
 (q+1) ...



- **S.chain = q** /* 即 **W.codebegin** */

- **W.chain = q+1** /*在**W**之下一条语句的**cb** */

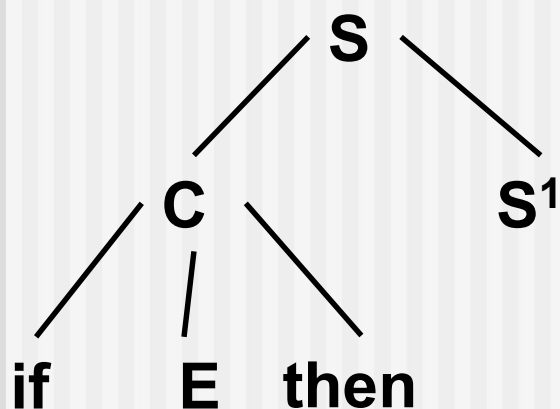
- **S'.chain = q ? S''.chain = E2.f = q**

条件转移语句文法的改写

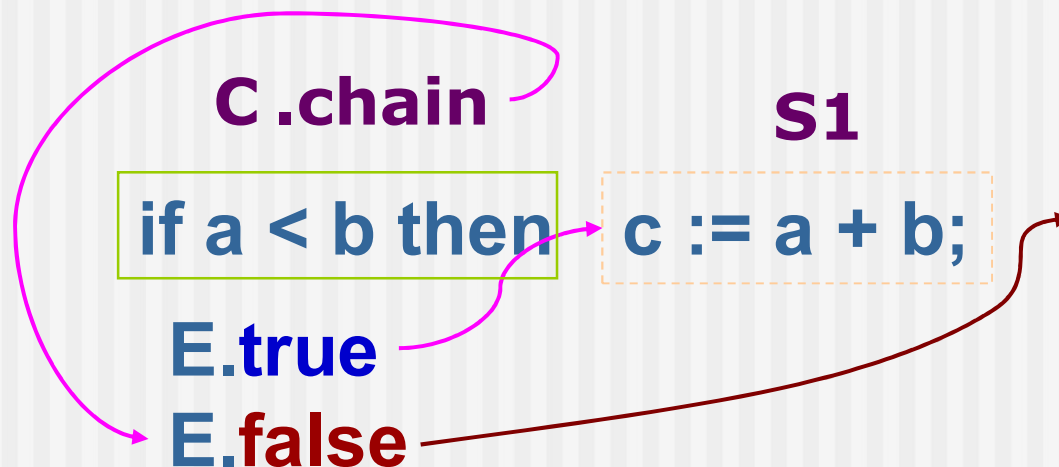
- 为了能及时地回填有关四元式串的转移目标，并使语义动作能置于每个产生式之后，对**G[S]**文法进行改写成**G'[S]**：
- (1) $S \rightarrow CS^1$
- (2) $\quad \quad \quad | T^pS^2$
- (8) $C \rightarrow \text{if } E \text{ then}$
- (9) $T^p \rightarrow C S \text{ else}$
- (3) $S \rightarrow W^dS^3$
- (4) $\quad \quad \quad | \text{begin } L \text{ end}$
- (10) $W^d \rightarrow W E \text{ do}$
- (11) $W \rightarrow \text{while}$

分支语句产生式的语义动作

- (1) $S \rightarrow C S^1$ { $S.\text{chain} :=$
merge($C.\text{chain}, S^1.\text{chain}$) }
- (8) $C \rightarrow \text{if } E \text{ then}$ { backpatch($E.\text{true}, \text{nextstat}$);
 $C.\text{chain} := E.\text{false}$ }



