5.5实现L属性的SDD

- 递归下降语法分析器的扩展
- 递归下降分析器（边扫描边生成）
- LL语法分析器的扩展
- LR语法分析器的扩展

while语句的翻译

while(i<5) j=j+1;

k=15;

L_1: if(i<5) goto L_2

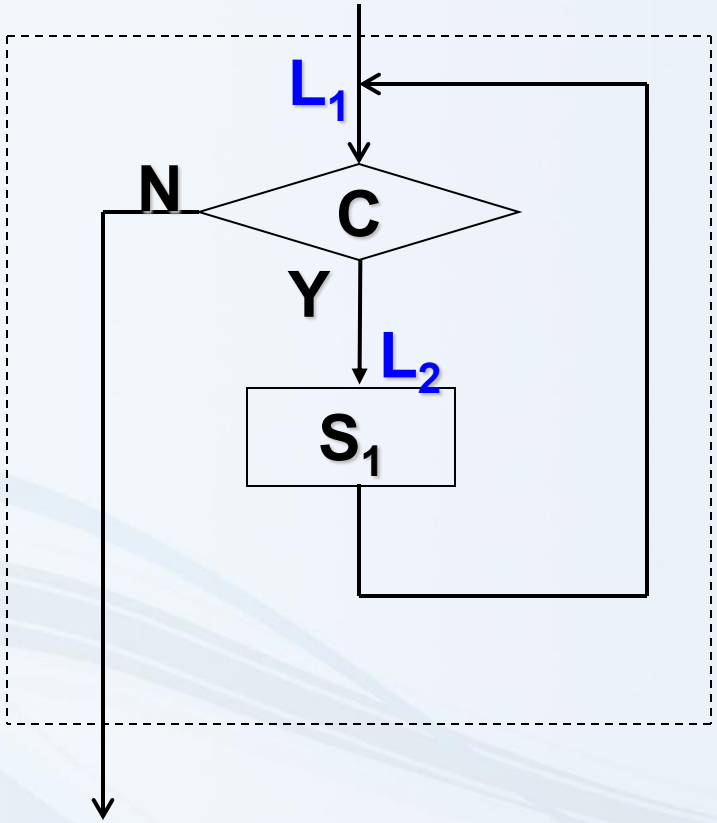
goto L_0

L_2:t_1=j+1

j=t_1

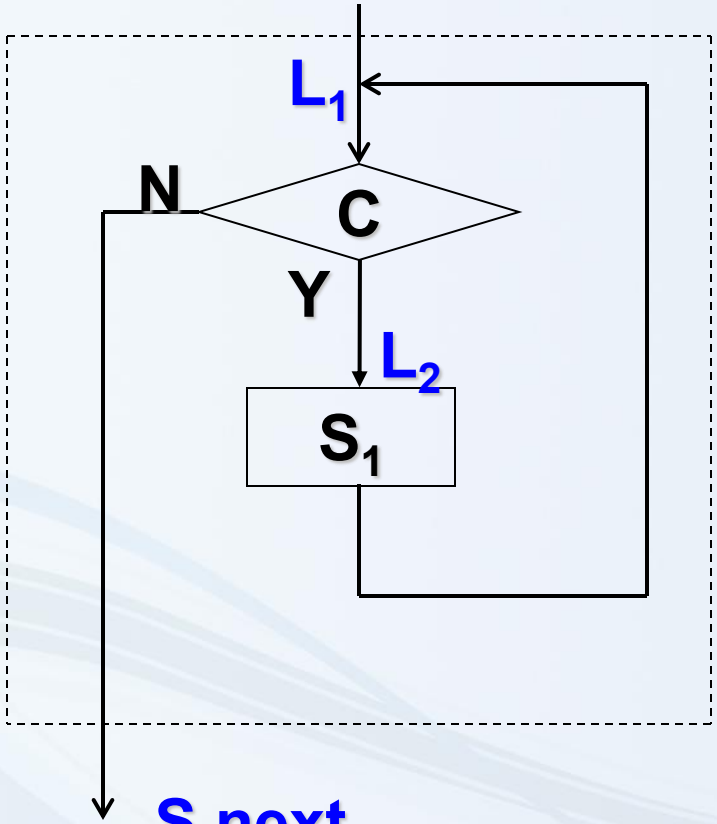
goto L_1

L_0: k=15



$S \rightarrow while(C)S_1$

while语句的翻译(2)



虚线框内有几个出口，哪些出口需要引入标记？

$S \rightarrow while(C)S_1$

{L_1 = new( );

