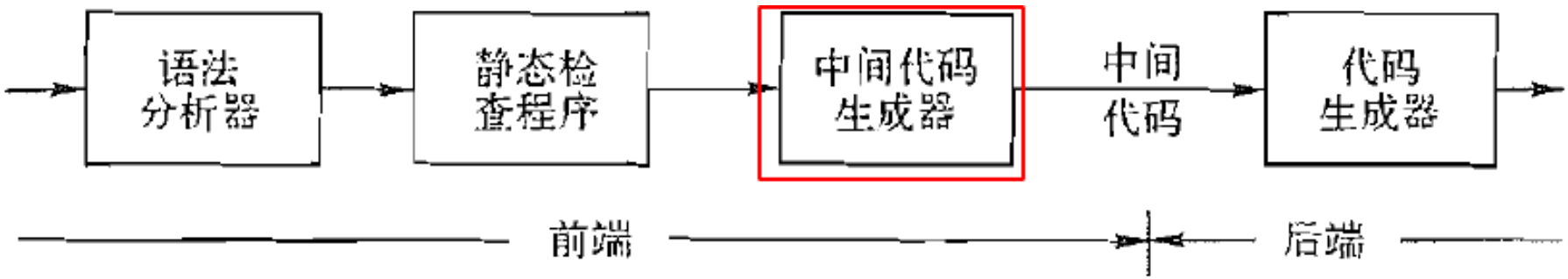
中间代码生成

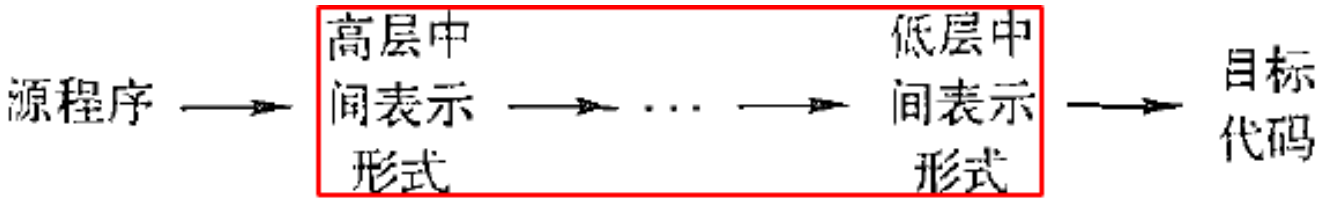
魏恒峰

hfwei@nju.edu.cn

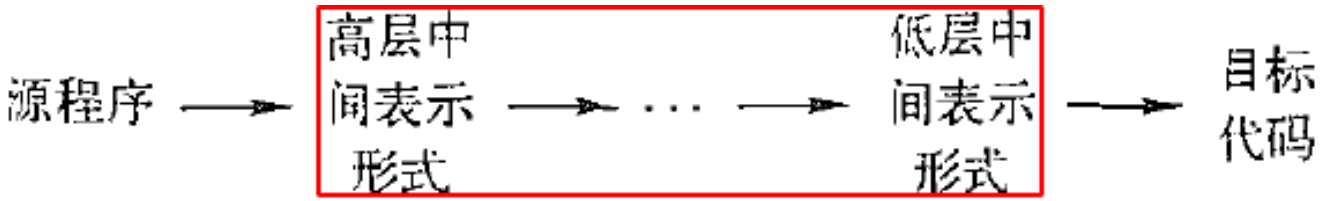
2021 年 5 月 5 日


语法
分析器
静态检
查程序
中间代码
生成器
中间
代码
代码
生成器
前端
后端

Intermediate Representation (IR)



Intermediate Representation (IR)



精确: 不能丢失源程序的信息

独立: 不依赖特定的源语言与目标语言

(如, 没有复杂的寻址方式)

Intermediate Representation (IR)

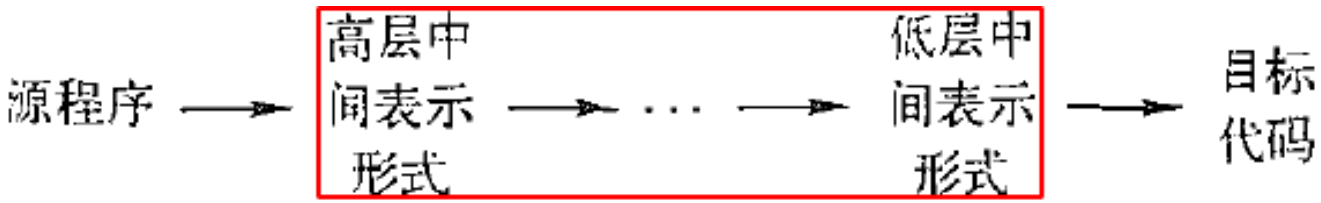
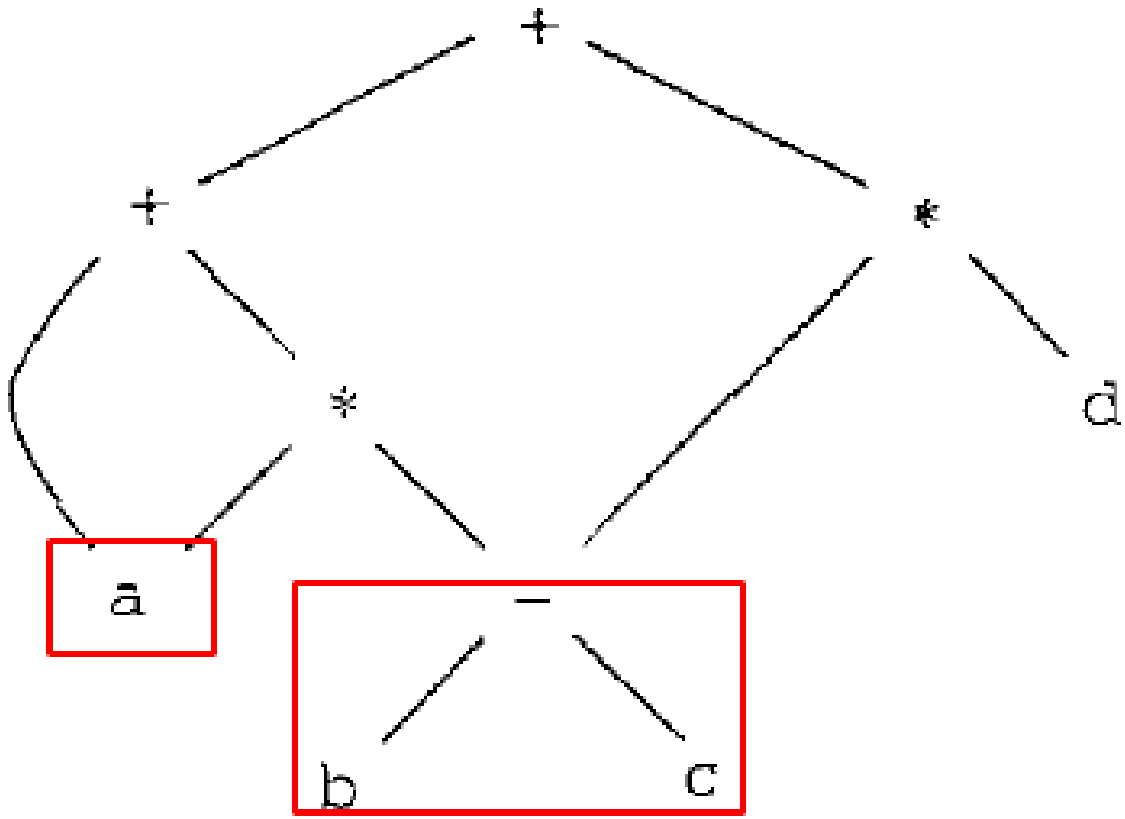


图 (抽象语法树)、**三地址代码**、C 语言

表达式的有向无环图



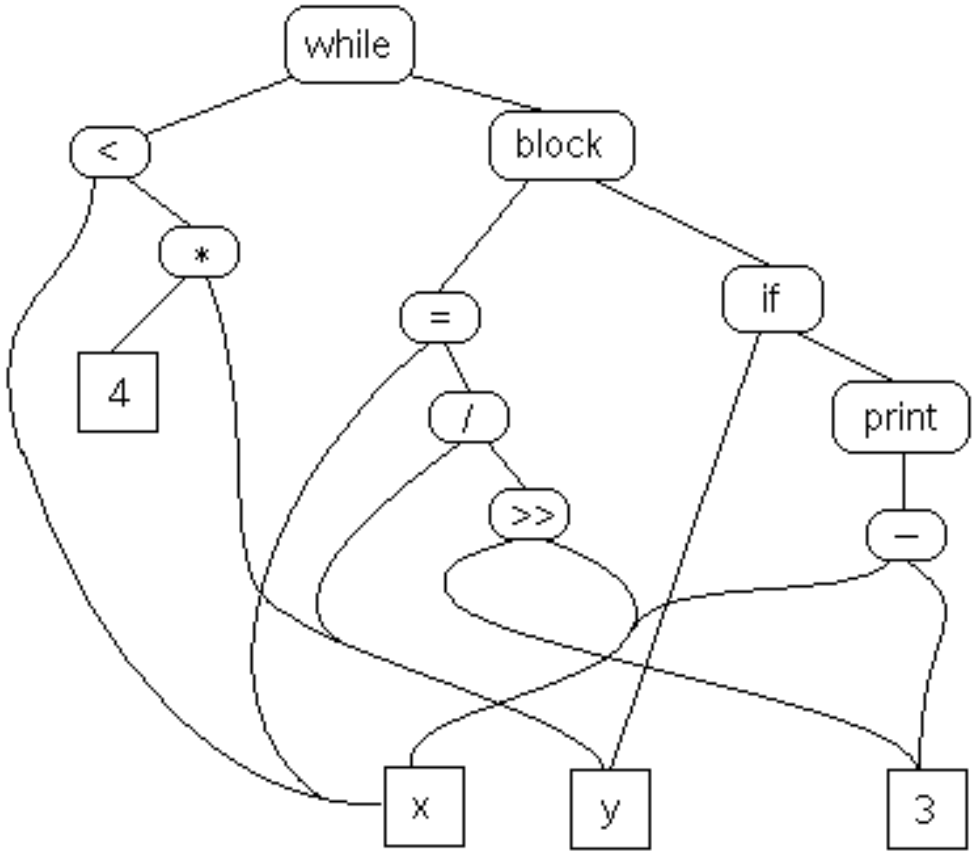
$$a + a * (b - c) + (b - c) * d$$

产生式		语义规则
1)	$E \to E_1 + T$	$E.node =$ **new** $Node('+', E_1.node, T.node)$
2)	$E \to E_1 - T$	$E.node =$ **new** $Node('-', E_1.node, T.node)$
3)	$E \to T$	$E.node = T.node$
	$T \to T_1 * F$	$T.node =$ **new** $Node('*', T_1.node, F.node)$
4)	$T \to (\ E\ )$	$T.node = E.node$
5)	$T \to$ **id**	$T.node =$ **new** $Leaf(\mathbf{id}, \mathbf{id}.entry)$
6)	$T \to$ **num**	$T.node =$ **new** $Leaf(\mathbf{num}, \mathbf{num}.val)$