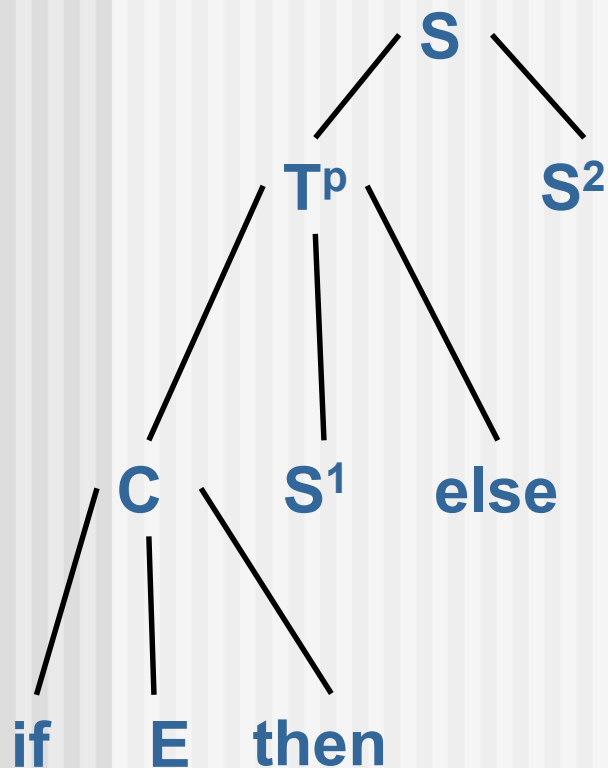
- 例如有语句:



分支语句产生式的语义动作(续1)

■ (2) $S \rightarrow T^p S^2$ { $S.chain :=$
 $merge(T^p.chain, S^2.chain)$ }

■ (9) $T^p \rightarrow C S^1$ else { $q := nextstat;$
 $emit('goto' ____);$
 $backpatch(C.chain, nextstat);$
 $T^p.chain := merge(S^1.chain, q)$ }



(8) $C \rightarrow$ if E then
{ $backpatch(E.true, nextstat);$
 $C.chain := E.false$ }

if E then S^1 else S^2

循环语句产生式的代码结构

□ while E do S¹ 代码结构

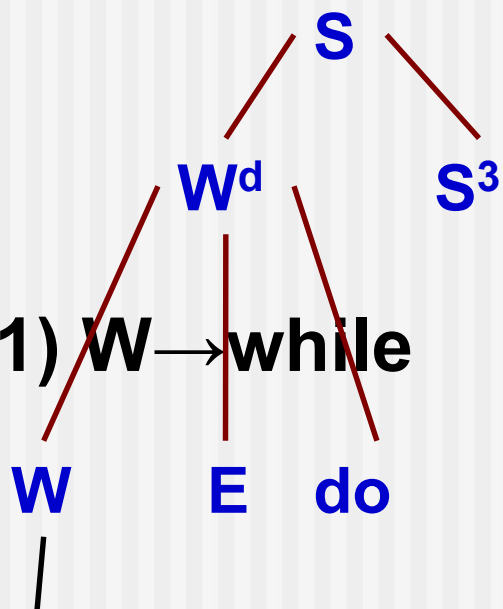


循环语句产生式的语义动作

- (3) $S \rightarrow W^d S^3$ { $\text{backpatch}(S^3.\text{chain}, W^d.\text{codebegin})$
 $\text{emit}(\text{' goto' } W^d.\text{codebegin})$
 $S.\text{chain} := W^d.\text{chain}$ }

- (10) $W^d \rightarrow W E \text{ do}$ { $\text{backpatch}(E.\text{true}, \text{nextstat})$
 $W^d.\text{chain} := E.\text{false}$
 $W^d.\text{codebegin} :=$
 $W.\text{codebegin}$ }

