编译原理第五章第三次作业

李昊宸2017K8009929044

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 -> L S | ε

请注意，列表中的任何语句都可能包含一条从它的内部跳转到下一个语句的跳转指令，因此简单地为各个语句按顺序生成代码是不够的。

答：

构造说明：S.next表示在执行完S段的代码后，最后一步跳转到的位置。所以，如果S1.next = L3，那么在S1.code中会在最后显式的增加跳转到L3的指令。

对于X而言，继承属性X.inh出现在X栈的上方，综合属性X.syn出现在X栈的下方

1）S -> if ( C ) S1 else S2 L2 = new();

S1.next = S.next;

C.false = L2;

S.code = C.code || S1.code || label || L2 || S2.code

2) S -> do S1 while ( C ) L1 = new();

C.true = L1;

S.code = label || L1 || S1.code || C.code

3) 存在左递归，故先消除左递归

S -> ‘{’ L ‘}’ S.code = “{” || L.code || “}”

L -> L’ L.code = L’.code

L’.inh=ε

L’ -> S L’1 L’1.inh = L’inh || S.code

L’.code = L’1.code

L’ -> ε L’.code = Li.inh

5.5.4按照 5.5.3 节的风格，将练习 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（或者指向代码的指针）存放在栈中。

答：具体的SDT动作见栈动作图

修改为SDT：

1）S -> if ( {L2 = new();

C.false = L2;}

C

) {S1.next = S.next; }

S1

else

S2 {S.code = C.code || S1.code || label || L2 || S2.code}

top

top

|  |  |
| --- | --- |
| S | Synthesize  S.code |
| next=x | code=? |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| if | ( | Action | C | Synthesize  C.code | ) | S1 | Synthesize  S1.code | else | S2 | Synthesize  S2.code | Synthesize  S.code |
|  |  | snext=x | true=? | code=? |  | next=? | code=? |  | next=? | code=? | code=? |
|  |  | L2=? |  |  |  |  |  |  |  | Ccode=? |  |
|  |  |  |  |  |  |  |  |  |  | code1=? |  |
|  |  |  |  |  |  |  |  |  |  | l2=? |  |

|  |
| --- |
| stack[top-1].code=Ccode||code1||”label”||l2||code |

|  |
| --- |
| L2=new(); |
| stack[top-1].true=L2;  stack[top-4].next=x;  stack[top-7].next=x;  stack[top-8].l2= L2; |

|  |
| --- |
| stack[top-6].Ccode=code |

|  |
| --- |
| stack[top-3].code1=code |

2）S -> do S1 while ( {L1 = new();

top

C.true = L1;}

C

) {S.code = label || L1 || S1.code || C.code}

|  |  |
| --- | --- |
| S | Synthesize  S.code |
| next=x  top | code=? |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| do | S1 | Synthesize  S1.code | while | ( | Action | C | Synthesize  C.code | ) | Synthesize  S.code |
|  |  | code=? |  |  | L1=? | true=? | code=? |  | code=? |
|  |  |  |  |  |  | false=? | code1=? |  |  |
|  |  |  |  |  |  |  | l1=? |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |
| --- |
| stack[top-5].code1=code |

|  |
| --- |
| stack[top-2].code=”label”||l1||code1||code |

|  |
| --- |
| L1=new(); |
| stack[top-1].true=L1;  stack[top-1].false=x;  stack[top-2].l1 = L1; |

3)

S -> ‘{’ L ‘}’ { S.code = “{” || L.code || “}”}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| { | L | Synthesize  L.code | } | Synthesize  S.code |
|  |  | code=? |  | code=? |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |
| --- |
| stack[top-2].code=”{”||code||”}” |

L -> L’ { L.code = L’.code; L’.inh=ε}

|  |  |  |
| --- | --- | --- |
| L’ | Synthesize  L.code | Synthesize  L.code |
| L’.inh=? | code=? | code=? |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| stack[top-1].code=code |

L’ -> {stack[top+2].L’.inh=stack[top].L’inh}

S { L’1.inh = L’inh || S.code}

L’1 { L’.code = L’1.code }

|  |  |
| --- | --- |
| L’ | Synthesize  L’.code |
| L’inh | code=? |

|  |
| --- |
| stack[top+2].L’.inh=stack[top].L’inh |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | Synthesize  S.code | ACTION | L’1 | Synthesize  L’1.code | Synthesize  L’.code |
|  | code=? | L’.inh=? | ihn=? | code=? | code=? |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |
| --- |
| stack[top-1].code=stack[top].code |

|  |
| --- |
| stack[top-2].inh=stack[top-1].inh||code |

L’ -> ε { L’.code = L’.inh}

|  |  |
| --- | --- |
| L’ | Synthesize  L’.code |
| L’.inh=? | code=? |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| stack[top-1].code=inh |

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

答：

对于X而言，综合属性X.syn出现在X的记录中，继承属性X.inh出现在X栈的下方

1）

S -> if ( M1 C ) M2 S1 else M3 S2 {S.code = C.code || S1.code || label || L2 || S2.code}

M1 -> ε {L2 = new();C.false = L2;}

M2 -> ε {S1.next = S.next;}

M3 -> ε {S2.next = S.next;}

top

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ？ | if | ( | M1 | C | ) | M2 | S1 | else | M3 | S2 |
| S.next |  |  | C.true | C.code |  | S1.next | S1.code |  | S2.next | S2.code |
|  |  |  | L2 |  |  |  |  |  |  |  |

|  |
| --- |
| tempCode=stack[top-6].code||stack[top-3].code||”label”||stack[top-7].L2||stack[top].code;  top=top-9;  stack[top].code=tempCode; |

|  |
| --- |
| L2=new(); |
| C.true=L2; |

|  |
| --- |
| S1.next=stack[top-6].next; |

|  |
| --- |
| S2.next=stack[top-9].next; |

2）S -> do S1 while (M1 C ) {S.code = label || L1 || S1.code || C.code}

M1 -> ε {L1 = new();C.true = L1;}

top

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ? | do | S1 | while | ( | M1 | C | ) |
| S.next |  | S1.code |  |  | C.true | C.code |  |
|  |  |  |  |  | C.false |  |  |
|  |  |  |  |  | L1 |  |  |
|  |  |  |  |  |  |  |  |

|  |
| --- |
| L1=new(); |
| C.true=L1;  C.false=stack[top-4].next; |

|  |
| --- |
| tempCode=”label”||stack[top-2].L1||stack[top-4].code||stack[top-1].code;  top=top-5;  stack[top].code=tempCode; |

3)S -> ‘{’ L ‘}’ {tempCode=”{”||stack[top-1].L.code||”}”;

top=top- 2;

stack[top].code=tempCode;}

L -> N L’ { stack[top]L.code= stack[top].L’.code; }

N -> ε {stack[top].L’.inh = ε}

L’ -> S M1 L’1 {

tempCode = stack[top].L’1.code;

top = top-2;

stack[top].L’.code = tempCode;

}

M1 -> ε {stack[top].L’1.inh = stack[top-2].L’.inh || stack[top-1].S.code}

L’ -> ε {stack[top].L’.code = stack[top].Li.inh}

ps:5.5.5中的文法3）因为有很多个产生式，如果每个产生式都画出一个对应的栈状态图太过繁琐，并且每个产生式的右部文法符号或终结符不超过三个，较为直观，故不再给出栈状态图。