产生式		语义规则
1)	$E \to E_1 + T$	$E.node =$ **new** $Node('+', E_1.node, T.node)$
2)	$E \to E_1 - T$	$E.node =$ **new** $Node('-', E_1.node, T.node)$
3)	$E \to T$	$E.node = T.node$
	$T \to T_1 * F$	$T.node =$ **new** $Node('*', T_1.node, F.node)$
4)	$T \to (\ E\ )$	$T.node = E.node$
5)	$T \to$ **id**	$T.node =$ **new** $Leaf(\mathbf{id}, \mathbf{id}.entry)$
6)	$T \to$ **num**	$T.node =$ **new** $Leaf(\mathbf{num}, \mathbf{num}.val)$

在创建节点之前，先判断是否已存在 (哈希表)

```
while (x < 4 * y) {
    x = y / 3 >> x;
    if (y) print x - 3;
}
```



Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y \ \mathbf{op} \ z \tag{1}$$

$$x = \mathbf{op} \ y \tag{2}$$

$$x = y \tag{3}$$

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y \ \mathbf{op} \ z \quad (1)$$

$$x = \mathbf{op} \ y \quad (2)$$

$$x = y \quad (3)$$

$$\mathbf{goto} \ L \quad (4)$$

$$\mathbf{if} \ x \ \mathbf{goto} \ L \quad (5)$$

$$\mathbf{if \ False} \ x \ \mathbf{goto} \ L \quad (6)$$

$$\mathbf{if} \ x \ \mathbf{relop} \ y \ \mathbf{goto} \ L \quad (7)$$

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$\textbf{param}\ x \tag{8}$$
$$\textbf{call}\ p, n \tag{9}$$
$$y = \textbf{call}\ p, n \tag{10}$$
$$\textbf{return}\ y \tag{11}$$

```
param x1
param x2
...
param xn
call p, n
```

$p(x_1, x_2, \ldots, x_n)$

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y[i] \tag{12}$$

$$x[i] = y \tag{13}$$

距离位置 y 处 i 个内存单元

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

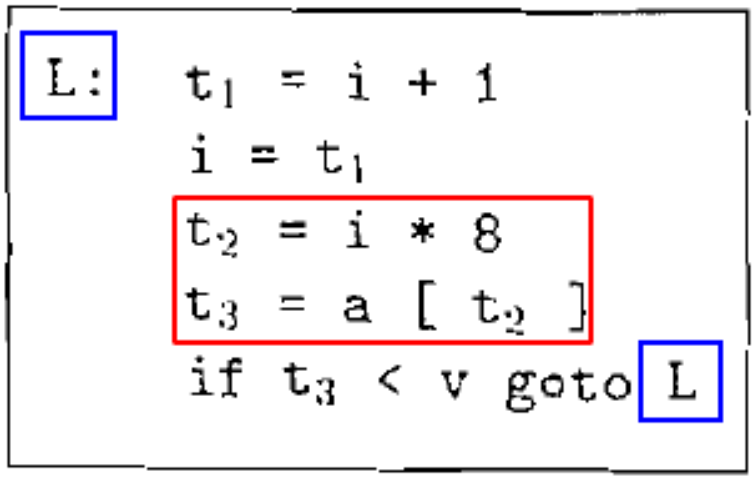
每个 **TAC** 指令**最多**包含三个操作数。

$$x = y[i] \tag{12}$$
$$x[i] = y \tag{13}$$

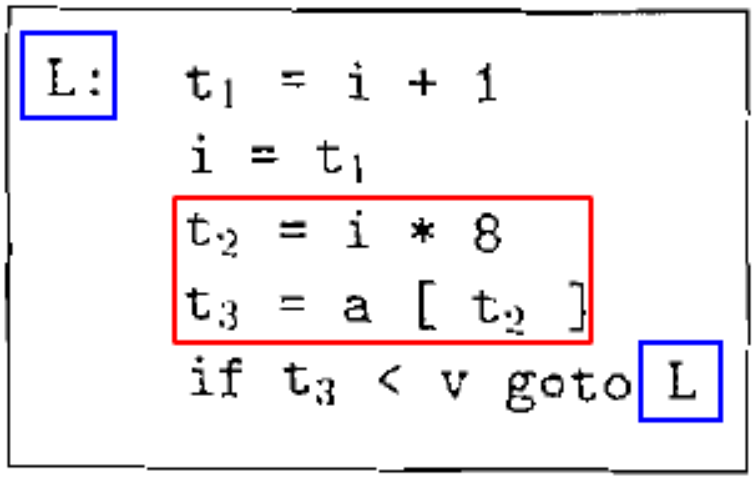
距离位置 y 处 i 个内存单元

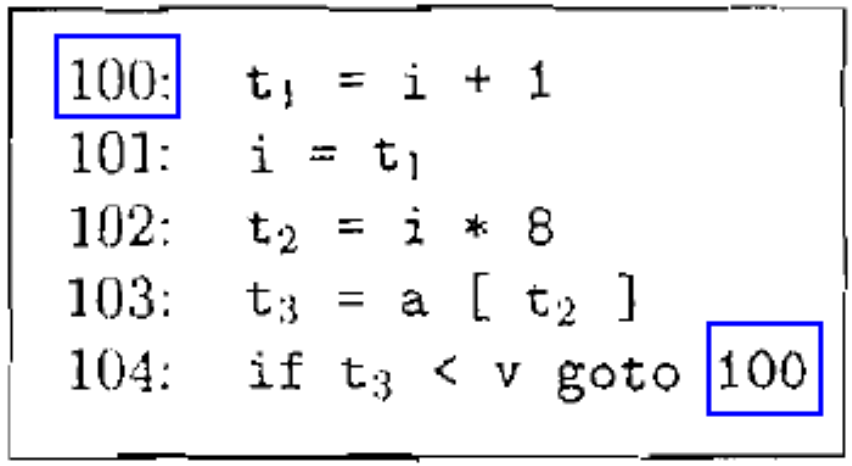
$$x = \&y \tag{14}$$
$$x = *y \tag{15}$$
$$*x = y \tag{16}$$

```
do i = i + 1; while (a[i] < v);
```



```
do i = i + 1; while (a[i] < v);
```





三地址代码的四元式表示

Definition (四元式 (Quadruple))

一个四元式包含四个字段, 分别为 op、arg_1、arg_2 与 $result$。

三地址代码的四元式表示

Definition (四元式 (Quadruple))

一个四元式包含四个字段, 分别为 op、arg_1、arg_2 与 $result$。

$$a + a * (b - c) + (b - c) * d$$

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

	op	arg_1	arg_2	result
0	minus	c		t_1
1	*	b	t_1	t_2
2	minus	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
		...		

$$x = y[i] \qquad =[\,] \quad y \quad i \quad x$$

$$x[i] = y \qquad [\,]= \quad i \quad y \quad x$$

$$x = y[i] \qquad =[\,] \quad y \quad i \quad x$$

$$x[i] = y \qquad [\,]= \quad i \quad y \quad x$$

$$x = \&y$$

$$x = *y$$

$$*x = y$$

$$x = y[i] \qquad =[\,] \quad y \quad i \quad x$$

$$x[i] = y \qquad [\,]= \quad i \quad y \quad x$$

$$x = \&y \qquad =\& \quad y \quad x$$

$$x = *y \qquad =* \quad y \quad x$$

$$*x = y \qquad *= \quad y \quad x$$

表达式的中间代码翻译

产生式	语义规则
$S \rightarrow \mathbf{id} = E\ ;$	$S.code = E.code \ \|\|$ $gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \ \|\| \ E_2.code \ \|\|$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ - E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \ \|\|$ $gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

符号表条目

综合属性 $E.code$ 与 $E.addr$

产生式	语义规则
$S \rightarrow \mathbf{id} = E\ ;$	$S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ - E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

符号表条目

```
t1 = minus c
t2 = b + t1
a = t2
```

$$a = b + -c$$

表达式的中间代码翻译 (增量式)

S	$\rightarrow$	**id** = E ;	{ $gen($ $top.get($**id**$.lexeme)$ $'='$ $E.addr)$; }
E	$\rightarrow$	E_1 + E_2	{ $E.addr =$ **new** $Temp()$; $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$; }
	|	$-$ E_1	{ $E.addr =$ **new** $Temp()$; $gen(E.addr\ '='\ '$**minus**$'\ E_1.addr)$; }
	|	(E_1)	{ $E.addr = E_1.addr$; }
	|	**id**	{ $E.addr = top.get($**id**$.lexeme)$; }