L_2 = new( );

$S_1.next = L_1$;

$C.false = S.next$

$C.true = L_2$;

S.code = label || L_1 || C.code|| label || L_2 || S_1.code}

||是什么意思？

5.5.1递归下降语法分析器的扩展

- 在递归下降分析中应用L属性定义
- 递归下降分析
  - 文法要求：没有左递归，没有左公共因子
  - 为每个非终结符，构建一个过程来处理以它为左部的所有产生式

❖扩展分析器（设*P*为分析非终结符A的过程）

- *P*的参数中，加上A的继承属性
- *P*的返回值，为A的综合属性
- 过程体以局部变量形式保存:
  - ✓ 规则右部涉及到的非终结符的属性（继承属性和综合属性）
  - ✓ 为了计算这些属性值所需的其它变量

$E \rightarrow T \{R.i = T.val\}\ R \{E.val = R.s\}$

$R \rightarrow + T \{R_1.i = R.i + T.val\}\ R_1 \{R.s = R_1.s\}$

$R \rightarrow - T \{R_1.i = R.i - T.val\}\ R_1 \{R.s = R_1.s\}$

$R \rightarrow \varepsilon \quad \{R.s = R.i\}$

$T \rightarrow (E) \{T.val = E.val\}$

$T \rightarrow num \{T.val = num.lexval\}$

例1

非终结符R的递归子程序（语法分析）

```
void R( ){
   switch(current input){
      case '+': advance input; T( ); R( );
      case '-':  advance input; T( ); R( );
      default: ;
   }
}
```

$E \rightarrow T\ \{R.i = T.val\}\ R\ \{E.val = R.s\}$

$R \rightarrow +\ T\ \{R_1.i = R.i + T.val\}\ R_1\ \{R.s = R_1.s\}$

$R \rightarrow -\ T\ \{R_1.i = R.i - T.val\}\ R_1\ \{R.s = R_1.s\}$

$R \rightarrow \varepsilon\ \ \{R.s = R.i\}$

$T \rightarrow (E)\ \{T.val = E.val\}$

$T \rightarrow num\ \{T.val = num.lexval\}$

例1

对照语义动作，说明：
R函数的形参是什么？
R函数返回什么？

非终结符R的递归子程序（扩展语义处理）

```
int R(int i){
    int R1i, R1s, Tval, Rs;
    switch(current input){
      case '+': advance input; Tval=T( ); R1i=i+Tval; R1s=R(R1i);Rs=R1s;
      case '-':  advance input; Tval=T( ); R1i=i-Tval;  R1s=R(R1i); Rs=R1s;
      default: Rs=i;
    }
     return Rs;
}
```

例2（回顾：while语句语法分析的代码）

L_1 = new(); L_2 = new()
$S_1.next = L_1$;
C.*false* = S.*next*
C.*true* = L_2;
S.code = label||L_1||C.code||label ||L_2||S_1.code

❖ $S \rightarrow while(C)S_1$

```
void S( ){
   if(current input == token while){
         advance input;
         check '(' is next on the input , and advance;
         C( );
         check ')' is next on the input , and advance;
         S( );
    }
    else /* other statement types*/
}
```

为什么先check，而不是直接advance input?

例2

L_1 = new(); L_2 = new()
$S_1.next = L_1$;
C.*false* = S.*next*
C.*true* = L_2;
S.code = label||L_1||C.code||label ||L_2||S_1.code

❖ $S \rightarrow while(C)S_1$

```
string S(label next){
    string S1code, Ccode;
    label   L1, L2;
    if(current input == token while){
            advance input;
            check '(' is next on the input, and advance;
            L1          =  new( );
            L2          =  new( );
            Ccode       =  C(next, L2)
            check ')' is next on the input , and advance;
            S1code      =  S(L1);
            return ("label"||L1||Ccode || "label" || L2 ||S1code);
     }
     else /* other statement types*/
}
```

一次性输出可能需要非常大的存储空间

5.5.2边扫描边生成代码

$S \rightarrow while(C)S_1$

S → while ({L1=new(); L2=new( );
C.false=S.next; C.true=L2; print("label", L1);}
C) {print ("label", L2); S1.next=L1; } S1

```
void S(label next){
  label  L1, L2;
  if(current input == token while){
        advance input;
        check '(' is next on the input , and advance;
        L1      = new( );
        L2      = new( );
        print("label", L1);
        C(next, L2);
        check ')' is next on the input , and advance;
        print("label" , L2);
        S(L1);
  }
  else /* other statement types*/
}
```

