第五章作业第三次作业

5.4.4：为下面的产生式写出一个和例 5.19 类似的 L 属性 SDD。这里的每个产生式表示一个常见的 C 语言中那样的控制流结构。你可能需要生成一个三地址语句来跳转到某个标号 L，此时你可以生成语句 goto L

1) S → if (C) S1 else S2

2) S → do S1 while (C)

3) S → ‘{’ L ‘}’ ; L → LS | ε

Answer

1)

S → if（C）S1 else S2	L1 = new(); L2 = new(); C.true = L1; C.false = L2; S1.next = S.next; S2.next = S.next S.code = C.code || label || L1 || S1.code || goto S.next label || L2 || S2.code

2)

S → do S1 while（C）	L1 = new(); L2 = new(); C.true = L1; C.false = S.next S1.next = L2; S.code = label || L1 || S1.code || label || L2 || C.code

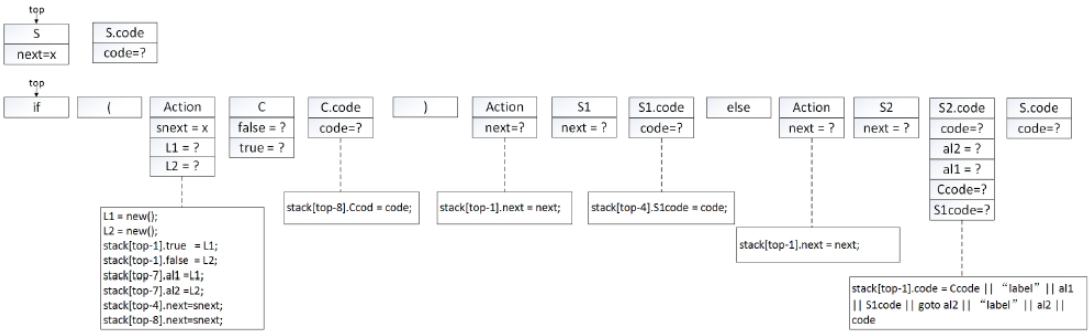
3)

S → ‘{’L‘}’	L.next = S.next; S.code = L.code;
L → L1 S	L2 = new(); L1.next = L2; S.next = L.next; L.code = L1.code || label || L2 || S.code;
L → ε	L.code = ‘’

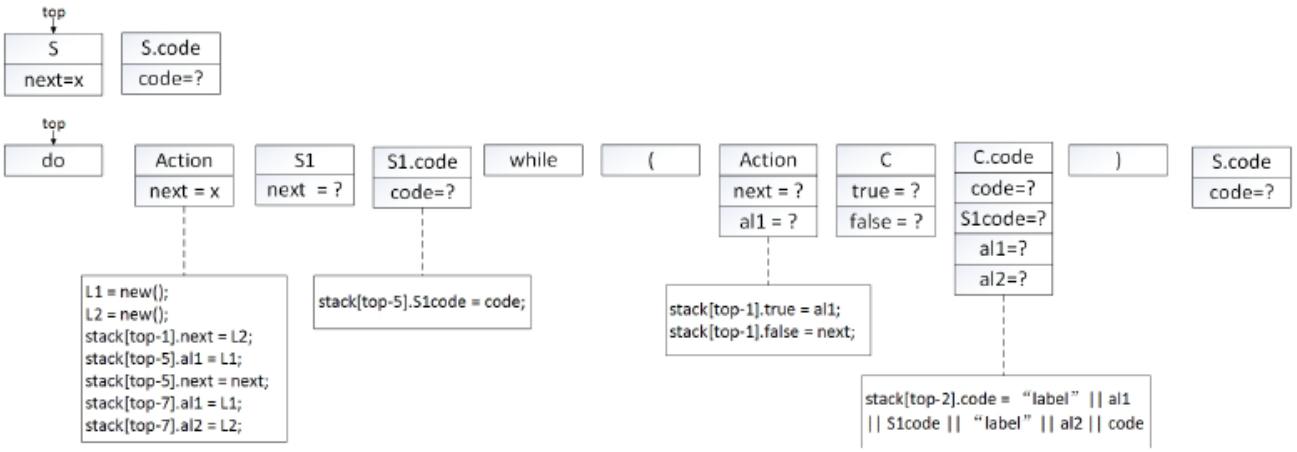
5.5.4：按照 5.5.3 节的风格，将 5.4.4 中得到的每个 SDD 和一个 LL 语法分析器一起实现，但是代码（或指向代码的指针）存放在栈中