- (11) $W \rightarrow \text{while}$ { $W.\text{codebegin} := \text{nextstat}$ }



其它产生式的语义动作

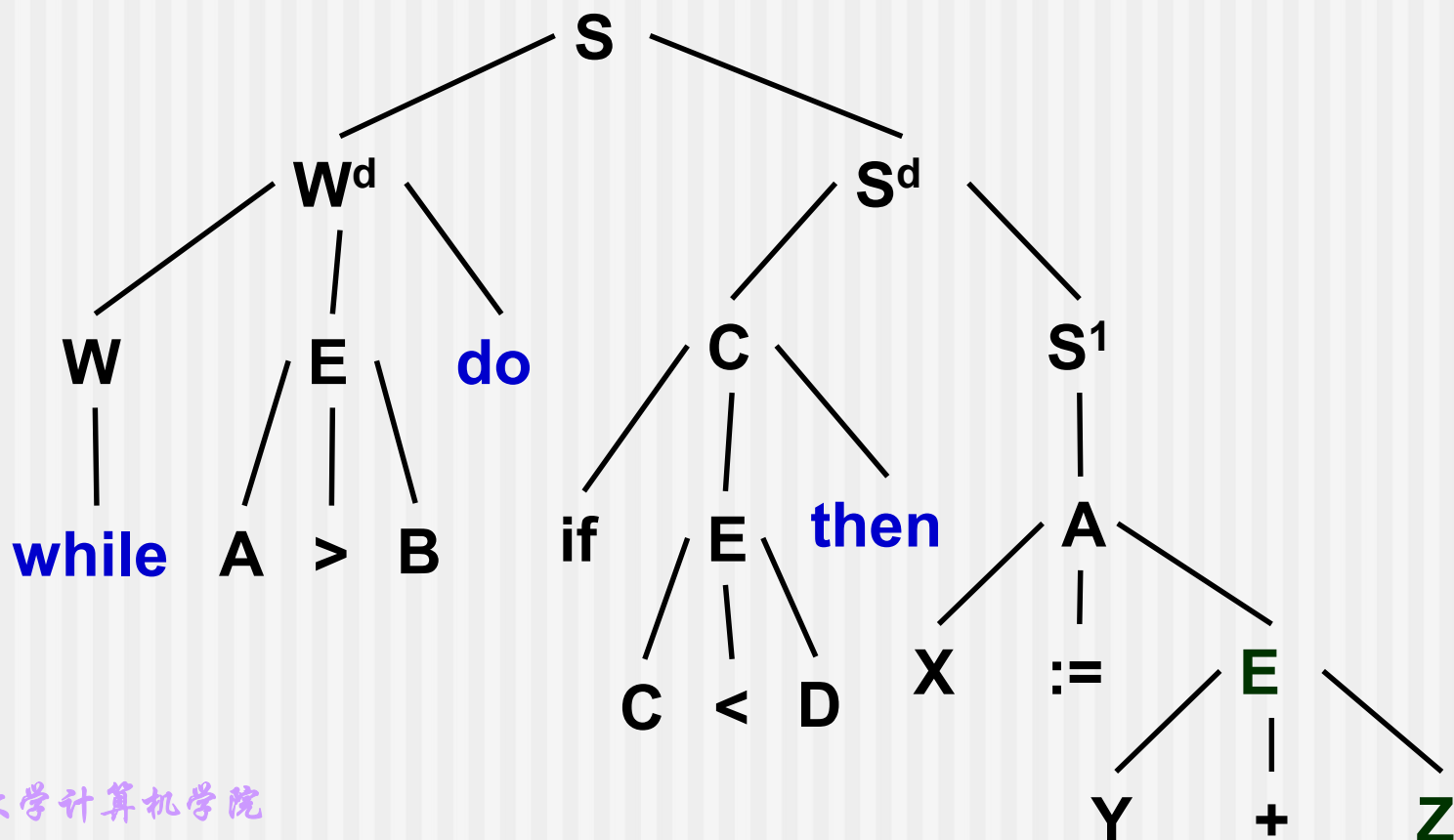
- (4) $S \rightarrow \text{begin } L \text{ end}$ $\{ S.\text{chain} := L.\text{chain} \}$
- (7) $L \rightarrow S$ $\{ L.\text{chain} := S.\text{chain} \}$
- (6) $L \rightarrow L^s S^1$ $\{ L.\text{chain} := S^1.\text{chain} \}$
- (12) $L^s \rightarrow L;$ $\{ \text{backpatch}(L.\text{chain}, \text{nextstat}) \}$
- (5) $S \rightarrow A$ $\{ S.\text{chain} := 0 \text{ /*空链*/ } \}$

控制语句翻译举例

- 翻译语句:

while **A** < **B** **do** **if** **C** < **D** **then** **X** := **Y** + **Z**

- 首先通过语法树理解其规约过程:



控制语句翻译举例

■ 翻译语句:

while **A**<**B** **do** **if** **C**<**D** **then** **X** := **Y** + **Z**

- **100:** **if** **A**<**B** **goto** ~~[E1.true]~~ **102**
- 101:** **goto** ~~[E1.false]~~ **107**
- **102:** **if** **C**<**D** **goto** **104**
- 103:** **goto** ~~106~~ **100**
- **104:** **t1** := **Y** + **Z**
- 105:** **X** := **t1**;
- **106:** **goto** **100**
- **107:** ...

