中间代码生成
(3. Switch/Case 语句与过程调用的翻译)

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 24 日

```
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
}
```

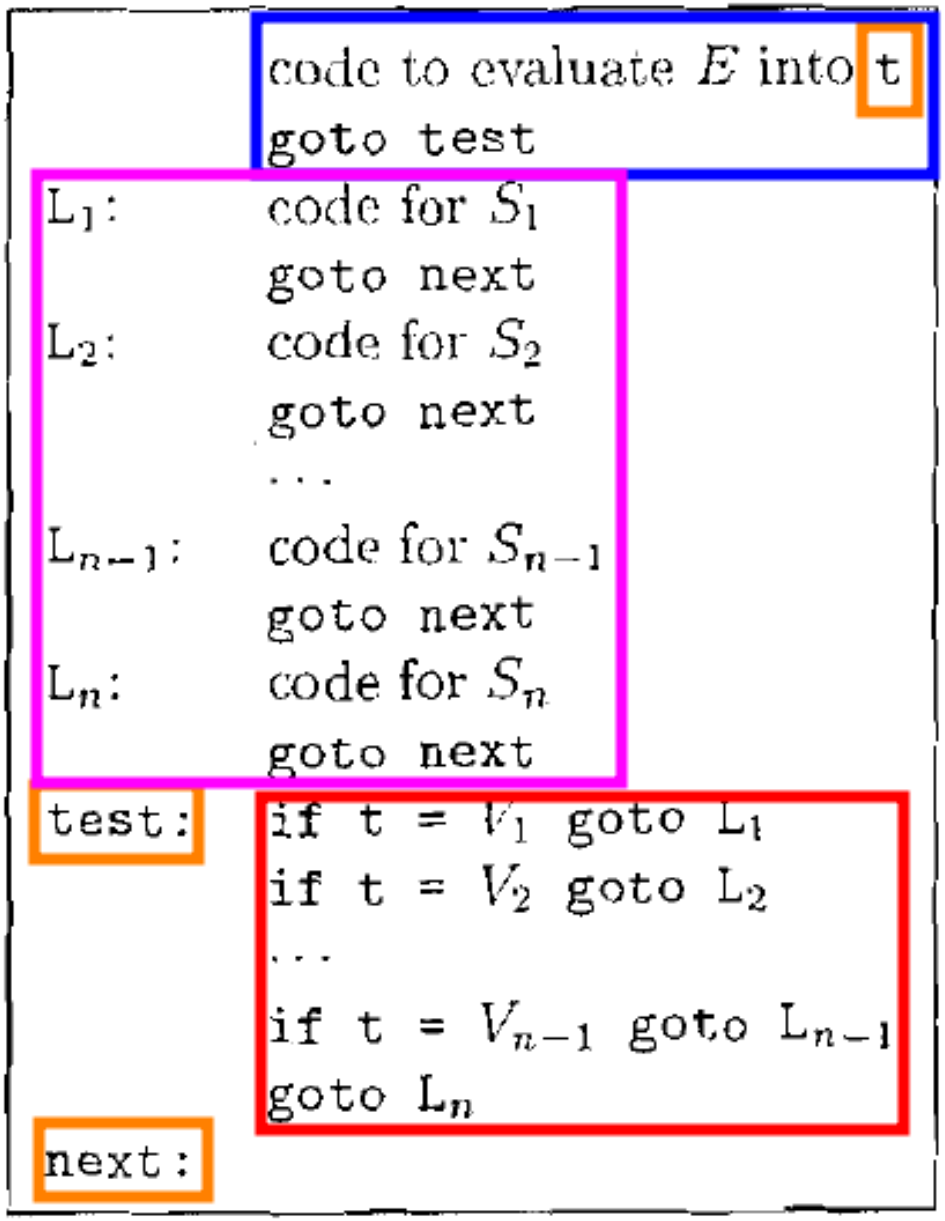
非 C 语言语义 (break)

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

$V_i : L_i$ 队列

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

case 三地址代码

```
        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:
```

```
        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:
```

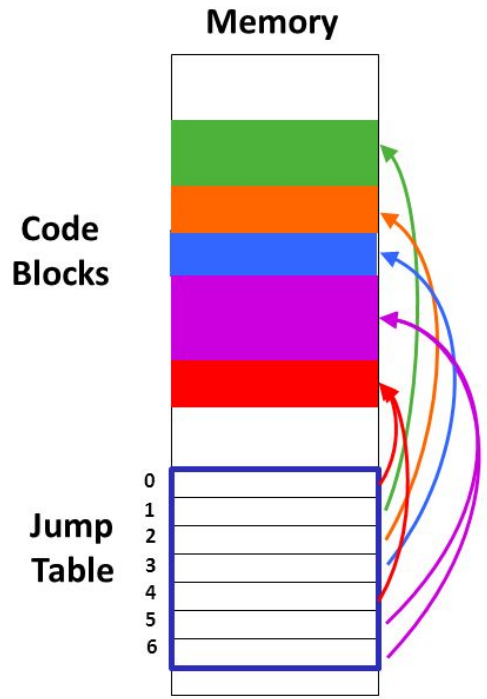

Jump Table Structure

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

$$\textbf{param}\ x \tag{8}$$

$$\textbf{call}\ p, n \tag{9}$$

$$y = \textbf{call}\ p, n \tag{10}$$

$$\textbf{return}\ y \tag{11}$$

```
param x_1
param x_2
...
param x_n
call p, n
```

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

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

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

$$A \rightarrow \epsilon \mid E \ , \ A$$

```
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

$$g(u, v, f(w)) \qquad g(u, f(v, w))$$

计算实参 x_1 的中间代码

param x_1

计算实参 x_2 的中间代码

param x_2

$\cdots$

计算实参 x_m 的中间代码

param x_n

call p, n

计算实参 x_1 的中间代码

param x_1

计算实参 x_2 的中间代码

param x_2

$\cdots$

计算实参 x_m 的中间代码

param x_n

call p, n

计算实参 x_1 的中间代码

计算实参 x_2 的中间代码

$\cdots$

计算实参 x_m 的中间代码

param x_1

param x_2

$\cdots$

param x_n

call p, n

函数调用

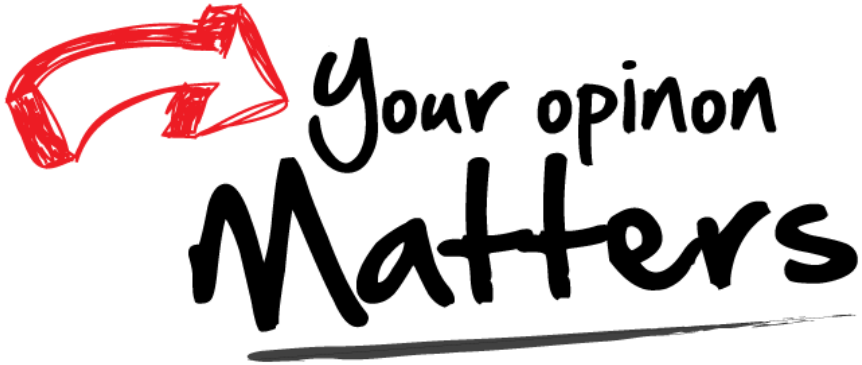
```
S::= CALL id(Elist)A
      { Count := 0; S.code := Elist.code;
        while NOT EmptyQ(q) do
        begin
           t := HeadQ(q);
           S.code := S.code || gencode("PARAM",t);
           DelQ(q); Count := Count + 1
        end;
        S.code := S.code || gencode("CALL",id.place,Coun
  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

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