综合属性 $E.addr$

数组引用的中间代码翻译

声明：$\text{int } a[2][3]$

数组引用：$x = a[1][2];\ a[1][2] = x$

数组引用的中间代码翻译

声明：int $a[2][3]$

数组引用：$x = a[1][2];\ a[1][2] = x$

需要计算 $a[1][2]$ 的相对于**数组基地址** a 的**偏移地址**

数组引用的中间代码翻译

int $a[2][3]$

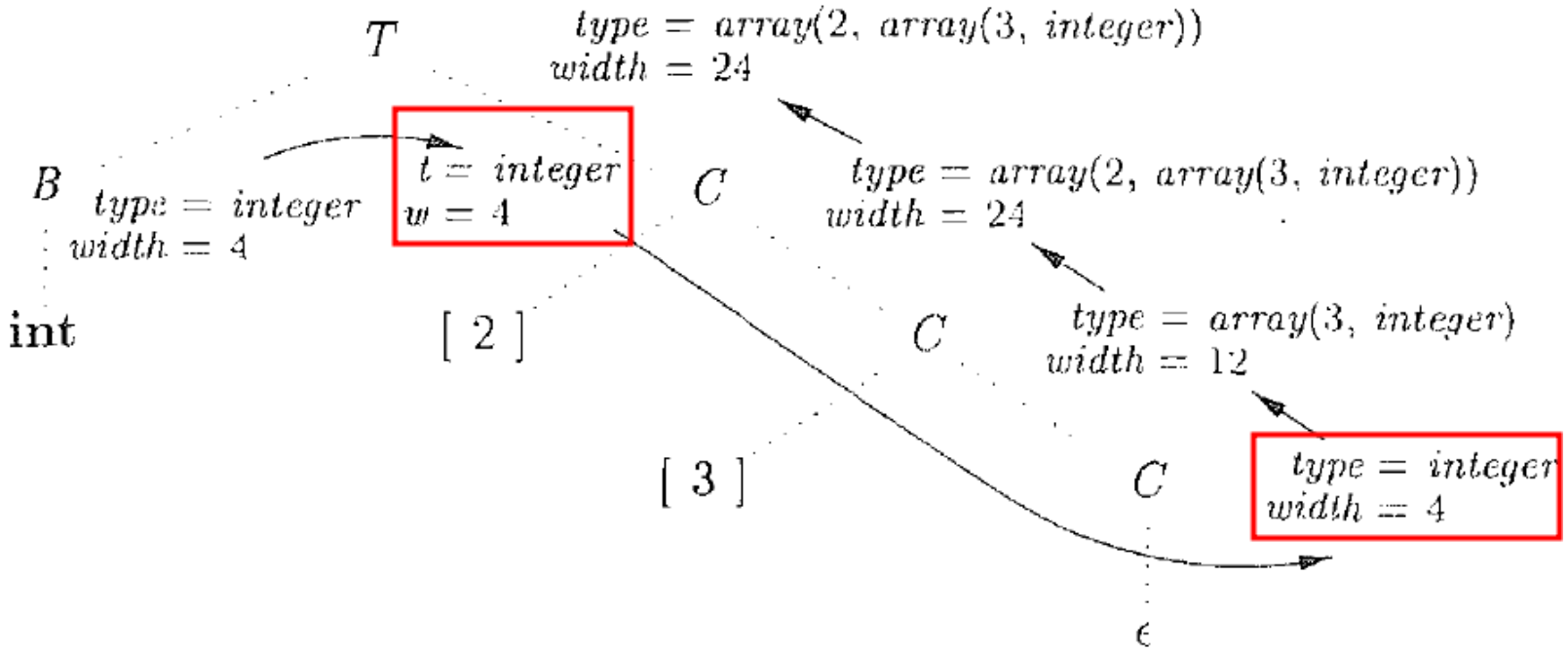


图 6-16　数组类型的语法制导翻译

数组类型声明

int $a[2][3]$

$$array(2, array(3, \text{integer}))$$

	类型	宽度
a	$array(2, array(3, \text{integer}))$	24
$a[i]$	$array(3, \text{integer})$	12
$a[i][j]$	integer	4

int $a[2][3]$

$$array(2, array(3, \text{integer}))$$

	类型	宽度
a	$array(2, array(3, \text{integer}))$	24
$a[i]$	$array(3, \text{integer})$	12
$a[i][j]$	integer	4

$$addr(a[1][2]) = base + 1 \times 12 + 2 \times 4$$

S	$\rightarrow$ **id** = E ;	{ *gen*(*top.get*(**id**.*lexeme*) '=' E.*addr*); }
	| L = E ;	{ *gen*(L.*array* .*base* '[' L.*addr* ']' '=' E.*addr*); }
E	$\rightarrow$ E_1 + E_2	{ E.*addr* = **new** *Temp* (); *gen*(E.*addr* '=' E_1.*addr* '+' E_2.*addr*); }
	| **id**	{ E.*addr* = *top.get*(**id**.*lexeme*); }
	| L	{ E.*addr* = **new** *Temp* (); *gen*(E.*addr* '=' L.*array*.*base* '[' L.*addr* ']'); }
L	$\rightarrow$ **id** [E]	{ L.*array* = *top.get*(**id**.*lexeme*); L.*type* = L.*array*.*type*.*elem*; L.*addr* = **new** *Temp* (); *gen*(L.*addr* '=' E.*addr* '*' L.*type*.*width*); }
	| L_1 [E]	{ L.*array* = L_1.*array*; L.*type* = L_1.*type*.*elem*; t = **new** *Temp* (); L.*addr* = **new** *Temp* (); *gen*(t '=' E.*addr* '*' L.*type*.*width*); *gen*(L.*addr* '=' L_1.*addr* '+' t); }

int $a[2][3]$

综合属性 $L.array.base$：数组基地址（即，数组名）

$$
\begin{array}{lcll}
S & \rightarrow & \mathbf{id} = E\ ; & \{\ gen(\ top.get(\mathbf{id}.lexeme)\ '='\ E.addr);\ \} \\
 & | & L = E\ ; & \{\ gen(L.array.base\ '['\ L.addr\ ']'\ '='\ E.addr); \\
E & \rightarrow & E_1 + E_2 & \{\ E.addr = \mathbf{new}\ Temp\,();\ gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \} \\
 & | & \mathbf{id} & \{\ E.addr = top.get(\mathbf{id}.lexeme);\ \} \\
 & | & L & \{\ E.addr = \mathbf{new}\ Temp\,();\ gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \}
\end{array}
$$

综合属性 $L.addr$：偏移地址

综合属性 $L.array$：数组名 id对应的符号表条目

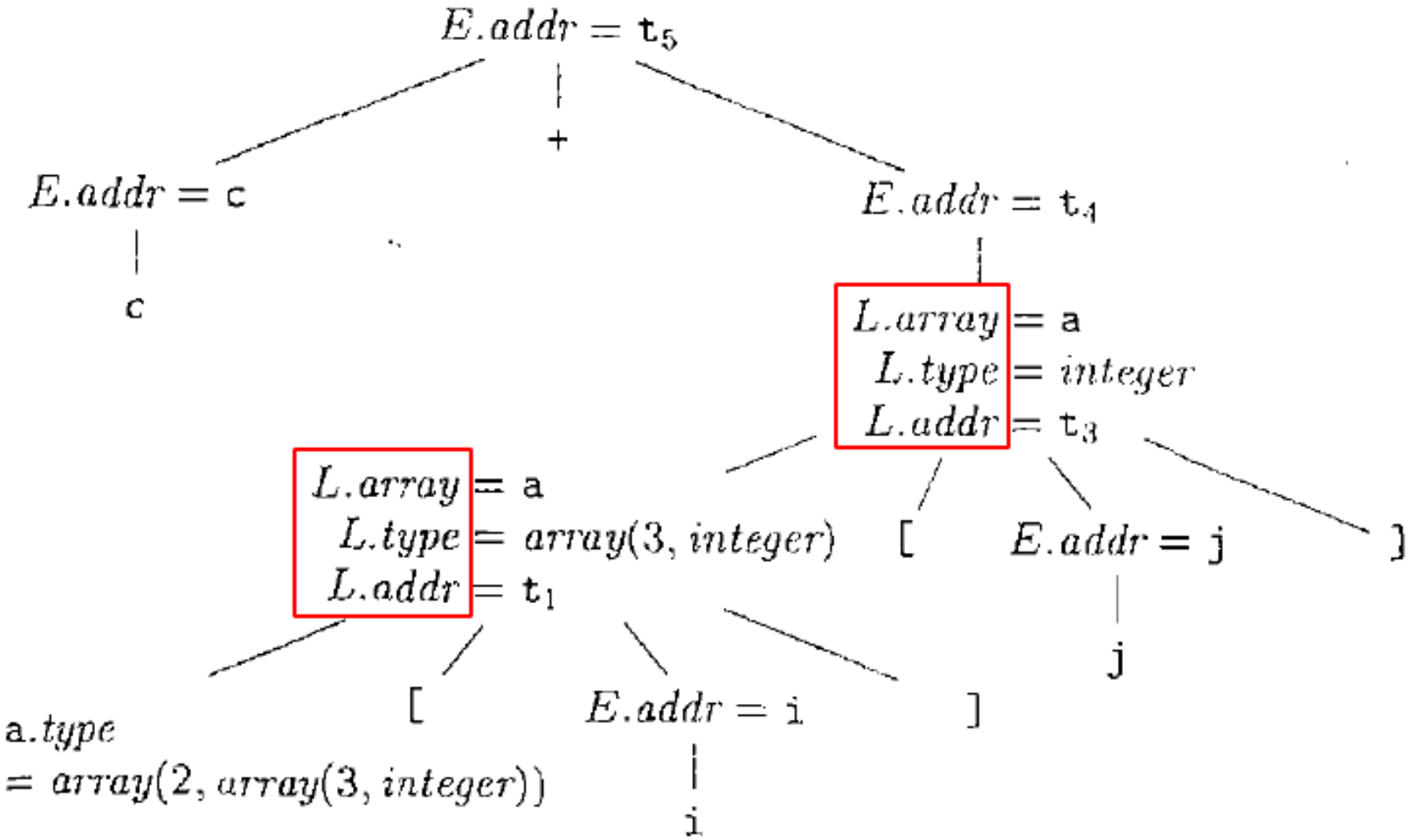
$$
\begin{aligned}
L \rightarrow\ & \mathbf{id}\ [\ E\ ] \quad \{\ L.array = top.get(\mathbf{id}.lexeme); \\
& \quad L.type = L.array.type.elem; \\
& \quad L.addr = \mathbf{new}\ Temp\,(); \\
& \quad gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \} \\
\mid\ & L_1\ [\ E\ ] \quad \{\ L.array = L_1.array; \\
& \quad L.type = L_1.type.elem; \\
& \quad t = \mathbf{new}\ Temp\,(); \\
& \quad L.addr = \mathbf{new}\ Temp\,(); \\
& \quad gen(t\ '='\ E.addr\ '*'\ L.type.width); \\
& \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{aligned}
$$

综合属性 $L.type$：(当前) 元素类型

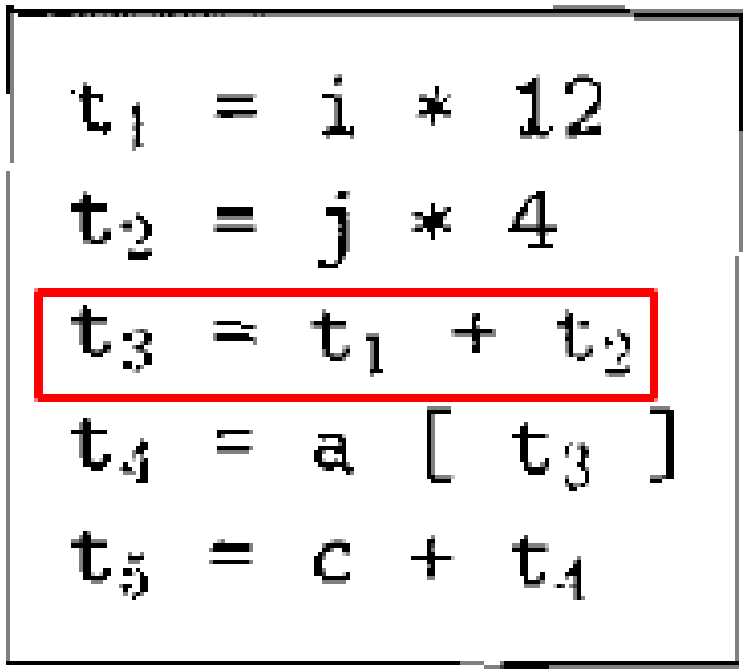
$$
\begin{aligned}
L \rightarrow\ & \mathbf{id}\ [\ E\ ] \quad \{\ L.array = top.get(\mathbf{id}.lexeme);\\
& \quad L.type = L.array.type.elem;\\
& \quad L.addr = \mathbf{new}\ Temp\,();\\
& \quad gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \}\\
\\
|\ \ & L_1\ [\ E\ ] \quad \{\ L.array = L_1.array;\\
& \quad L.type = L_1.type.elem;\\
& \quad t = \mathbf{new}\ Temp\,();\\
& \quad L.addr = \mathbf{new}\ Temp\,();\\
& \quad gen(t\ '='\ E.addr\ '*'\ L.type.width);\\
& \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{aligned}
$$