为什么返回值为空？

立即输出不需要太大存储空间

5.5.4 LR分析器的扩展

- 自底向上分析显然可以处理S属性定义
- L属性定义要解决的问题:
  - 嵌入语义动作的处理
    - ✓引入标记非终结符号M（也称为占位符）和规则M→ε，将原语义动作与规则M→ε结合
    - ✓去掉原来的嵌入语义动作
  - 继承属性的处理
    - ✓用综合属性代替继承属性

自底向上分析（嵌入语义动作的处理）

E→ T R

R→ + T {print('+');} R_1 | - T {print('-');} R_1 | ε

T → num {print(num.*val*) ;}

E→ T R

R→ + T M R_1 | - T N R_1 | ε

T → num {print(num.*val*) ;}

M → ε {print('+');}

N → ε {print('-');}

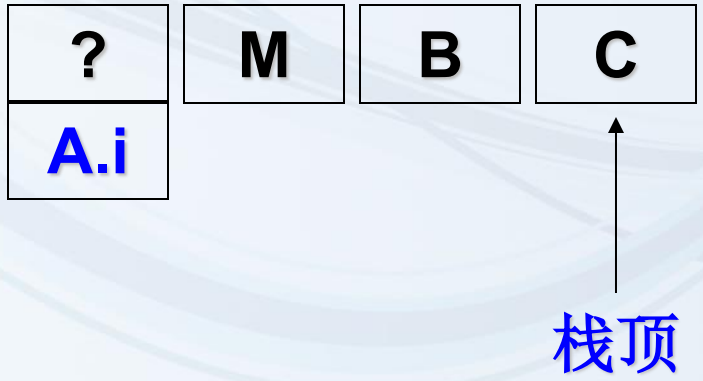
自底向上分析（继承属性处理）

- 规则右部文法符号继承属性的处理
  - 参照嵌入语义动作处理
- 规则左部文法符号继承属性的处理
  - 约定：信息直接埋入栈内该文法符号下面一个符号的属性值内

$A \rightarrow \{B.i=f(A.i);\}\ B\ C$

$A \rightarrow M\ B\ C$

$M \rightarrow \varepsilon\{M.s=f(A.i);\}$



例

1) $T \rightarrow F$ { $T'.inh = F.val$ } T' { $T.val = T'.syn$ }

2) $T' \rightarrow *F$ { $T_1'.inh = T'.inh \times F.val$ } T_1' { $T'.syn = T_1'.syn$ }

3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }

4) $F \rightarrow$ digit { $F.val$ = digit .$lexval$ }

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }

$M \rightarrow \varepsilon$ { $M.s = F.val$ }

2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }

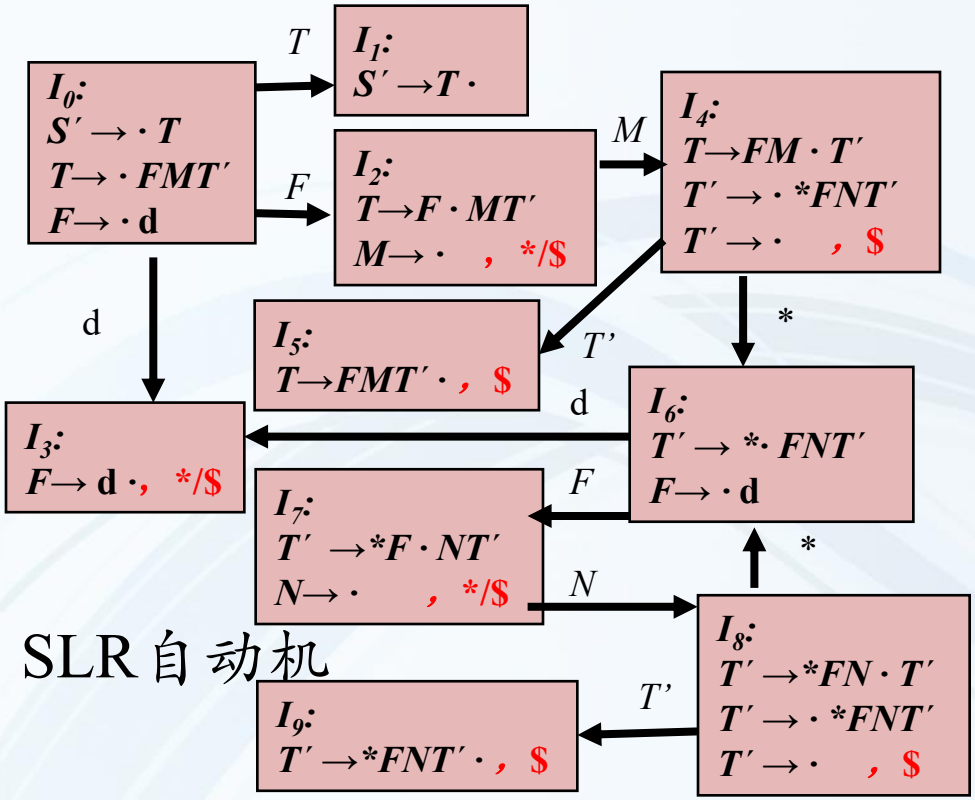
$N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }

3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }

4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }

例

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }



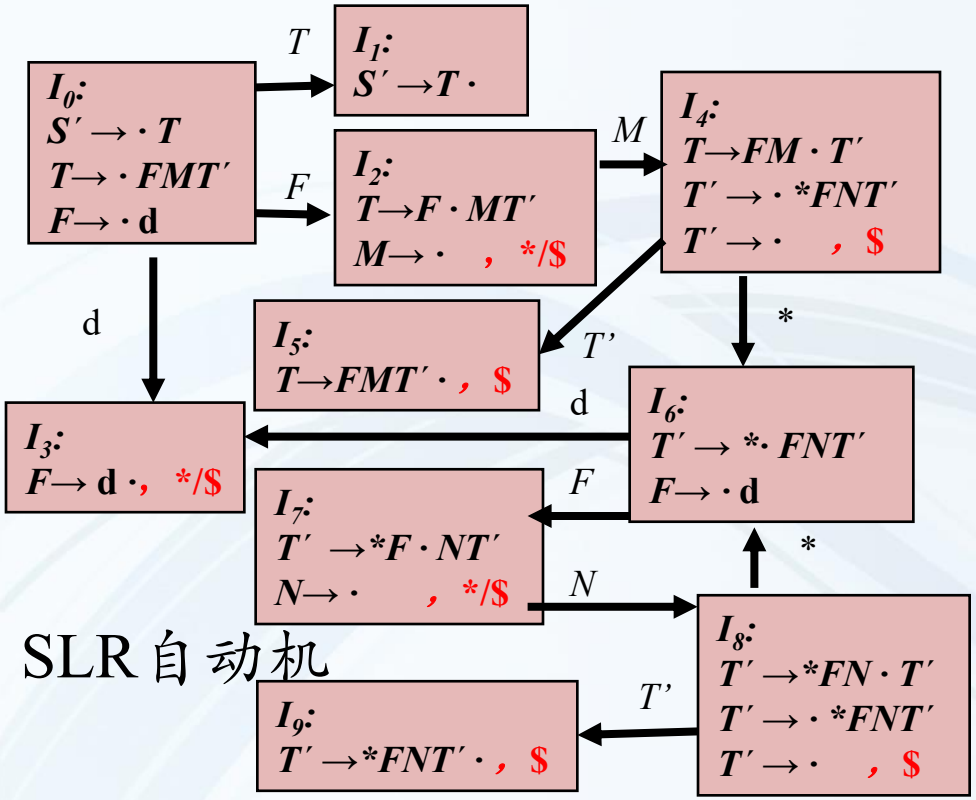
SLR自动机

输入： 3 * 5

0	3
$	d
	3

例

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }



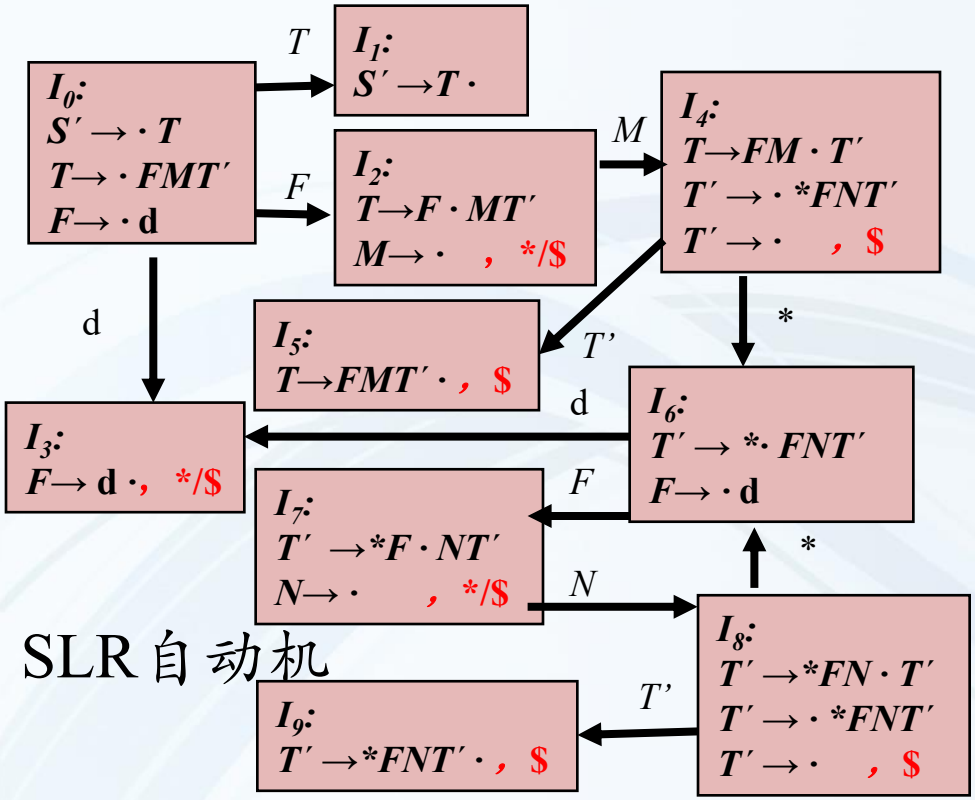
SLR自动机

输入： 3 * 5

0	2	4	6	3
$	F	M	*	d
	3	T′inh=3		5

例

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }



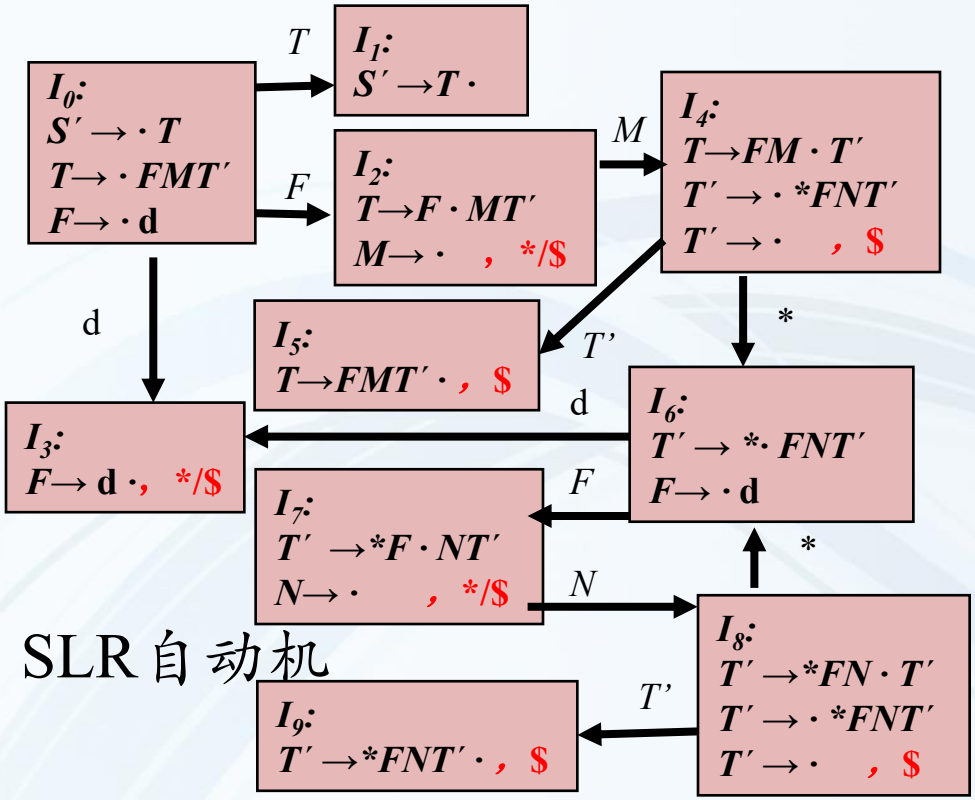
SLR自动机

输入： 3 * 5

0	2	4	6	7	8	9
$	F	M	*	F	N	T′
	3	T′inh=3		5	T_1'inh=15	syn=15

例

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }



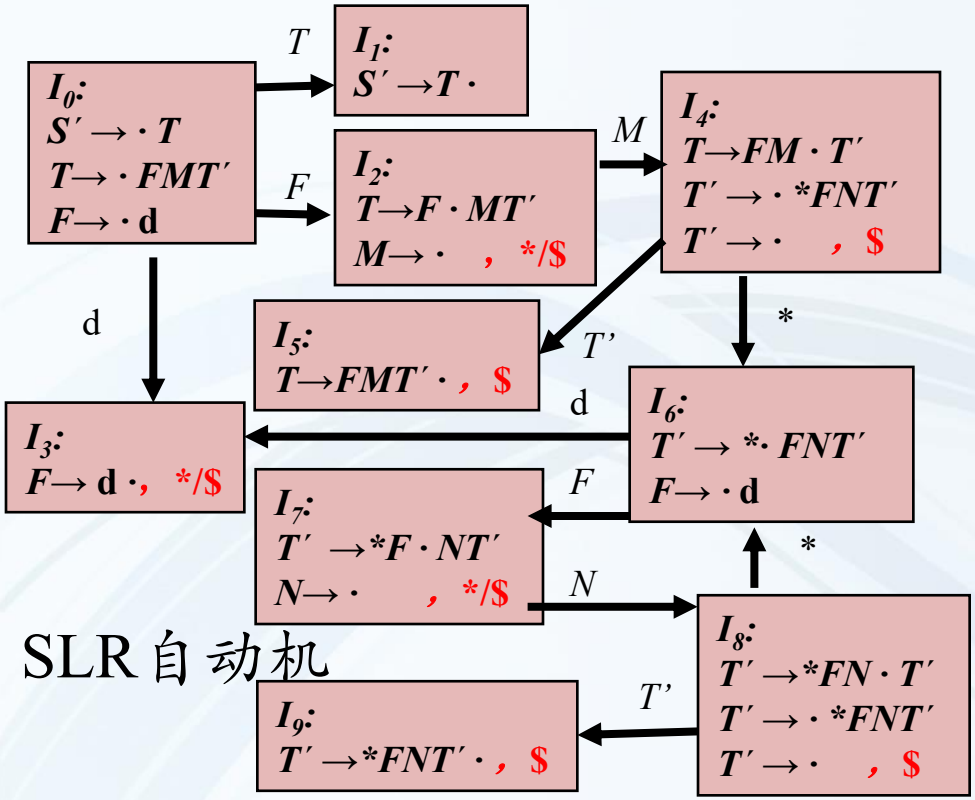
SLR自动机

输入： 3 * 5