Answer

1)



2)

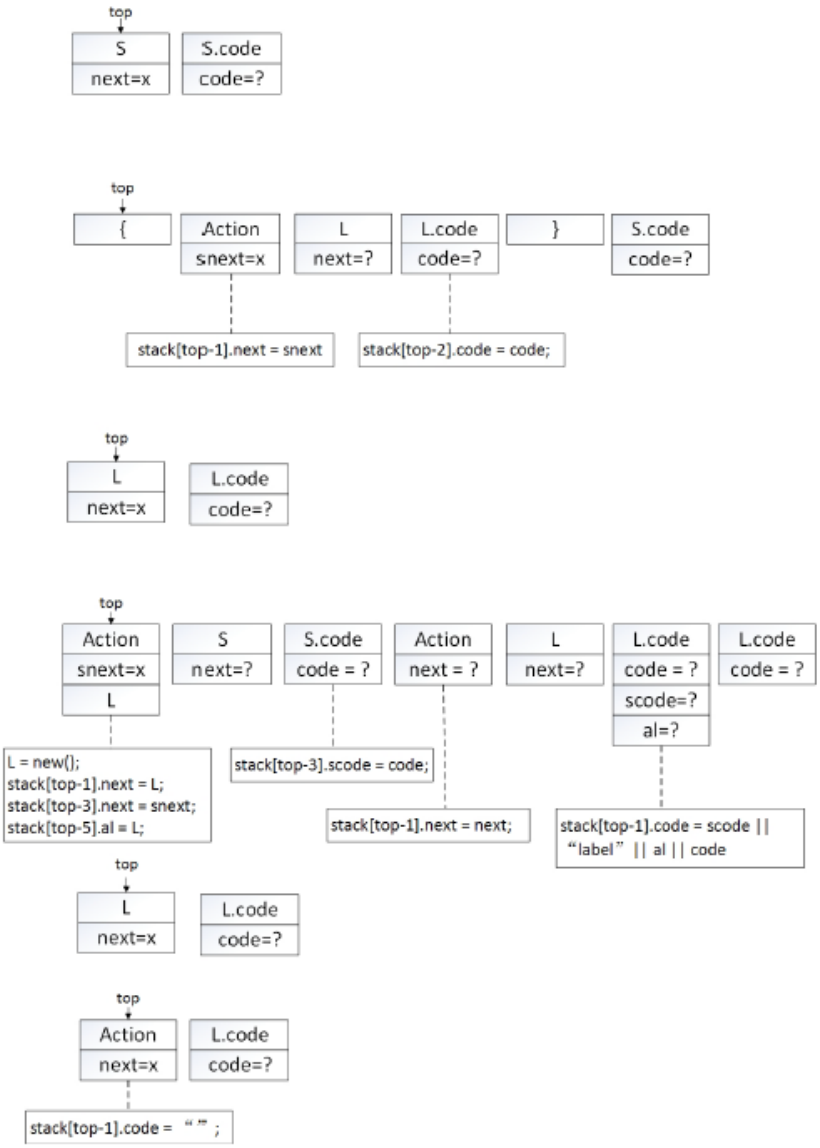


3) 消除左递归：

S → '{'L'}'

L → L'

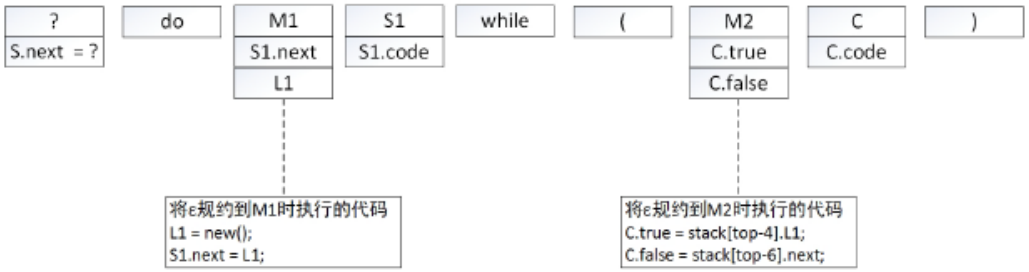
L' → SL' | ε



5.5.5：按照 5.5.4 节的风格，将 5.4.4 中得到的每个 SDD 和一个 LR 语法分析器一起实现

Answer

1)



