编译原理 H12

PB20111686 黄瑞轩

7.4

引入新的非终结符 L，表示名字表，为其引入综合属性 *width* 表示宽度，引入综合属性 *type* 表示类型。

把图 7.5 的算法 D → **id**: T 那一行用下面的翻译方案来代替：

D → L : T { L.type = T.type;
L.width = T.width; }

L_1 → **id**, L_2 { *enter*(**id**.lexeme, L_1.type, offset);
offset = offset + L_1.width;
L_2.type = L_1.type;
L_2.width = L_1.width; }

| **id** { *enter*(**id**.lexeme, L_1.type, offset);
offset = offset + L_1.width; }

7.9

```
main:
.LFB0:
        pushq   %rbp                # 入栈
        movq    %rsp, %rbp
        jmp     .L2                 # 跳转到L2
.L5:
        movl    -4(%rbp), %eax      # j保存到中间寄存器
        movl    %eax, -8(%rbp)      # 中间寄存器保存到i（这两步是i=j操作）
.L2:                                # 开始条件判断
        cmpl    $0, -8(%rbp)        # 把i与0比较
        jne     .L3                 # 如果不等于，转到L3（||j的值不用算了，肯定是
1，短路）
        cmpl    $0, -4(%rbp)        # 如果等于，把j与0比较
        je      .L4                 # 如果等于0，则(i||j)为0，&&后面的不用算了（短
路）
```

```
.L3:
        cmpl    $5, -4(%rbp)     # 把j的值与5比较
        jg      .L5              # 如果大于（条件满足，计算到此说明前面条件都满
足，则进入循环）
.L4:
        movl    $0, %eax         # 返回值设为0
        popq    %rbp             # 出栈
        ret
```

7.10

先简单翻译一下：

```
main:
    pushl %ebp              # 入栈
    movl %esp,%ebp
    subl $8,%esp
    cmpl $0,-8(%ebp)        # 将j与0比较
    je .L2                  # 如果j=0，执行else（L2）
    incl -4(%ebp)           # j≠0，执行i++
    jmp .L3                 # 跳到L3（程序结束）
.L2:
.L4:
    cmpl $0,-4(%ebp)        # 比较i和0
    jne .L6                 # 如果i≠0，执行循环体L6
    jmp .L5                 # 如果i=0，循环结束，跳转到L5（程序结束）
.L6:
    incl -8(%ebp)           # j++
    jmp .L4                 # 返回循环
.L5:
.L3:
.L1:
    leave
    ret
```

（a）L4、L5、L6 是 while 循环使用的标号，L2、L3 是 else 部分的标号。while 语句在文法上属于 else 语句的一部分，所以 L4、L5、L6 在 L2 与 L3 之间，又没有合并相同的标号，所以导致了标号的重叠。

（b）L1 位于最外层，可能是为了代码提前返回准备的，本函数不存在提前返回的情况，所以没有引用此标号。

（c）简单翻译一下关键部分

```
main:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp
```

```
        cmpq    $0, -16(%rbp)    # j=0吗?
        je      .L4              # 等于，转到L4执行else
        addq    $1, -8(%rbp)     # 不等于，i++
        jmp     .L3              # 转到L3，函数执行结束
.L5:
        addq    $1, -16(%rbp)    # j++
.L4:
        cmpq    $0, -8(%rbp)     # 进入while条件判断：i=0吗?
        jne     .L5              # 不等于，执行while循环体（L5）
.L3:
        movl    $0, %eax
        popq    %rbp
        ret
```

L3：标识函数主体结束，进入返回值操作

L4：开始 while 的条件判断

L5：while 的循环体

没有 L1、L2 可能是因为编译器优化，合并了相同位置的标号。

7.17

（a）$A[i_1][i_2]\ldots[i_k]$ 的地址 = $base + i_1 \times w_1 + i_2 \times w_2 + \ldots + i_k \times w_k$

（b）为 L 引入新的综合属性 *dim* 表示包含数组的维度，为 0 表示简单变量；*getWidth*(p, k) 函数表示到符号表中取 k 维 p 数组元素的字节数。

$L \rightarrow L_1[E]$ { L.array = L_1.array;

L.dim = L_1.dim + 1;

w = *getWidth*(L.place, L.dim);

if (L.dim == 1) **begin**

L.offset = *newTemp*();

emit(L.offset, '=', E.place, '*', w);

end

else begin

t = *newTemp*();

L.offset = *newTemp*();

emit(t, '=', E.place, '*', w);

emit(L.offset, '=', L_1.offset, '+', t);

```
                end }
| id            { L.place = id.place;
                  L.offset = null;
                  L.dim = 0;
                  L.array = id.place; }
```