0	2	4	5
$	F	M	T'
	3	T'inh=3	syn=15

例

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }



SLR自动机

输入： 3 * 5

0	1
$	T
	val=15

将语义动作改写为可执行的栈操作（哈工大版）

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }

1) $T \rightarrow F\ M\ T'$ {$stack[top-2].val = stack[top].syn;\ top = top-2;$}

   $M \rightarrow \varepsilon$ {$stack[top+1].T'inh = stack[top].val;\ top = top+1;$}

2) $T' \rightarrow *F\ N\ T_1'$ {$stack[top-3].syn = stack[top].syn;\ top = top-3;$}

   $N \rightarrow \varepsilon$ {$stack[top+1].T'inh = stack[top-2].T'inh \times stack[top].val;\ top = top+1;$}

3) $T' \rightarrow \varepsilon$ {$stack[top+1].syn = stack[top].T'inh;\ top = top+1;$}

4) $F \rightarrow$ digit {$stack[top].val = stack[top].lexval;$}

将语义动作改写为可执行的栈操作（龙书版）

1) $T \rightarrow F\ M\ T'$ { $T.val = T'.syn$ }
   $M \rightarrow \varepsilon$ { $M.s = F.val$ }
2) $T' \rightarrow *\ F\ N\ T_1'$ { $T'.syn = T_1'.syn$ }
   $N \rightarrow \varepsilon$ { $N.s = T'.inh \times F.val$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }

龙书版的语义动作和哈工大版的语义动作，对于什么规则是不同的？

1) $T \rightarrow F\ M\ T'$ {$stack[top-2].val = stack[top].syn$; $top = top-2$;}

   $M \rightarrow \varepsilon$ {$stack[top].T'inh = stack[top-1].val$;}

2) $T' \rightarrow *\ F\ N\ T_1'$ {$stack[top-3].syn = stack[top].syn$; $top = top-3$;}

   $N \rightarrow \varepsilon$ {$stack[top].T'inh = stack[top-3].T'inh \times stack[top-1].val$ }