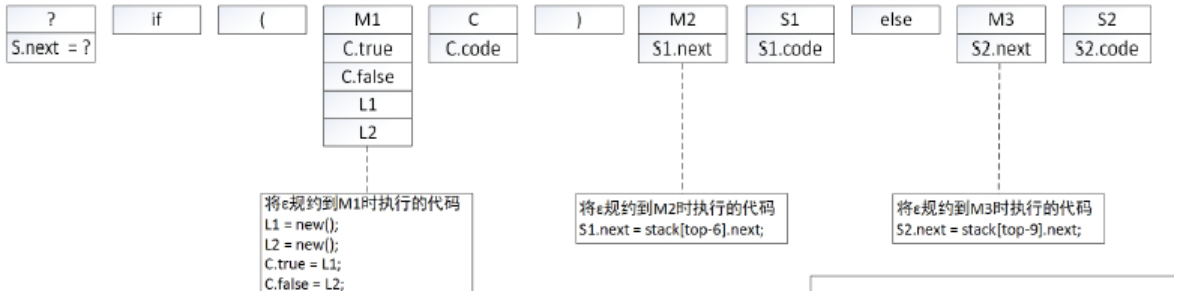
将 do(M1 S1) while(M2 c)规约到 S 时的代码：

tmpCode = 'lablel' || stack[top-6].L1 || stack[top-5].code || stack[top-1].code;

top = top – 7;

stack[top].code = tmpCode

2)



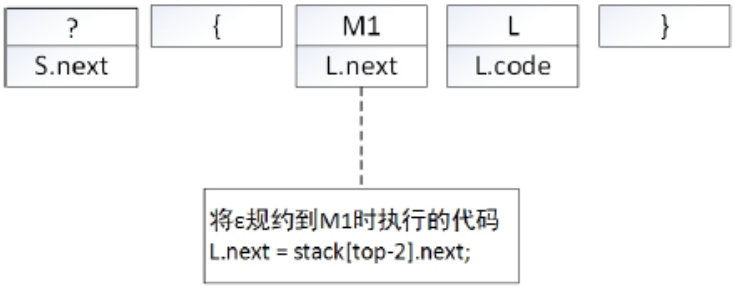
将 do(M1 S1) while(M2 c)规约到 S 时的代码：

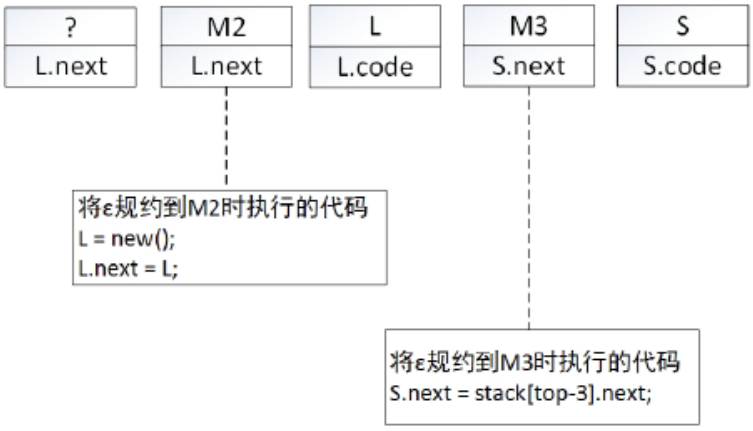
tmpCode = 'lablel' || stack[top-6].L1 || stack[top-5].code || stack[top-1].code;

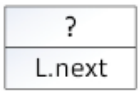
top = top – 7;

stack[top].code = tmpCode

3)







将{L}规约到 S 时执行的代码：

```
tmpCode = stack[top-1].code;
top = top - 3;
stack[top].code = tmpcode;
```

将 M2 L M3 S 规约到 L 时执行的代码：

```
tmpCode = stack[top-2].code || stack[top].code;
top = top - 3;
stack[top].code = tmpcode;
```

将 ε 规约到 L 时执行的代码：

```
tmpCode = “ ”;
top = top + 1;
stack[top].code = tmpcode;
```