综合属性 $L.addr$：(当前) 偏移地址

$$
\begin{aligned}
L \rightarrow\ & \mathbf{id}\ [\ E\ ] \quad \{\ L.array = top.get(\mathbf{id}.lexeme);\\
& \quad L.type = L.array.type.elem;\\
& \quad L.addr = \mathbf{new}\ Temp\,();\\
& \quad gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \}\\
\mid\ & L_1\ [\ E\ ] \quad \{\ L.array = L_1.array;\\
& \quad L.type = L_1.type.elem;\\
& \quad t = \mathbf{new}\ Temp\,();\\
& \quad L.addr = \mathbf{new}\ Temp\,();\\
& \quad gen(t\ '='\ E.addr\ '*'\ L.type.width);\\
& \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{aligned}
$$


E.addr = t5
+
E.addr = c
c
E.addr = t4
L.array = a
L.type = integer
L.addr = t3
L.array = a
L.type = array(3, integer)
L.addr = t1
[
E.addr = j
]
j
a.type
= array(2, array(3, integer))
[
E.addr = i
]
i

int $a[2][3]$



int $a[2][3]$

控制流语句与布尔表达式的中间代码翻译

$$S \rightarrow \textbf{if}\ (B)\ S_1$$

$$S \rightarrow \textbf{if}\ (B)\ S_1\ \textbf{else}\ S_2$$

$$S \rightarrow \textbf{while}\ (B)\ S_1$$

控制流语句与布尔表达式的中间代码翻译

产生式	语义规则
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

继承属性 $S.next$: S 的下一条指令

$$P \rightarrow S \quad \left| \begin{array}{rcl} S.next & = & newlabel() \\ P.code & = & S.code \,||\, label(S.next) \end{array} \right.$$

$S.next$ 为语句 S 指明了 “跳出” S 的目标

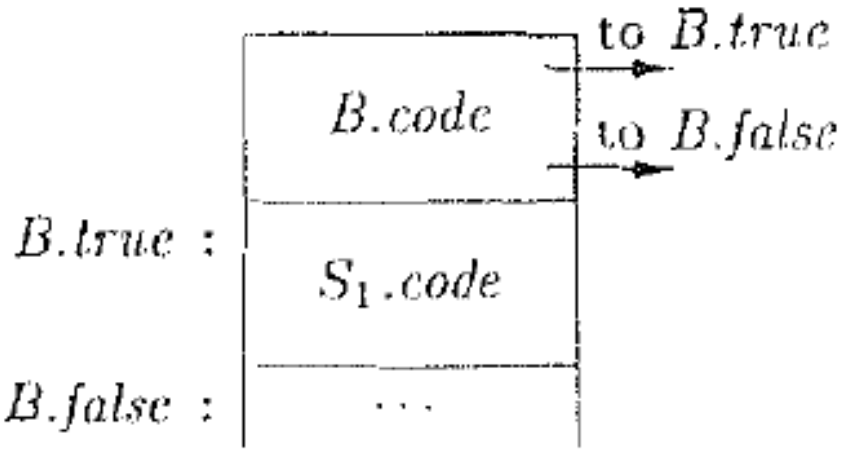
$$S \rightarrow \textbf{assign} \quad \Big| \quad S.code = \textbf{assign}.code$$

代表了表达式的翻译, 包括数组引用

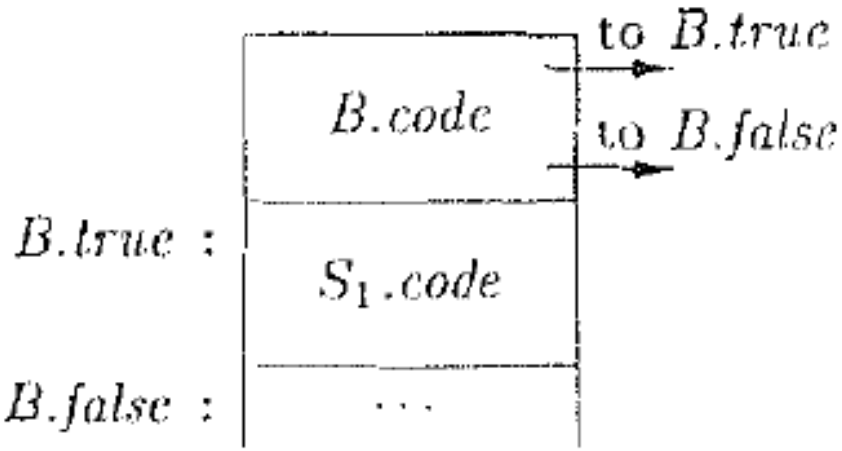
$$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$$

$$\left|\begin{array}{l} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{array}\right.$$

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$



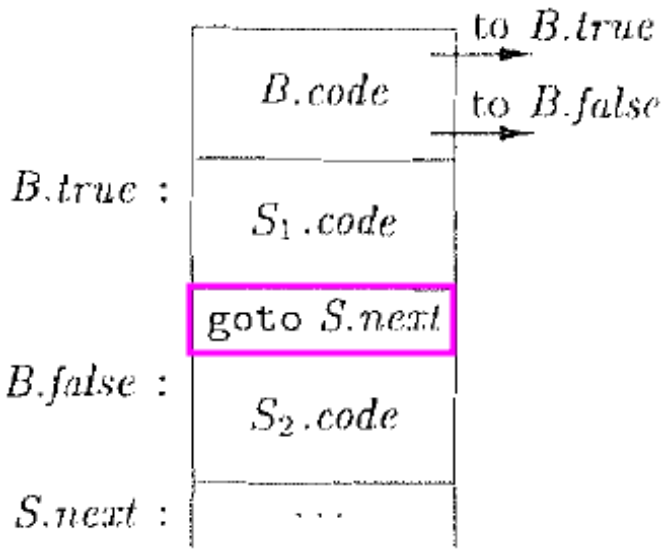
$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$



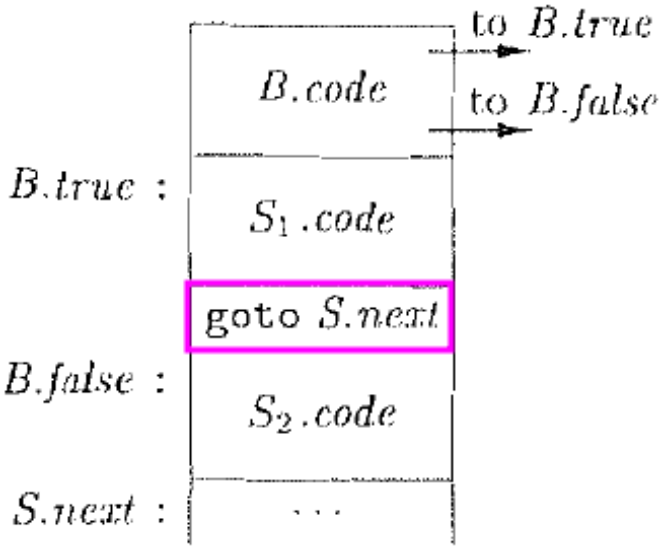

```
if (true)
  if (false) assign
```

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\quad \| \| label(B.true) \| \| S_1.code$ $\quad \| \| gen('\texttt{goto}'\ S.next)$ $\quad \| \| label(B.false) \| \| S_2.code$

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\quad \| \| label(B.true) \| \| S_1.code$ $\quad \| \| gen('\texttt{goto}'\ S.next)$ $\quad \| \| label(B.false) \| \| S_2.code$



$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\|\| label(B.true) \|\| S_1.code$ $\|\| gen('\texttt{goto}'\ S.next)$ $\|\| label(B.false) \|\| S_2.code$



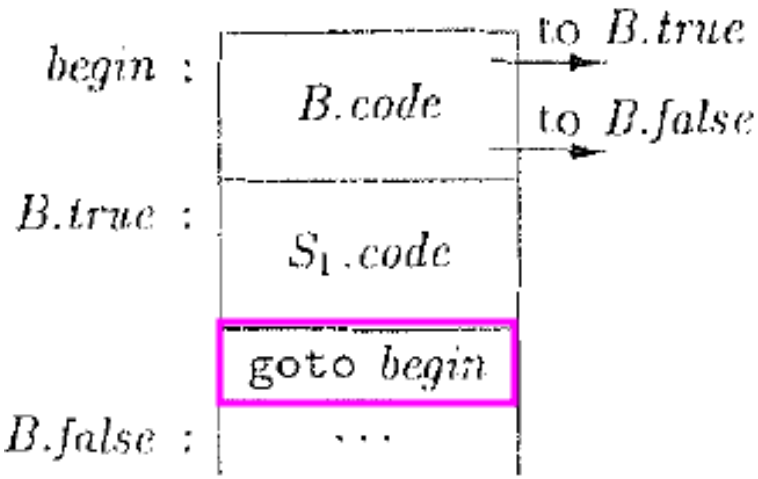

```
if (true)
  if (true) assign else assign
else
  assign
```

$S \rightarrow \textbf{while}\ (\ B\ )\ S_1$

$$
\begin{aligned}
&begin = newlabel() \\
&B.true = newlabel() \\
&B.false = S.next \\
&S_1.next = begin \\
&S.code = label(begin) \,||\, B.code \\
&\qquad ||\, label(B.true) \,||\, S_1.code \\
&\qquad ||\, gen(\texttt{'goto'}\ begin)
\end{aligned}
$$