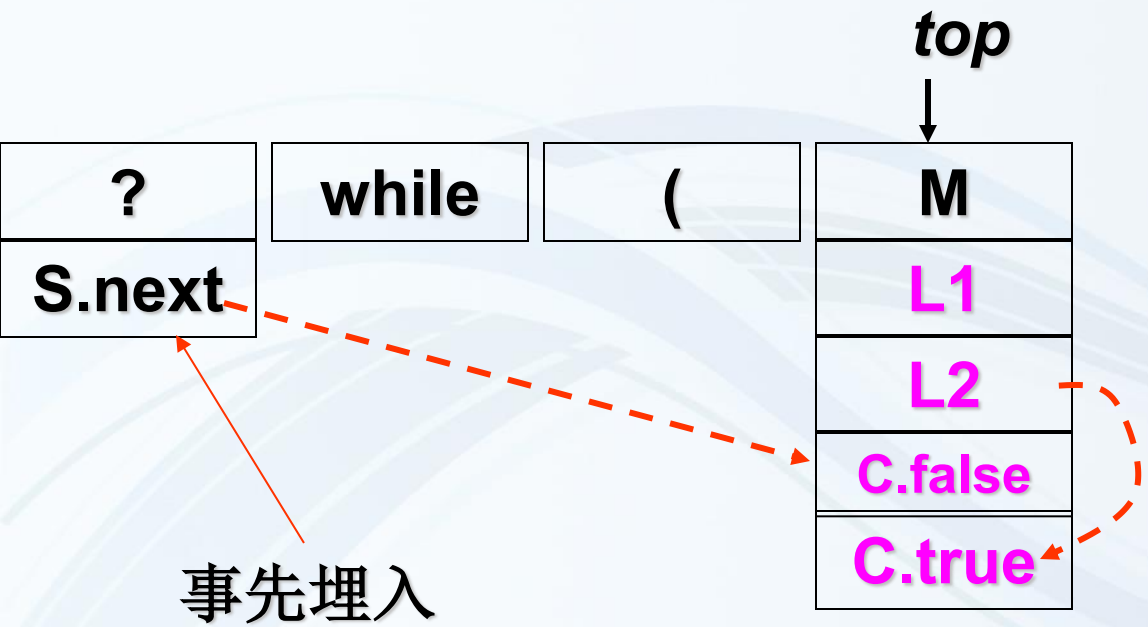
3) $T' \rightarrow \varepsilon$ {$stack[top+1].syn = stack[top].T'inh$; $top = top+1$;}

4) $F \rightarrow$ digit {$stack[top].val = stack[top].lexval$;}

$S \rightarrow$ while (M C) N S_1

$M \rightarrow \varepsilon$ {L_1=new(); L_2=new(); C.false=stack[top-3].next; C.true=L_2}

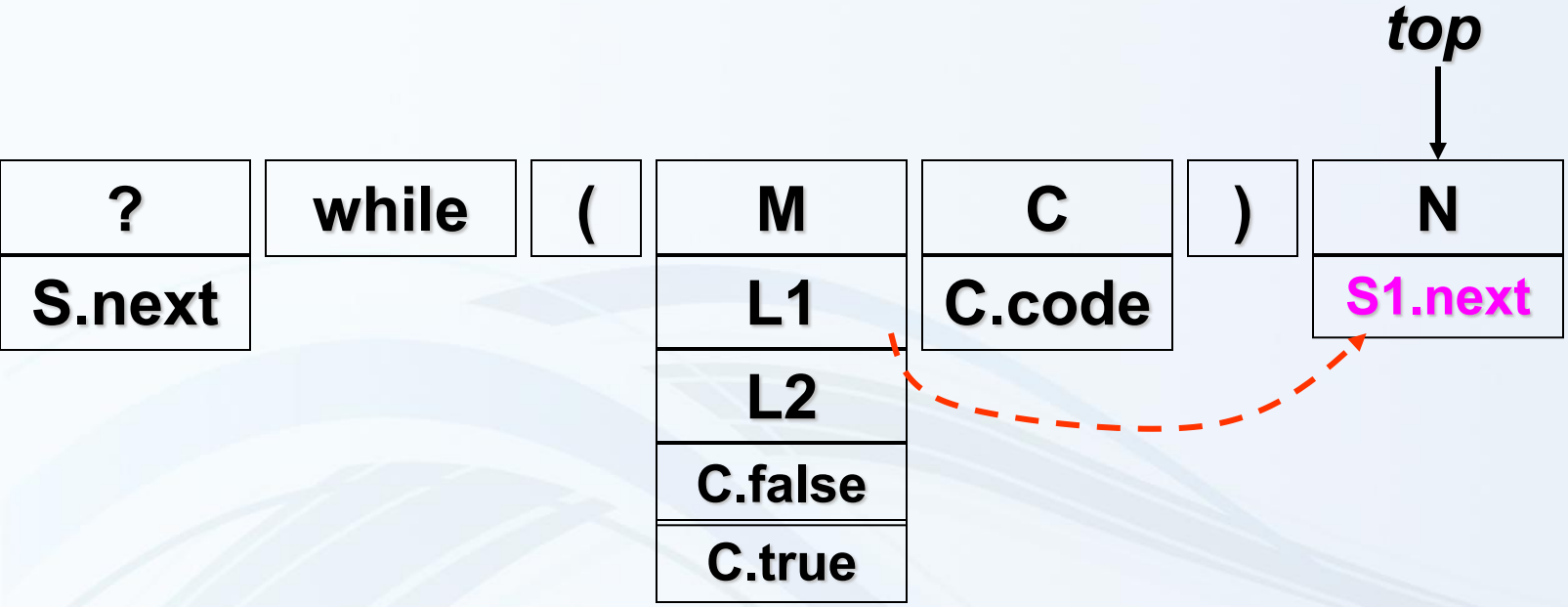
$N \rightarrow \varepsilon$ {S_1.next=stack[top-3].L_1 }