本课内容

■ 8.6 控制结构的翻译

■ 1. 条件转移

■ 2. 开关语句

■ 3. for循环语句

■ 4. 出口语句

■ 5. goto语句

■ 6. 过程调用的四元式产生

2. for循环语句的翻译

- 循环语句的形式(Pascal风格):

for $i := E1$ **step** $E2$ **until** $E3$ **do** $S1$

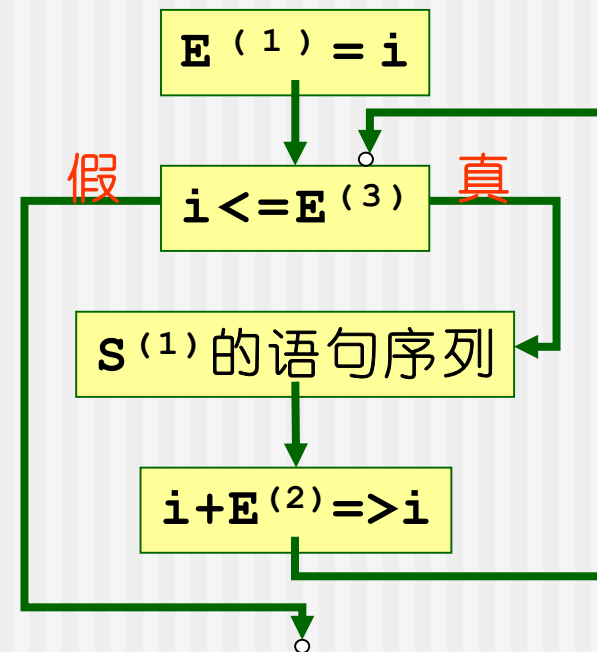
- 翻译成C风格:

for ($i = E1; i \leq E3; i += E2$) { $S1$ }

- 为了简单起见, 假定 $E2$ 总是正的。在这种假定下, 上述循环句的意义等价于:

- $i := E1;$
goto **OVER**;
AGAIN: $i := i + E2;$

OVER: if $i \leq E3$ then
 begin
 $S1;$
 goto **AGAIN**
 end;



