Chapter 7

作业

1.1

> 已知 C 语言的 for 语句文法:
>
> $$\mathbf{G(S)}: S \rightarrow for\ (E_1;\ E_2;\ E_3)\ S_1$$
>
> 1. 构建 `两遍扫描` 的 `属性文法`
> 2. 构建 `一遍扫描` 的 `翻译模式`

- 两遍扫描的属性文法

单独的 `while` 语句的 `两边翻译属性文法` 我们是很熟悉的, 所以这里先把 `for` 语句的文法进行一个简单的 `等效替换`:

$$S \rightarrow E_1;\ while\ (E_2)\ \{S_1;\ E_3\}$$

于是, 我们可以轻松的得到 `两遍扫描` 的 `属性文法`:

(`newlabel` 函数产生一个 `标号`, `gen` 函数将产生的 `三地址码` 放入某个语法单元的 `code` 综合属性中)

$$\begin{aligned}
S.begin &:= \text{newlabel} \\
E_2.true &:= \text{newlabel} \\
E_2.false &:= S.next \\
S_1.next &:= S.begin \\
S.code &:= E_1.code \\
&\quad || gen(S.begin\ \text{':'}) || E_2.code \\
&\quad || gen(E_2.true\ \text{':'}) || S_1.code || E_3.code \\
&\quad || gen(\text{'goto'}\ S.begin)
\end{aligned}$$

- 一遍扫描的翻译模式

$$S \to E_1;\ while\ (E_2)\ \{S_1;\ E_3\}$$

这一 等效文法, 为了利用回填技术, 进行 一遍扫描, 需要对 `while` 子句执行一些修改, 最终改为:

$$\begin{aligned} S &\to E_1;\ while\ (M_1\ E_2)\ \{M_2\ S_1;\ E_3\} \\ M &\to \epsilon \end{aligned}$$

所以, 自然而然地, 我们需要将原有的 `for` 文法改造为:

$$\begin{aligned} S &\to for\ (E_1;\ M_1\ E_2;\ M_2\ E_3)\ S_1 \\ M &\to \epsilon \end{aligned}$$

随后, 便可得到 一遍扫描 的 翻译模式:

$$\begin{aligned} S \to\ & for\ (M_1\ E_1;\ M_2\ E_2;\ M_3\ E_3)\ M_4\ S_1 \\ & \{ \\ & \quad \text{backpatch}(S.nextlist,\ M_1.quad) \\ & \quad \text{backpatch}(E_2.truelist,\ M_4.quad) \\ & \quad \text{backpatch}(S_1.nextlist,\ M_3.quad) \\ & \quad \text{emit}(\text{'j, -, -, '}M_3.quad) \\ & \quad \text{backpatch}(E_3.nextlist,\ M_2.quad) \\ & \quad E_1.nextlist := E_2.falselist \\ & \} \\ M \to\ & \epsilon \\ & \{\ M.quad = \text{nextquad}\ \} \end{aligned}$$

2.1

> 请将

```
A[i, j] := B[i, j] + C[A[k, l]] + D[i + j]
```

> 这个 赋值语句 翻译成 三地址码序列

首先, 这里题目做出了进一步的限定:

1. A, B 是 `10 * 20` 的 `二维数组`
2. C, D 是 `10` 长度的 `一维数组`
3. 数组的下标 `从 1 开始`
4. 每个数据的 `宽度` 为 `4`

随后, 我们可以生成下列 `三地址码序列`:

```
t11 = i * 20
t12 = t11 + j
t13 = A + 0 - (1 * 20 + 1) * 4 = A - 84
t14 = 4 * t12
t15 = t13[t14] // A[i, j]

t21 = i * 20
t22 = t21 + j
t23 = B + 0 - (1 * 20 + 1) * 4 = B - 84
t24 = 4 * t22
t25 = t23[t24] // B[i, j]

t31 = k * 20
t32 = t31 + l
t33 = A + 0 - (1 * 20 + 1) * 4 = A - 84
t34 = 4 * t32
t35 = t33[t34] // A[k, l]
t36 = C + 0 - (1) * 4 = C - 4
t37 = 4 * t35
t38 = t36[t37] // C[A[k, l]]

t41 = i + j
t42 = D + 0 - (1) * 4 = D - 4
t43 = 4 * t41
t44 = t42[t43] // D[i + j]

t1 = t25 + t38 // B[i, j] + C[A[k, l]]
t2 = t1 + t44 // B[i, j] + C[A[k, l]] + D[i + j]
```

```
t23[t24] = t2 // A[i, j] = B[i, j] + C[A[k, l]] + D[i + j]
```

2.2

> 请将

```
A or (B and not (C or D))
```

> 这个 布尔表达式 翻译成 四元式序列

结果如下:

```
format := (operation, argL, argR, result / label)
----------------------
100: (jnz, A, -, 0)
    // if A is false, exit
101: (j, _, _, 102)
    // A is true, continue
102: (jnz, B, -, 104)
    // if B is false, continue
103: (j, _, _, 0)
    // B is true, exit
104: (jnz, C, -, 103)
    // if C is false, exit
105: (j, _, _, 106)
    // C is true, continue
106: (jnz, D, -, 104)
    // D is false (exit)
107: (j, _, _, 0)
    // D is true (exit)
```

2.3

> 请将

```
while A < C and B < D do
    if A = 1 then
        C := C + 1
    else
        while A <= D do
            A := A + 2
        end
    end
end
```

> 这串由 `循环` 和 `分支` 控制流语句组成的 `语句` 翻译成 `四元式序列`

结果如下:

```
format := (operation, argL, argR, result / label)
----------------------
100: (j<, A, C, 102)
    // while A < C
101: (j, -, -, 114)
102: (j<, B, D, 104)
    // and B < D
103: (j, -, -, 114)
104: (j=, A, 1, 106)
    // if A = 1
105: (j, -, -, 109)
106: (+, C, 1, T)
    // T = C + 1
107: (:=, T, -, C)
    // C = T
108: (j, -, -, 100)
109: (j<=, A, D, 111)
    // while A <= D
```

```
110: (j, -, -, 100)
111: (+, A, 2, T)
    // T = A + 2
112: (:=, T, -, A)
    // A = T
113: (j, -, -, 109)
114: ......
----------------------
```