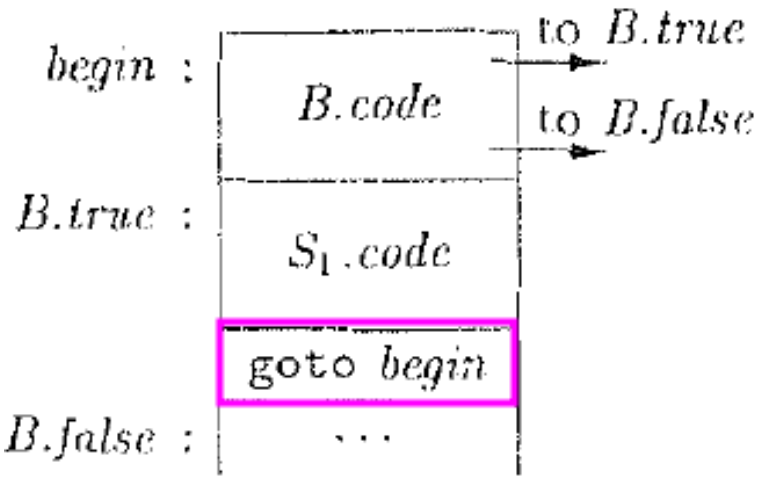
$S \rightarrow$ **while** (B) S_1

$begin = newlabel()$
$B.true = newlabel()$
$B.false = S.next$
$S_1.next = begin$
$S.code = label(begin) \,||\, B.code$
$\,||\, label(B.true) \,||\, S_1.code$
$\,||\, gen('\texttt{goto}'\ begin)$



$S \rightarrow$ **while** (B) S_1

$begin = newlabel()$
$B.true = newlabel()$
$B.false = S.next$
$S_1.next = begin$
$S.code = label(begin) \,||\, B.code$
$\,||\, label(B.true) \,||\, S_1.code$
$\,||\, gen(\text{'goto'}\ begin)$




```
while (true)
  if (false) assign else assign
```

$$S \rightarrow S_1\ S_2$$

$$\begin{array}{l} S_1.next = newlabel() \\ S_2.next = S.next \\ S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code \end{array}$$

$$
\begin{array}{lll}
S \rightarrow S_1\ S_2 & \boxed{S_1.next} = newlabel() \\
& \boxed{S_2.next} = S.next \\
& S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code
\end{array}
$$

```
if (true) assign else assign assign
```

产生式	语义规则
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

布尔表达式的中间代码翻译

产生式	语义规则
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$ $\mid\mid gen('\texttt{goto}'\ B.false)$
$B \rightarrow$ **true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$ **false**	$B.code = gen('\texttt{goto}'\ B.false)$

B	$\rightarrow$ true	$B.code = gen('\texttt{goto}'\ B.true)$
B	$\rightarrow$ false	$B.code = gen('\texttt{goto}'\ B.false)$

$B \rightarrow$	**true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$	**false**	$B.code = gen('\texttt{goto}'\ B.false)$

if (true) assign

$S \rightarrow$	**if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$

if (false) assign

$$B \rightarrow \; ! \; B_1$$

$$\begin{cases} B_1.true = B.false \\ B_1.false = B.true \\ B.code = B_1.code \end{cases}$$

$$B \rightarrow\ !\ B_1 \qquad \left| \begin{array}{l} B_1.true = B.false \\ B_1.false = B.true \\ B.code = B_1.code \end{array} \right.$$

if (!true) assign

$$S \rightarrow \textbf{if}\ (\ B\ )\ S_1 \qquad \left| \begin{array}{l} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{array} \right.$$

if (!false) assign

短路求值

$$
\begin{array}{lll}
B & \rightarrow & B_1 \mid\mid B_2 \\
\end{array}
\quad
\left|
\begin{array}{l}
B_1.true = B.true \\
B_1.false = newlabel() \\
B_2.true = B.true \\
B_2.false = B.false \\
B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code
\end{array}
\right.
$$

短路求值

$$
\begin{array}{lll}
B & \rightarrow & B_1 \ || \ B_2 \\
\end{array}
\quad
\left|
\begin{array}{l}
B_1.true = B.true \\
B_1.false = newlabel() \\
B_2.true = B.true \\
B_2.false = B.false \\
B.code = B_1.code \ || \ label(B_1.false) \ || \ B_2.code
\end{array}
\right.
$$

```
if (true || false) assign
```

$$
\begin{array}{lll}
S & \rightarrow & \textbf{if} \ (\ B \ ) \ S_1 \\
\end{array}
\quad
\left|
\begin{array}{l}
B.true = newlabel() \\
B.false = S_1.next = S.next \\
S.code = B.code \ || \ label(B.true) \ || \ S_1.code
\end{array}
\right.
$$

```
if (false || true) assign
```

短路求值

$$
\begin{array}{lll}
B \rightarrow B_1 \ \&\& \ B_2 & \left| \begin{array}{l} B_1.true = newlabel() \\ \boxed{B_1.false} = B.false \\ B_2.true = B.true \\ \boxed{B_2.false} = B.false \\ B.code = B_1.code \ || \ label(B_1.true) \ || \ B_2.code \end{array} \right. \\
\end{array}
$$

短路求值

$$
\begin{array}{lll}
B & \rightarrow & B_1 \ \&\& \ B_2 \\
\end{array}
\quad
\left|
\begin{array}{l}
B_1.true = newlabel() \\
B_1.false = B.false \\
B_2.true = B.true \\
B_2.false = B.false \\
B.code = B_1.code \ || \ label(B_1.true) \ || \ B_2.code
\end{array}
\right.
$$

if (true && false) assign

$$
\begin{array}{lll}
S & \rightarrow & \textbf{if} \ (\ B \ ) \ S_1 \\
\end{array}
\quad
\left|
\begin{array}{l}
B.true = newlabel() \\
B.false = S_1.next = S.next \\
S.code = B.code \ || \ label(B.true) \ || \ S_1.code
\end{array}
\right.
$$

if (false && true) assign

$$B \rightarrow E_1 \ \mathbf{rel} \ E_2 \quad \Big| \quad \begin{array}{l} B.code = E_1.code \ || \ E_2.code \\ \quad || \ gen(\text{'}\mathtt{if}\text{'} \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ \text{'}\mathtt{goto}\text{'} \ B.true) \\ \quad || \ gen(\text{'}\mathtt{goto}\text{'} \ B.false) \end{array}$$

```
if (x < 100 || x > 200 && x != y) x = 0;
```

```
        if x < 100 goto L2
        goto L3
L3:     if x > 200 goto L4
        goto L1
L4:     if x != y goto L2
        goto L1
L2:     x = 0
L1:
```

布尔表达式的作用：**布尔值** *vs.* **控制流跳转**

$$S \rightarrow \textbf{id} = E\ ; \mid \textbf{if}\ (E)\ S \mid \textbf{while}\ (E)\ S \mid S\ S$$

$$E \rightarrow E \mid\mid E \mid E \&\& E \mid E\ \textbf{rel}\ E \mid E + E \mid (E) \mid \textbf{id} \mid \textbf{true} \mid \textbf{false}$$

布尔表达式的作用: **布尔值** *vs.* **控制流跳转**

$$S \rightarrow \textbf{id} = E\ ; \mid \textbf{if}\ (E)\ S \mid \textbf{while}\ (E)\ S \mid S\ S$$

$$E \rightarrow E \,\|\, E \mid E \&\& E \mid E\ \textbf{rel}\ E \mid E + E \mid (E) \mid \textbf{id} \mid \textbf{true} \mid \textbf{false}$$

函数 $jump(t, f)$: 生成控制流代码

函数 $rvalue()$: 生成计算布尔值的代码, 并将结果存储在临时变量中

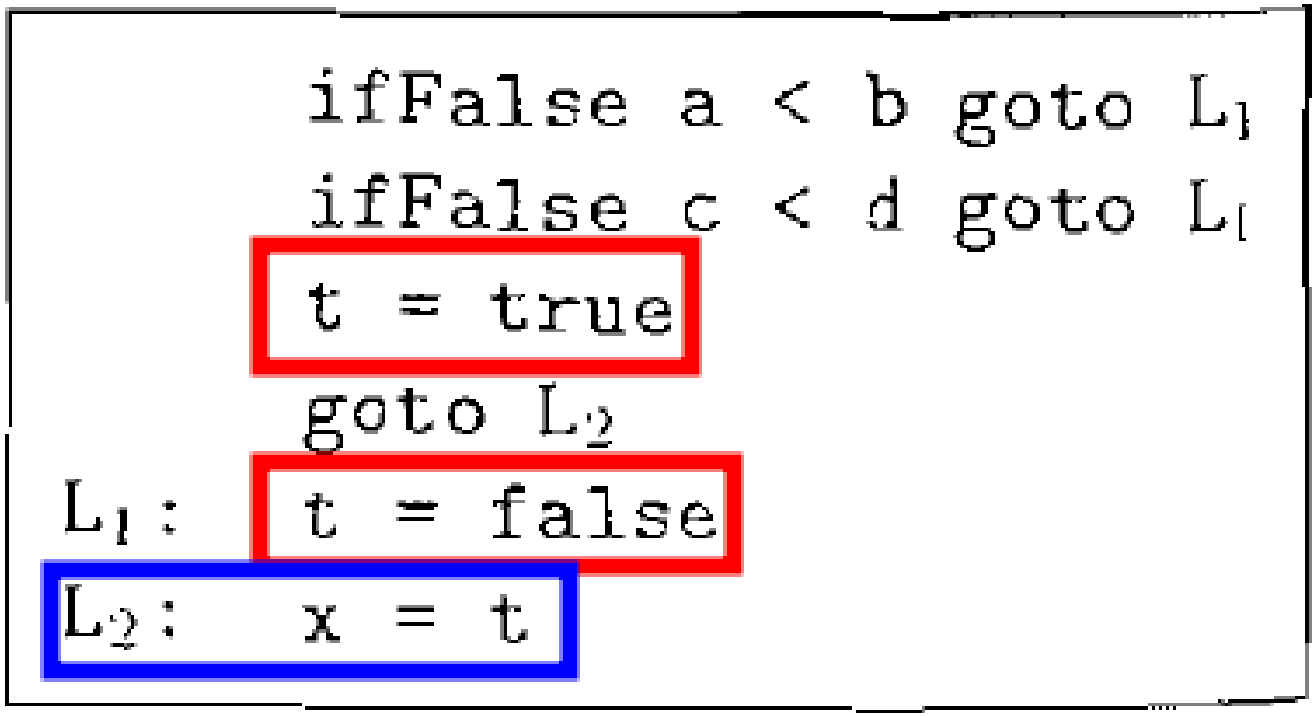
产生式	语义规则
$S \rightarrow \mathbf{id} = E\ ;$	$S.code = E.code \mid\mid gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid E_2.code \mid\mid gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ -E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (E_1)$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