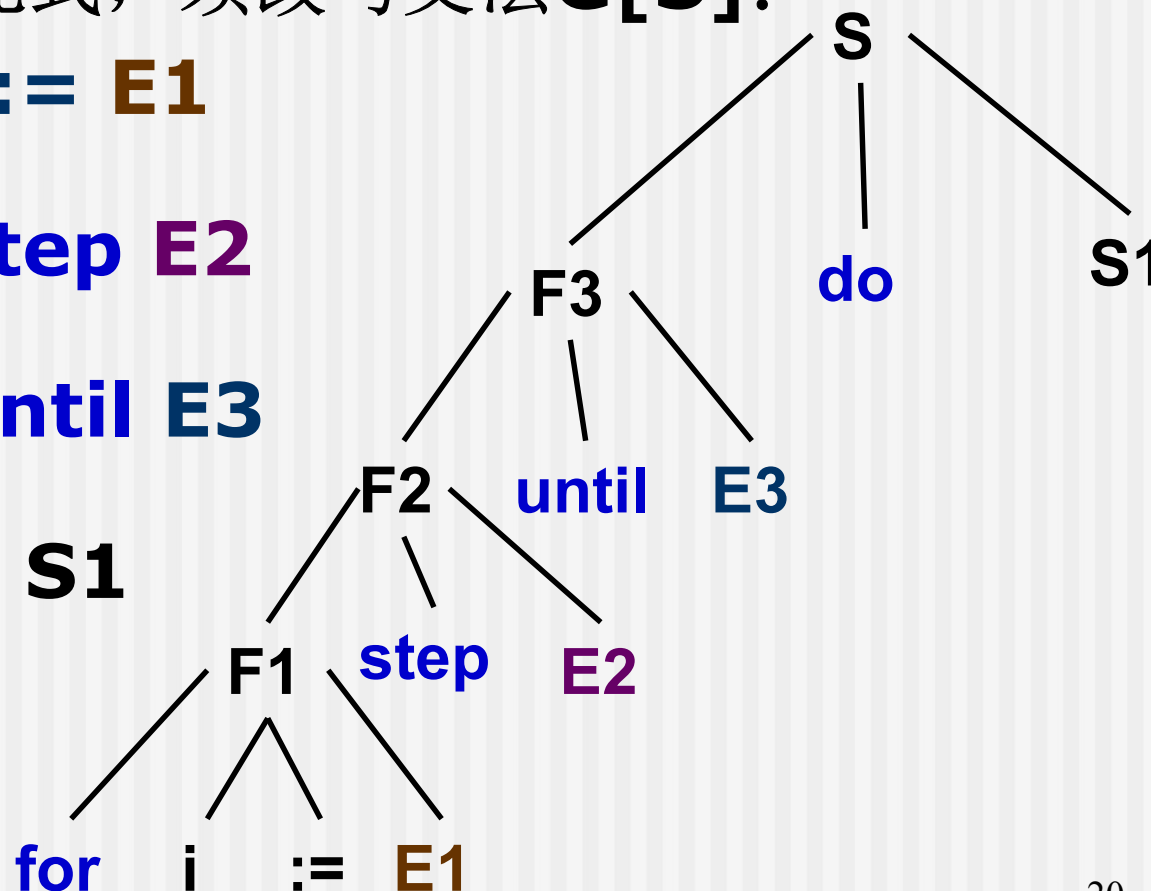
for循环语句的文法

- 为了产生语句

for **i** **:** **=** **E1** **step** **E2** **until** **E3** **do** **S1**

按上述顺序的四元式，须改写文法**G[S]**:

- (1) **F1** \rightarrow **for** **i** **:** **=** **E1**
- (2) **F2** \rightarrow **F1** **step** **E2**
- (3) **F3** \rightarrow **F2** **until** **E3**
- (4) **S** \rightarrow **F3** **do** **S1**



for循环语句文法的翻译方案

- 要翻译的语句模型:

for $i := E1$ **step** $E2$ **until** $E3$ **do** $S1$

- 下面是文法产生式相应的语义动作 (P191):

获得控制变量*i*在符号表中的表项

- (1) $F1 \rightarrow \text{for } i := E1$
- { emit(entry(i), ':=', $E1.place$);
- $F1.place := \text{entry}(i)$;
- $F1.chain := \text{nextstat}$;
- emit('goto' --);
- $F1.codebegin := \text{nextstat};$ }

for语句的循环条件所在的地
址, 即 **AGAIN** (P191)

- 生成的语句序列:
- (1): $\text{entry}(i) := E1.place$
- (2): goto **OVER**
- (3) : **AGAIN**:
- $F1.chain :=$ (2)
- $F1.codebegin :=$ (3)

for循环语句文法的翻译(续1)

- ## ■ 要翻译的语句模型:

for **i** := **E1** **step** **E2** **until** **E3** **do**

S1

- ### ■ 文法产生式相应的语义动作(P191):

AGAIN 所在的地址(3)

- **(2) $F2 \rightarrow F1$ step E2**

- { F2.codebegin := F1.codebegin;

- **F2.place := F1.place;**

- emit(F1.place, '=', E2.place, '+' F1.place);

- ```

 backpatch(F1.chain, nextstat);

```

- ### ■ 生成的语句序列:

- (1)  $\text{entry}(i) := \dots$

- (2) goto (4)

- (3) AGAIN:  $i := E2 + i$

- (4) ...

## 保存控制变量i在符号表中的位置

**F1.chain = (2).**  
**nextstat: OVER**的地址

# for循环语句文法的翻译(续2)

- 要翻译的语句模型:

**for**  $i := E1$  **step**  $E2$  **until**  $E3$   
**do**  $S1$

- 文法产生式相应的语义动作:

- (3)  $F3 \rightarrow F2 \text{ until } E3$

- $\{ F3.cb := F2.cb;$

- $q := \text{nextstat};$

- $\text{emit}(\text{'if' } F2.place \text{ '}\leq\text{'},$   
 $E3.place, \text{'goto' } q+2);$

- $F3.chain := \text{nextstat};$

- $\text{emit}(\text{'goto' } --);$

**F3.chain:** 转离循环的出口,  
也是整条**for**语句的出口

广东工业大学计算机学院

**AGAIN** 所在的地址(3)

- 生成的语句序列:

- (1)  $\text{entry}(i) := \dots$

- (2)  $\text{goto } (4);$

- (3) **AGAIN:**  $i := E2 + i$

- (4) **OVER:** if  $i \leq E$  then  
 $\text{goto } (6)$

- (5)  $\text{goto } \underline{\hspace{2cm}}$

(q) if ( $i \leq E3$ )  $\text{goto } q+2$

(q+1)  $\text{goto}$  循环的出口

(q+2)  $S(1)$  的代码

# for循环语句文法的翻译(续3)

- 要翻译的语句模型:

**for**  $i := E1$  **step**  $E2$  **until**  $E3$   
**do**  $S1$

- 文法产生式相应的语义动作:

- (4)  $S \rightarrow F3$  **do**  $S1$

- */\* 语句S1的相应代码 \*/*

- { emit('goto'  $F3.cb$ )

- backpatch( $S1.chain$ ,  
                                 $F3.cb$ );

- $S.chain := F3.chain$ ; }

**F3.chain = 5**: 转离循环的  
出口, 也是**for**语句的出口

广东工业大学计算机学院

**AGAIN** 所在的地址(3)

- 生成的语句序列:

- (1) entry( $i$ ) := ...

- (2) goto (4);

- (3) **AGAIN**:  $i := E2 + i$

- (4) **OVER**: if  $i \leq E$  then  
                                goto (6)

- (5) goto (p+1)

- (6) ..... */\*S1的代码\*/*  
                                .....

- (p) goto (3)



# for循环语句翻译举例

- 例如循环语句:

**for** **I** := 1 **step** 1 **until** **Y** **do** **X** := **X** + 1

- 将被翻译成如下的四元式序列(对照P191):

- 100      I := 1
- 101      goto \_\_\_\_
- 102      I := I + 1
- 103      if I ≤ Y goto \_\_\_\_
- 104      goto \_\_\_\_
- 105      T := X + 1
- 106      X := T
- 107      goto \_\_\_\_
- 108      .....

该语句的等价意义:

```
 i := 1;
 goto OVER;
AGAIN: i := i + 1;
OVER: if i ≤ Y then
 begin
 X := X + 1;
 goto AGAIN
 end;
```

# 本课内容

---

## ■ 8.6 控制结构的翻译

### ■ 1. 条件转移

### ■ 2. 开关语句

### ■ 3. **for**循环语句

### ■ 4. 出口语句

### ■ 5. **goto**语句

### ■ 6. 过程调用的四元式产生

### 3. 开关语句的形式(了解)

- 开关语句的形式为:
- **switch** **E** **of**
- **case** **V1**: **S1**
- **case** **V2**: **S2**
- .....
- **case** **V<sub>n-1</sub>**: **S<sub>n-1</sub>**
- **default**: **S<sub>n</sub>**
- **end**

- 翻译方案: 分支数较少时

- **t** := **E**;
- **L1**: if **t** ≠ **V1** goto **L2**;  
   **S1**;  
   goto **next**;
- **L2** : if **t** ≠ **V2** goto **L3**;  
   **S2**;  
   goto **next**;
- ...
- **L<sub>n-1</sub>**: if **t** ≠ **V<sub>n-1</sub>** goto **L<sub>n</sub>**;  
   **S<sub>n-1</sub>**;  
   goto **next**;
- **L<sub>n</sub>**: **S<sub>n</sub>**;
- **next**:

# 常用的开关语句翻译方案

- (\* 计算**E**值、把结果放到临时变量**t**的中间代码 \*)

- **goto test**

- **L1: S1**的中间代码  
**goto next**

- **L2: S2**的中间代码  
**goto next**

- .....

- **Ln: Sn**的中间代码  
**goto next**

- **test:** if **t** = **V1** goto **L1**  
if **t** = **V2** goto **L2**  
.....  
if **t** = **Vn-1** goto **Ln-1**  
goto **Ln**

- **next:**

- 开关语句的形式为:

- **switch E of**

- **case V1: S1**

- **case V2: S2**

- .....

- **case Vn-1: Sn-1**

- **default: Sn**

- **end**

# 本课内容

---

## ■ 8.6 控制结构的翻译

### ■ 1. 条件转移

### ■ 2. 开关语句

### ■ 3. **for**循环语句

### ■ 4. 出口语句

### ■ 5. **goto**语句

### ■ 6. 过程调用的四元式产生

# 4. goto语句的处理

- 多数程序语言中的转移是通过标号和**goto**语句实现。
  - 带标号语句的形式: **L: S**
  - **goto**语句的形式: **goto L**
- 对**goto**语句的处理分为两种情况:
- (1) 标号**L**在被**goto**语句使用之前就已经定义(定义性出现)
  - **L: S**
  - .....
  - **goto L**
- (2) 标号**L**在被**goto**语句使用之前还没有被定义(引用性出现)
  - **goto L**
  - .....
  - **L: S**