S → while (M C) N S1

M → ε {L1=new(); L2=new(); C.false=stack[top-3].next; C.true=L2}

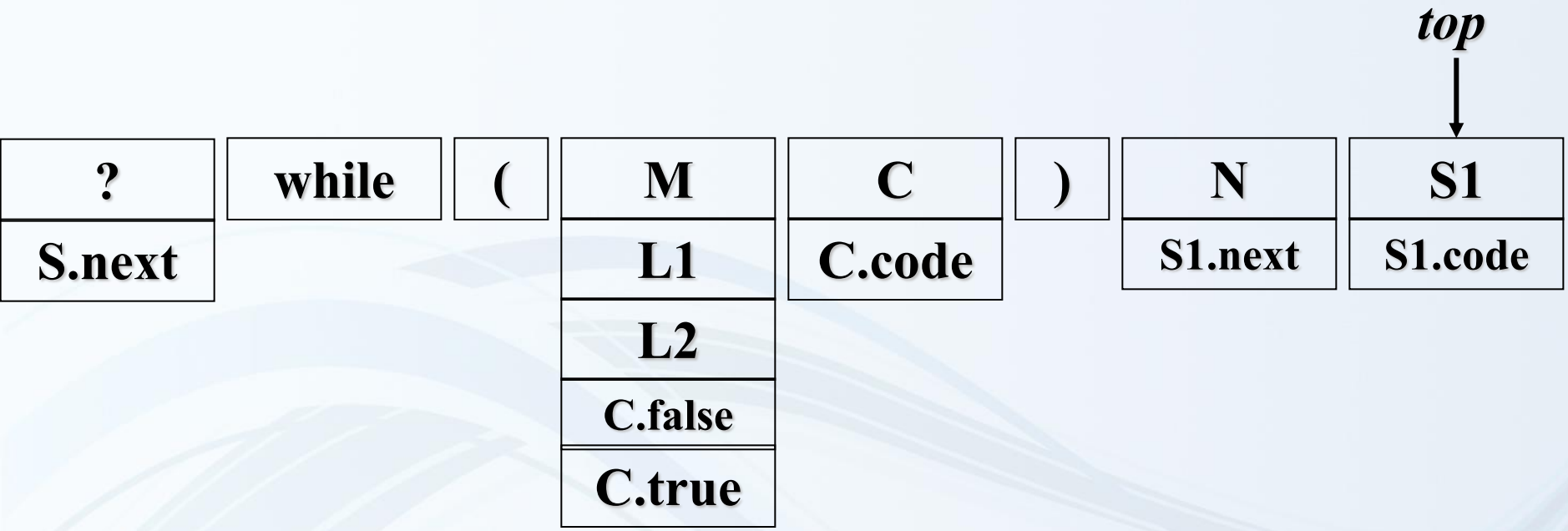
N → ε {S1.next=stack[top-3].L1 }



S → while (M C) N S1

M → ε {L1=new(); L2=new(); C.false=stack[top-3].next; C.true=L2}

N → ε {S1.next=stack[top-3].L1 }




```
tempCode = label || stack[top-4].L1 || stack[top-3].code ||
   label || stack[top-4].L2 || stack[top].code;
top  =  top – 6;
stack[top].code = tempCode;
```

为什么top=top-6?