符号表条目

$$E \rightarrow E_1 \&\& E_2$$

为 E 生成**跳转代码**, 在**真假出口处**将 **true** 或 **false** 存储到临时变量

x = a < b && c < d


ifFalse a < b goto L1
ifFalse c < d goto L1
t = true
goto L2
L1 : t = false
L2 : x = t

$$S \rightarrow \textbf{if}\ (\ B\ )\ S_1 \quad \left| \begin{array}{l} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{array} \right.$$

B 还不知道 $S.next$ 的指令地址，如何跳转？

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$

$$
\begin{array}{l}
B.true = newlabel() \\
B.false = S_1.next = S.next \\
S.code = B.code \,||\, label(B.true) \,||\, S_1.code
\end{array}
$$

B 还不知道 $S.next$ 的指令地址，如何跳转？

再扫描一遍中间代码，将标号替换成指令 (相对) 地址

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$

$$
\begin{array}{l}
B.true = newlabel() \\
B.false = S_1.next = S.next \\
S.code = B.code \,||\, label(B.true) \,||\, S_1.code
\end{array}
$$

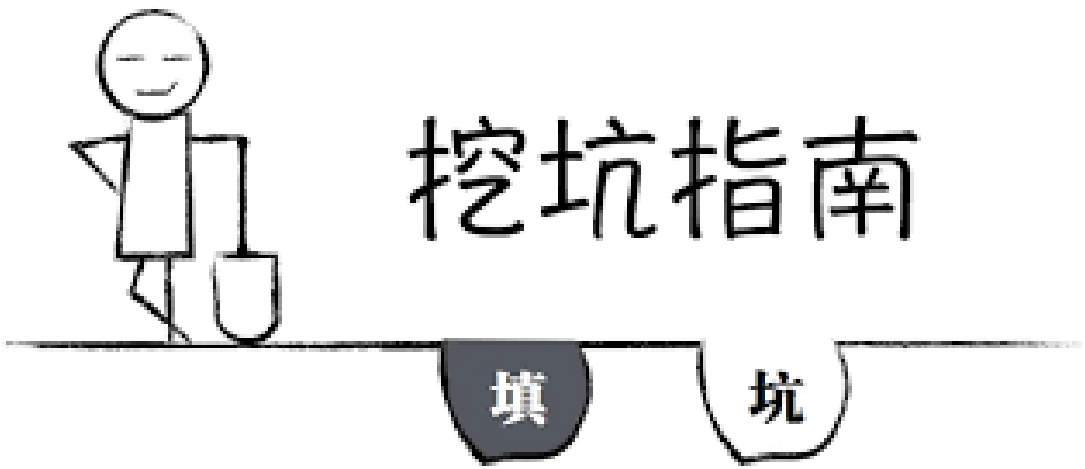
B 还不知道 $S.next$ 的指令地址，如何跳转？

再扫描一遍中间代码，将标号替换成指令 (相对) 地址

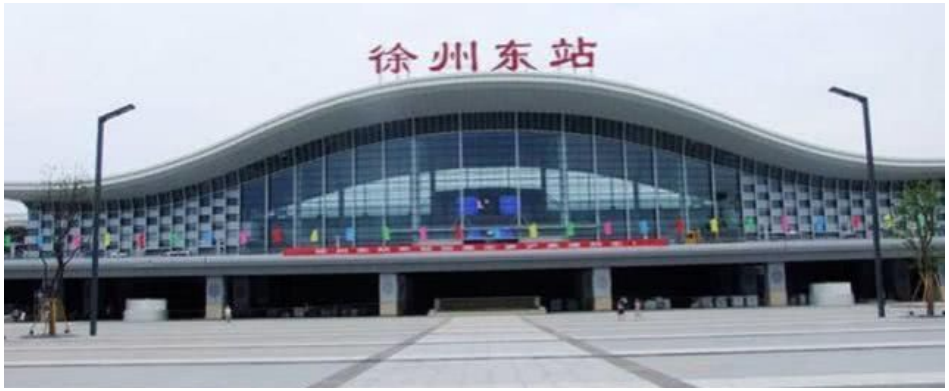
可否在生成中间代码的时候就填入指令地址？

回填 (Backpatching) 技术

回填 (Backpatching) 技术



子节点挖坑、祖先节点填坑


徐州东站

徐州观音机场
CARNOC.com

针对布尔表达式的回填技术

1)	$B \rightarrow B_1 \ \|\| \ M \ B_2$	{ $backpatch(B_1.falselist, M.instr)$; $B.truelist = merge(B_1.truelist, B_2.truelist)$; $B.falselist = B_2.falselist$; }
2)	$B \rightarrow B_1 \ \&\& \ M \ B_2$	{ $backpatch(B_1.truelist, M.instr)$; $B.truelist = B_2.truelist$; $B.falselist = merge(B_1.falselist, B_2.falselist)$; }
3)	$B \rightarrow \ ! \ B_1$	{ $B.truelist = B_1.falselist$; $B.falselist = B_1.truelist$; }
4)	$B \rightarrow (\ B_1 \ )$	{ $B.truelist = B_1.truelist$; $B.falselist = B_1.falselist$; }
5)	$B \rightarrow E_1 \ \mathbf{rel} \ E_2$	{ $B.truelist = makelist(nextinstr)$; $B.falselist = makelist(nextinstr + 1)$; $gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}\ \_')$; $gen('\texttt{goto}\ \_')$; }
6)	$B \rightarrow \mathbf{true}$	{ $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }
7)	$B \rightarrow \mathbf{false}$	{ $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }
8)	$M \rightarrow \epsilon$	{ $M.instr = nextinstr$; }

综合属性 $B.truelist$ 保存 需要跳转到 $B.true$ 的指令地址

6) $B \rightarrow$ **true** { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

7) $B \rightarrow$ **false** { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

综合属性 $B.falselist$ 保存 需要跳转到 $B.false$ 的指令地址

综合属性 $B.truelist$ 保存 需要跳转到 $B.true$ 的指令地址

6) $B \rightarrow$ **true** { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

7) $B \rightarrow$ **false** { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

综合属性 $B.falselist$ 保存 需要跳转到 $B.false$ 的指令地址

$B \rightarrow$ **true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$ **false**	$B.code = gen('\texttt{goto}'\ B.false)$

5) $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}\ \_')$;
$gen('\texttt{goto}\ \_')$; }

$B \rightarrow E_1 \ \mathbf{rel} \ E_2$	$B.code = E_1.code \ \|\| \ E_2.code$ $\|\| \ gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}' \ B.true)$ $\|\| \ gen('\texttt{goto}' \ B.false)$

3) $B \rightarrow\ !\ B_1$ { $B.truelist = B_1.falselist;$ $B.falselist = B_1.truelist;$ }

4) $B \rightarrow (\ B_1\ )$ { $B.truelist = B_1.truelist;$ $B.falselist = B_1.falselist;$ }

$$B \rightarrow\ !\ B_1 \quad \left| \begin{array}{l} B_1.true = B.false \\ B_1.false = B.true \\ B.code = B_1.code \end{array} \right.$$

2) $B \rightarrow B_1 \;\&\&\; M \; B_2$ { $backpatch(B_1.truelist, M.instr)$;
$B.truelist = B_2.truelist$;
$B.falselist = merge(B_1.falselist, B_2.falselist)$; }

8) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 \;\&\&\; B_2$

$B_1.true = newlabel()$
$B_1.false = B.false$
$B_2.true = B.true$
$B_2.false = B.false$
$B.code = B_1.code \;||\; label(B_1.true) \;||\; B_2.code$

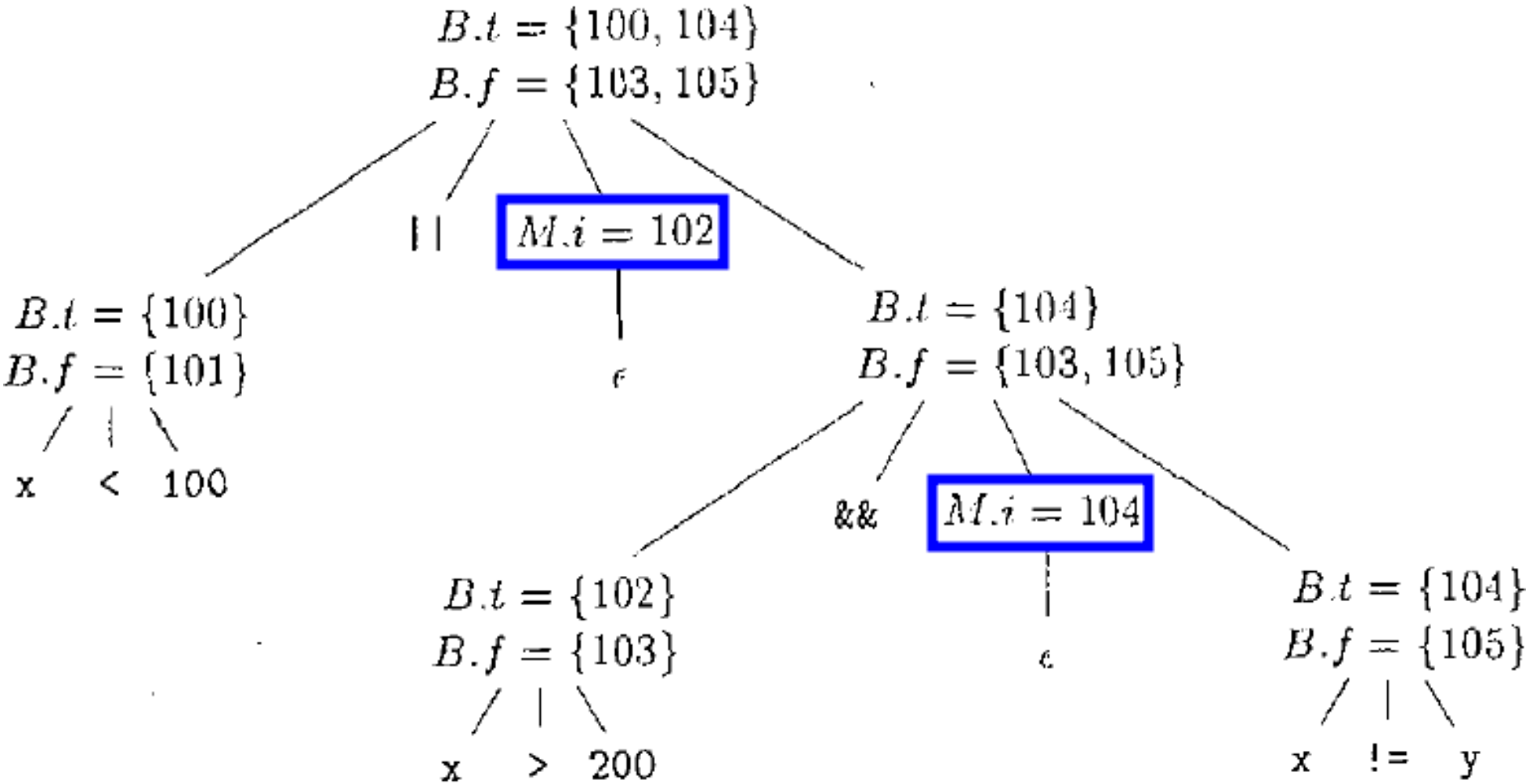
1) $B \rightarrow B_1 \mid\mid M\ B_2$ { $backpatch(B_1.falselist, M.instr)$;
$B.truelist = merge(B_1.truelist, B_2.truelist)$;
$B.falselist = B_2.falselist$; }

8) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 \mid\mid B_2$

$B_1.true = B.true$
$B_1.false = newlabel()$
$B_2.true = B.true$
$B_2.false = B.false$
$B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$

x < 100 || x > 200 && x != y


B.t = {100, 104}
B.f = {103, 105}
||
M.i = 102
ε
B.t = {100}
B.f = {101}
x < 100
B.t = {104}
B.f = {103, 105}
&&
M.i = 104
ε
B.t = {102}
B.f = {103}
x > 200
B.t = {104}
B.f = {105}
x != y

```
100:   if x < 100 goto _
101:   goto _
102:   if x > 200 goto 104
103:   goto _
104:   if x != y goto _
105:   goto _
```

a) 将 104 回填到指令 102 中之后

```
100:   if x < 100 goto _
101:   goto 102
102:   if x > 200 goto 104
103:   goto _
104:   if x != y goto _
105:   goto _
```

b) 将 102 回填到指令 101 中之后

$$S \rightarrow \textbf{if}\,(\,B\,)\,S \mid \textbf{if}\,(\,B\,)\,S\,\textbf{else}\,S \mid \textbf{while}\,(\,B\,)\,S \mid \{\,L\,\} \mid A\;;$$
$$L \rightarrow L\,S \mid S$$

1) $S \rightarrow \textbf{if}\,(\,B\,)\,M\,S_1\ \{\ backpatch(B.truelist,\ M.instr);$
$S.nextlist = merge(B.falselist,\ S_1.nextlist);\ \}$

6) $M \rightarrow \epsilon \qquad \{\ M.instr = nextinstr;\ \}$

1) $S \rightarrow \textbf{if}\ (\ B\ )\ M\ S_1\ \{\ backpatch(B.truelist,\ M.instr);$
$S.nextlist = merge(B.falselist,\ S_1.nextlist);\ \}$

6) $M \rightarrow \epsilon \qquad \{\ M.instr = nextinstr;\ \}$

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$

$$\begin{cases} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{cases}$$

$S \rightarrow$ **if** (B) M_1 S_1 N **else** M_2 S_2

{ $backpatch(B.truelist,\ M_1.instr)$;
$backpatch(B.falselist,\ M_2.instr)$;
$temp\ =\ merge(S_1.nextlist,\ N.nextlist)$;
$S.nextlist\ =\ merge(temp,\ S_2.nextlist)$; }

6) $M \rightarrow \epsilon$ { $M.instr\ =\ nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist\ =\ makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

$S \rightarrow$ **if** (B) M_1 S_1 N **else** M_2 S_2

{ $backpatch(B.truelist,\ M_1.instr)$;
$backpatch(B.falselist,\ M_2.instr)$;
$temp = merge(S_1.nextlist,\ N.nextlist)$;
$S.nextlist = merge(temp,\ S_2.nextlist)$; }

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist = makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

$S \rightarrow$ **if** (B) S_1 **else** S_2

$B.true = newlabel()$
$B.false = newlabel()$
$S_1.next = S_2.next = S.next$
$S.code = B.code$
$\quad || label(B.true) || S_1.code$
$\quad || gen('\texttt{goto}'\ S.next)$
$\quad || label(B.false) || S_2.code$

3) $S \rightarrow$ **while** M_1 (B) M_2 S_1

{ $backpatch(S_1.nextlist,\ M_1.instr)$;
$backpatch(B.truelist,\ M_2.instr)$;
$S.nextlist = B.falselist$;
$gen(\text{'goto'}\ M_1.instr)$; }

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

3) $S \rightarrow$ **while** M_1 (B) M_2 S_1

{ *backpatch*(S_1.*nextlist*, M_1.*instr*);
backpatch(B.*truelist*, M_2.*instr*);
S.*nextlist* = B.*falselist*;
gen('goto' M_1.*instr*); }

6) $M \rightarrow \epsilon$ { M.*instr* = *nextinstr*; }

$S \rightarrow$ **while** (B) S_1

begin = *newlabel*()
B.*true* = *newlabel*()
B.*false* = S.*next*
S_1.*next* = *begin*
S.*code* = *label*(*begin*) || B.*code*
|| *label*(B.*true*) || S_1.*code*
|| *gen*('goto' *begin*)

4) $S \rightarrow \{\ L\ \}$ $\{\ S.nextlist = L.nextlist;\ \}$

5) $S \rightarrow A\ ;$ $\{\ S.nextlist = \mathbf{null};\ \}$

6) $M \rightarrow \epsilon$ $\{\ M.instr = nextinstr;\ \}$

8) $L \rightarrow L_1\ M\ S$ $\{\ backpatch(L_1.nextlist,\ M.instr);$
$L.nextlist = S.nextlist;\ \}$

9) $L \rightarrow S$ $\{\ L.nextlist = S.nextlist;\ \}$

```
switch ( E ) {
        case V1: S1
        case V2: S2
                 ...
        case Vn-1: Sn-1
        default: Sn
}
```

非 C 语言语义 (break)

```
switch ( E ) {
        case V_1: S_1
        case V_2: S_2
                  ...
        case V_{n-1}: S_{n-1}
        default: S_n
}
```

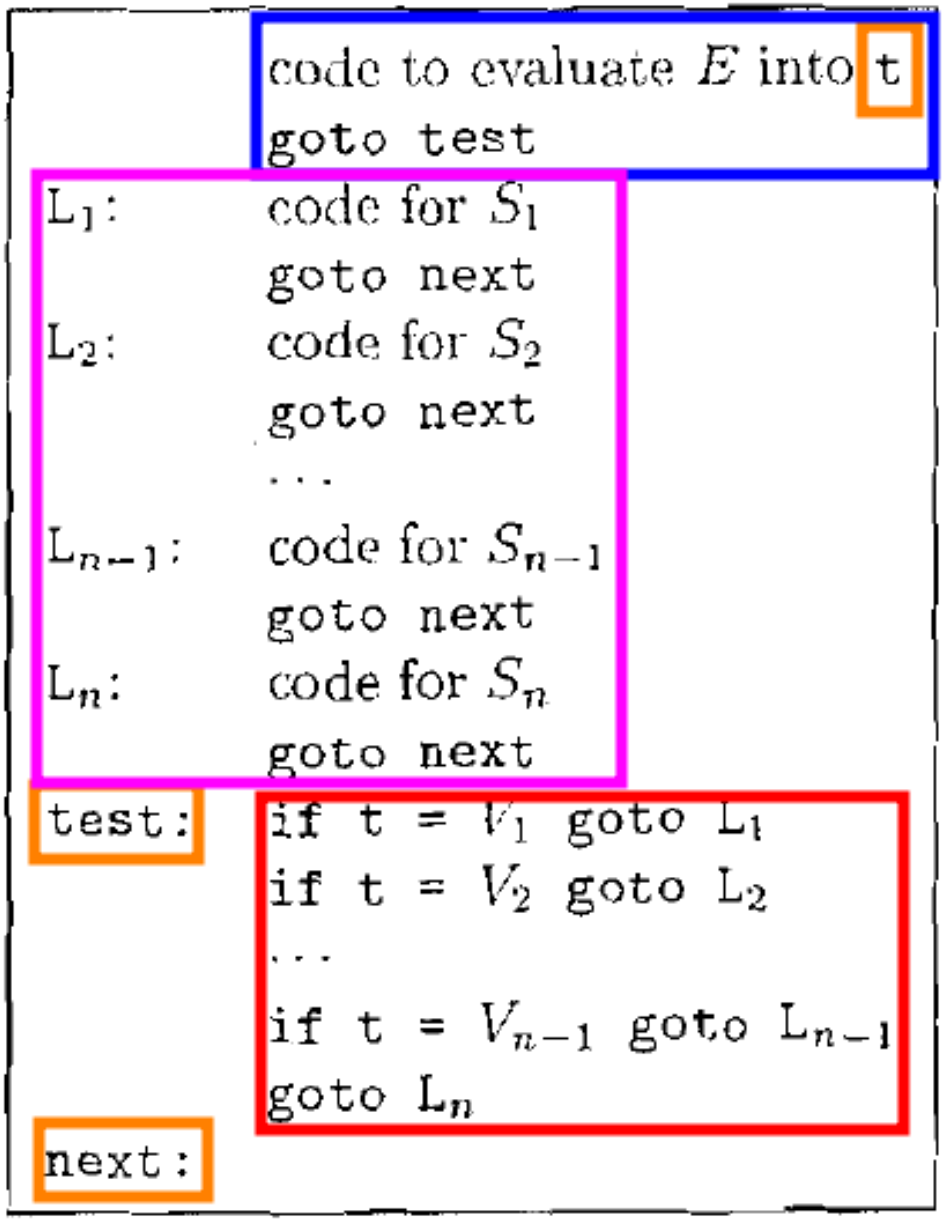
非 C 语言语义 (break)