# goto语句处理的情况(1)

- (1) 标号在被**goto**语句使用之前就已经定义
  - (p) L: S
  - .....
  - (q) goto L
- **goto L**是一个向上转移的语句。查找符号表即可获得**L**的定义地址**p**，可产生相应于此语句的四元式：**(j, -, -, p)**。

| 名字       | 类型 | ..... | 定义否 | 地址       |
|----------|----|-------|-----|----------|
| .....    |    |       |     |          |
| <b>L</b> | 标号 |       | 是   | <b>p</b> |
| .....    |    |       |     |          |

# goto语句处理的情况(2)

- (2) 标号在被**goto**语句使用之前还没有被定义
  - (q) goto L
  - .....
  - (r) L: S
- **goto L**是一个向下转移的语句，细分成两种情况：
- ① 若**L**是第一次出现，则把它填进符号表中。对**goto L**只能产生一个不完全的四元式(**goto** —)

| 名字       | 类型 | ..... | 定义否 | 地址 |
|----------|----|-------|-----|----|
| <b>L</b> | 标号 |       | 未   | -- |



# goto语句处理的情况(2)(续)

## ■ (2) 标号在被goto语句使用之前还没有被定义

■ (p) goto L

■ .....

■ (q) goto L

■ .....

■ (r) goto L

(t) L: S

第一次出现，0  
是链尾标志

四元式序列

.....

(p) L: (goto 0)

.....  
(q) L: (goto p)

.....  
(r) L: (goto q)

.....

■ ②若L不是第一次出现。

■ 则采用“拉链”的方法，把所有以L为转移目标的四元式串在一起。

| 名字    | 定义否 | 地址 |
|-------|-----|----|
| ..... |     |    |
| L     | 未   | r  |
| ..... |     |    |

# 标号语句的文法翻译

- 假定用下面的产生式来定义标号语句

■ (1)  $S \rightarrow \langle \text{label} \rangle S$  (2)  $\langle \text{label} \rangle \rightarrow i:$

- 当用  $\langle \text{label} \rangle \rightarrow i:$  进行归约时，应做如下语义动作：

- 1. 若  $i$  所指的标识符(假定为  $L$ , 即  $L:S$ ) 不在符号表中，则把它填入，置“类型”为“标号”，“定义否”为“已”，“地址”为 **nextstat**。
- 2. 若  $L$  已在符号表中，但“类型”不为“标号”或“定义否”为“已”，则报告出错。

| 名字       | 类型 | ... | 定义否 | 地址        |
|----------|----|-----|-----|-----------|
|          |    | ... |     |           |
| <b>L</b> | 标号 |     | 已   | <b>ns</b> |