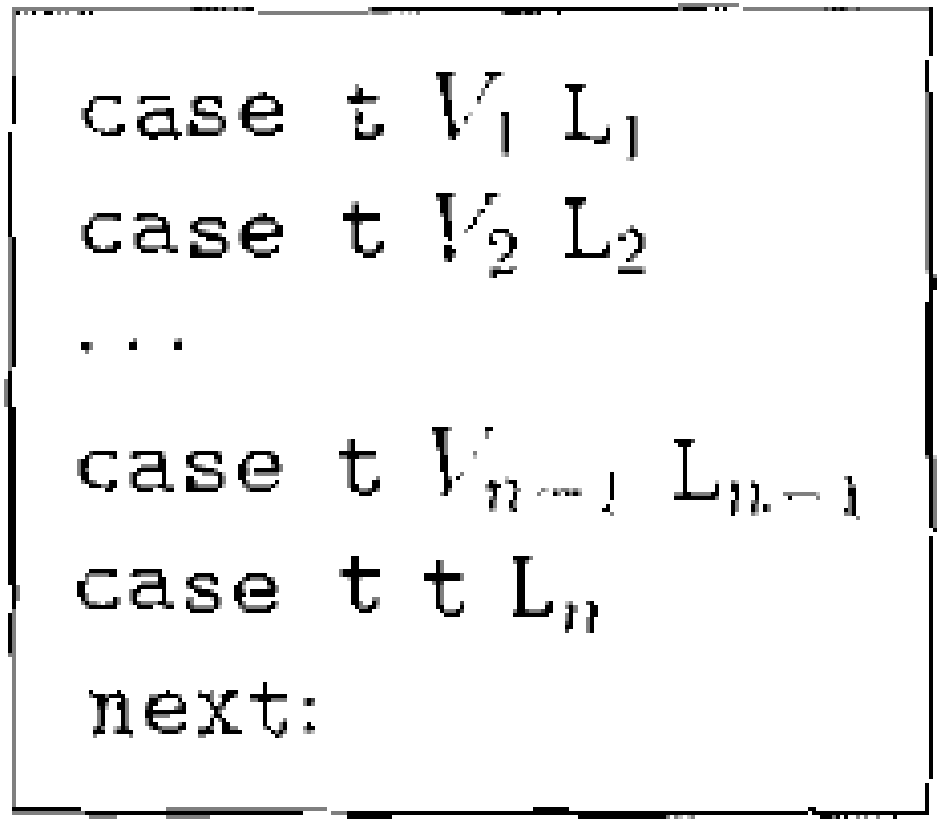
```
                code to evaluate E into t
                goto test
L_1:            code for S_1
                goto next
L_2:            code for S_2
                goto next
                ...
L_{n-1}:        code for S_{n-1}
                goto next
L_n:            code for S_n
                goto next
test:           if t = V_1 goto L_1
                if t = V_2 goto L_2
                ...
                if t = V_{n-1} goto L_{n-1}
                goto L_n
next:
```

```
switch ( E ) {
    case V1: S1
    case V2: S2
             ...
    case Vn-1: Sn-1
    default: Sn
}
```

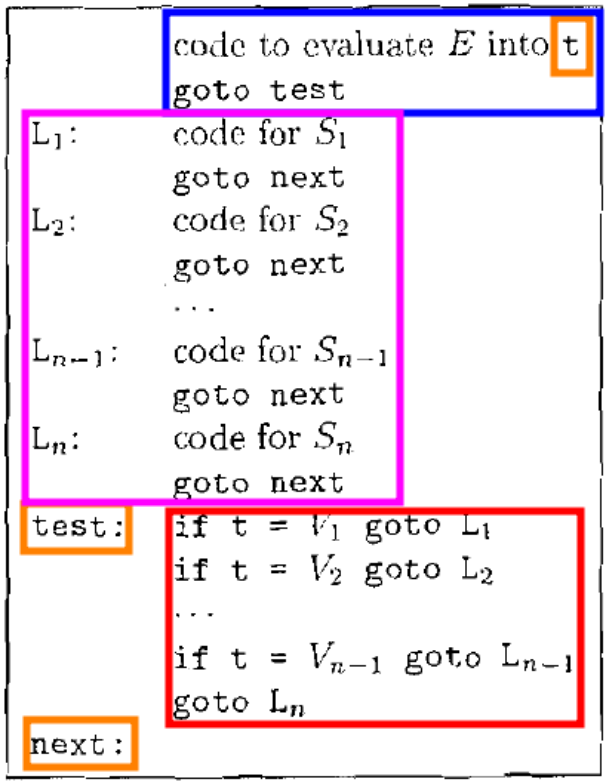
$V_i : L_i$ 队列

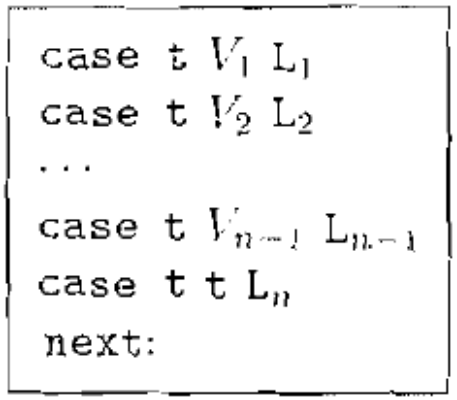
```
        code to evaluate E into t
        goto test
L1:     code for S1
        goto next
L2:     code for S2
        goto next
        ...
Ln-1:   code for Sn-1
        goto next
Ln:     code for Sn
        goto next
test:   if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
next:
```


code to evaluate E into t
goto test
L_1: code for S_1
goto next
L_2: code for S_2
goto next
...
L_{n-1}: code for S_{n-1}
goto next
L_n: code for S_n
goto next
test:
if t = V_1 goto L_1
if t = V_2 goto L_2
...
if t = V_{n-1} goto L_{n-1}
goto L_n
next:


case t V_1 L_1
case t V_2 L_2
...
case t V_{n-1} L_{n-1}
case t t L_n
next:

case 三地址代码


code to evaluate E into t
goto test
L_1: code for S_1
goto next
L_2: code for S_2
goto next
...
L_{n-1}: code for S_{n-1}
goto next
L_n: code for S_n
goto next
test: if t = V_1 goto L_1
if t = V_2 goto L_2
...
if t = V_{n-1} goto L_{n-1}
goto L_n
next:


case t V_1 L_1
case t V_2 L_2
...
case t V_{n-1} L_{n-1}
case t t L_n
next:

```
        code to evaluate E into t
        goto test
L1:     code for S1
        goto next
L2:     code for S2
        goto next
        ...
Ln-1:   code for Sn-1
        goto next
Ln:     code for Sn
        goto next
test:   if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
next:
```

```
case t V1 L1
case t V2 L2
...
case t Vn-1 Ln-1
case t t Ln
next:
```

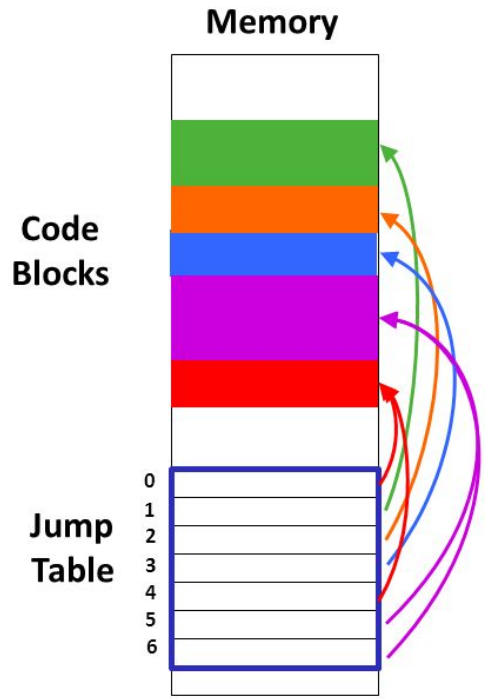

Jump Table Structure

C code:

```
switch(x) {
  case 1: <some code>
          break;
  case 2: <some code>
  case 3: <some code>
          break;
  case 5:
  case 6: <some code>
          break;
  default: <some code>
}
```

We can use the jump table when x <= 6:

```
if (x <= 6)
  target = JTab[x];
  goto *target;
else
  goto default;
```



Jump Table 优化

函数/过程的中间代码翻译

```
n = f(a[i])
```

```
1)   t1 = i * 4
2)   t2 = a [ t1 ]
3)   param t2
4)   t3 = call f, 1
5)   n = t3
```

新增文法以支持函数定义与调用

$$
\begin{array}{lcl}
D & \rightarrow & \textbf{define}\ T\ \textbf{id}\ (\ F\ )\ \{\ S\ \} \\
F & \rightarrow & \epsilon \mid T\ \textbf{id}\ ,\ F \\
S & \rightarrow & \textbf{return}\ E\ ; \\
E & \rightarrow & \textbf{id}\ (\ A\ ) \\
A & \rightarrow & \epsilon \mid E\ ,\ A
\end{array}
$$

函数定义

$$
\begin{array}{lcl}
D & \rightarrow & \textbf{define}\ T\ \textbf{id}\ (\ F\ )\ \{\ S\ \} \\
F & \rightarrow & \epsilon \mid T\ \textbf{id}\ ,\ F \\
S & \rightarrow & \textbf{return}\ E\ ;
\end{array}
$$

函数名 id 放入当前符号表，建立新的符号表，处理形参 F 与函数体 S

函数调用

$$E \rightarrow \boxed{\text{id (} A \text{)}}$$

$$A \rightarrow \epsilon \mid E \text{ , } A$$

```
param x1
param x2
...
param xn
call p, n
```

函数调用

S::=CALL id(Elist) { S.code := Elist.code
A
|| gencode("CALL", id.place, Elist.number) }

Elist::=Elist$_1$, E { Elist.code := E.code || Elist$_1$.code
逆序
|| gencode("PARAM", E.place);
Elist.number := Elist$_1$.number + 1 }

Elist::=E { Elist.code := E.code || gencode("PARAM", E.place);
Elist.number := 1 }

C 语言并未规定参数计算的顺序

函数调用

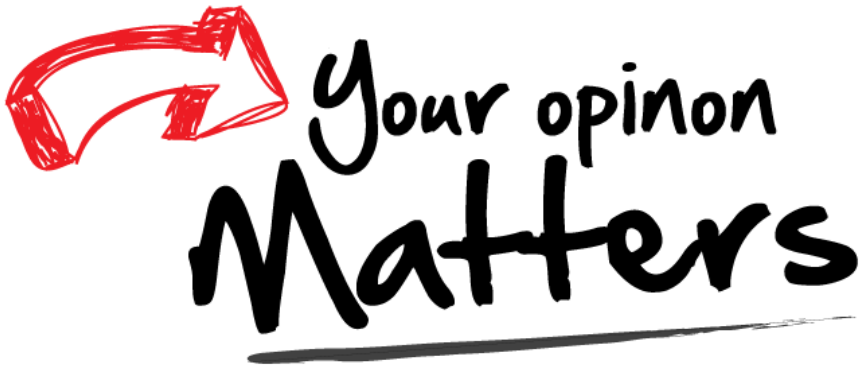
```
S::= CALL id(Elist)A
      { Count := 0; S.code := Elist.code;
        while NOT EmptyQ(q) do
        begin
           t := HeadQ(q);
           S.code := S.code || gencode("PARAM",t);
           DelQ(q); Count := Count + 1
        end;
        S.code := S.code || gencode("CALL",id.place,Coun
      }
```

逆序

```
Elist::= Elist1,E { Elist.code := E.code || Elist1.code;
                    EnterQ(E.place,q)}
Elist::= E        { Elist.code := E.code; CreateQ(q);
                    EnterQ(E.place,q)}
```

集中生成 **param** 指令, 代码更紧凑

Thank
You!



Office 926

hfwei@nju.edu.cn