# 标号语句的文法翻译(续)

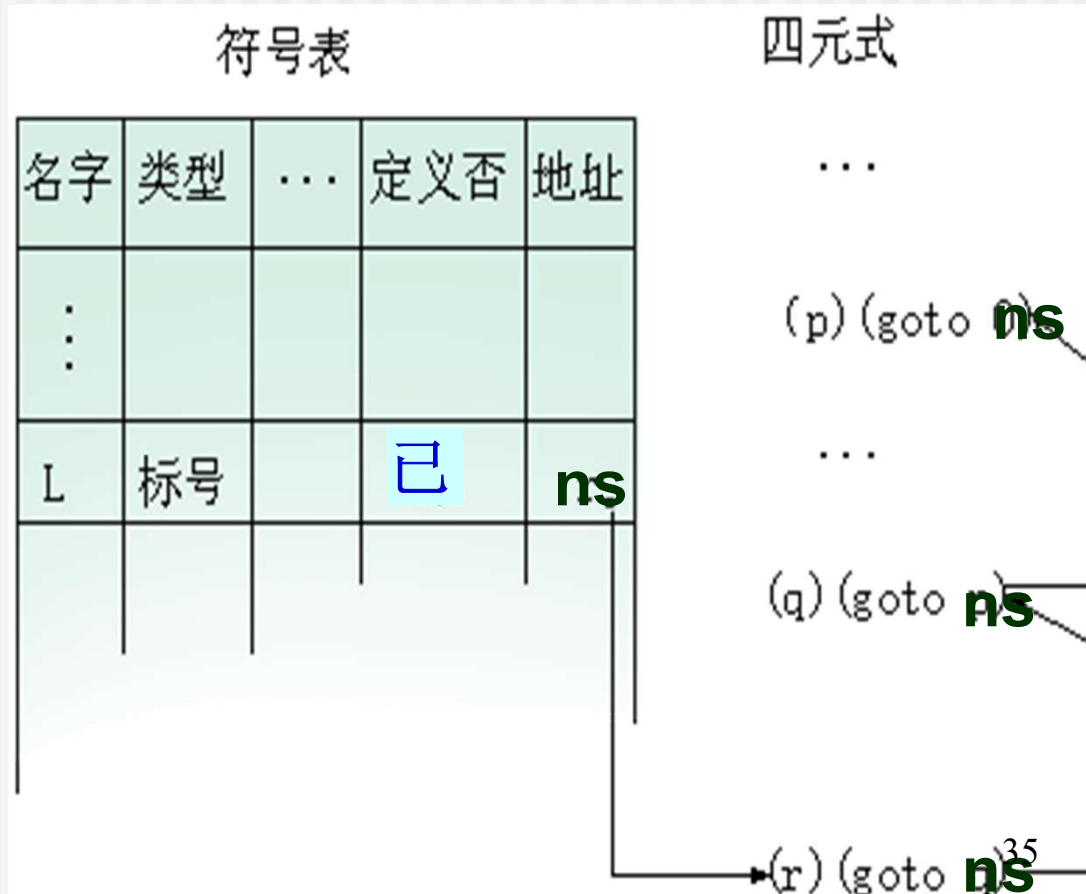
- 假定用下面的产生式来定义标号语句

■  $(p) S \rightarrow \langle \text{label} \rangle S$        $(q) \langle \text{label} \rangle \rightarrow i:$

- 当用  $\langle \text{label} \rangle \rightarrow i:$  进行归约时，应做如下语义动作：

- 3. 若 **L** 已在符号表中，则：
  - ① 把标志“未”改为“已”；
  - ② 把地址栏中的链首（记为 **r**）取出，同时把 **nextstat** 填在表中；
  - ③ 执行 **backpatch(r, nextstat)**。

广东工业大学计算机学院



# 本课内容

---

- **8.6** 控制结构的翻译
- **8.7** 说明语句的翻译

# 简单说明句的文法

- 程序设计语言中最简单的说明语句的语法描述为：
- $D \rightarrow \text{integer} \langle \text{namelist} \rangle \mid \text{real} \langle \text{namelist} \rangle$
- $\langle \text{namelist} \rangle \rightarrow \langle \text{namelist} \rangle, \text{id} \mid \text{id}$
- 使用关键字**integer**和**real**定义能够一串变量的性质。对这种说明句的翻译是指在符号表中登录该名和性质。
- 上述文法制导翻译存在这样的问题：
- 需要把所有的名字都规约成**namelist**后，才能把它们性质登记进符号表。
- 即**namelist**必须用一个队列或栈来保存所有这些名字。

# 简单说明句的翻译

- 为避免使用栈或者队列，可以把上述的文法改写成：

- $D \rightarrow D^1, id$ 
  - | integer id
  - | real id

**att:** 记录说明句所引入的  
名字的性质(**int**或**real**)

- (1)  $D \rightarrow \text{integer } id$

```
{ enter(id, int);
 D.att := int }
```

- (2)  $D \rightarrow \text{real } id$

```
{ enter(id, real);
 D.att := real }
```

- (3)  $D \rightarrow D^1, id$

```
{ enter(id, D1.att);
 D.att := D1.att }
```

# 过程(函数)中的说明

- 过程的翻译包括两部分：**过程说明**和**处理调用**。现在主要讨论对过程中说明的局部名字的处理。
- **(1)** 在建立符号表时，要登录过程的**局部变量名**和**存储的相对地址**。
- **(2)** 为记录相对地址，可以使用一个变量**offset**，用于记录当前可用空间的开始地址。
- **offset**的增加值称为数据对象的宽度，用属性**width**来表示。由该名字的类型决定。例如：
- **D** → **real id** { **enter(id, real, offset);**  
    **D.att := real;**  
    **D.width := 8;**  
    **offset := offset + D.width; }**

# 作业

---

## ■ 整章一起布置