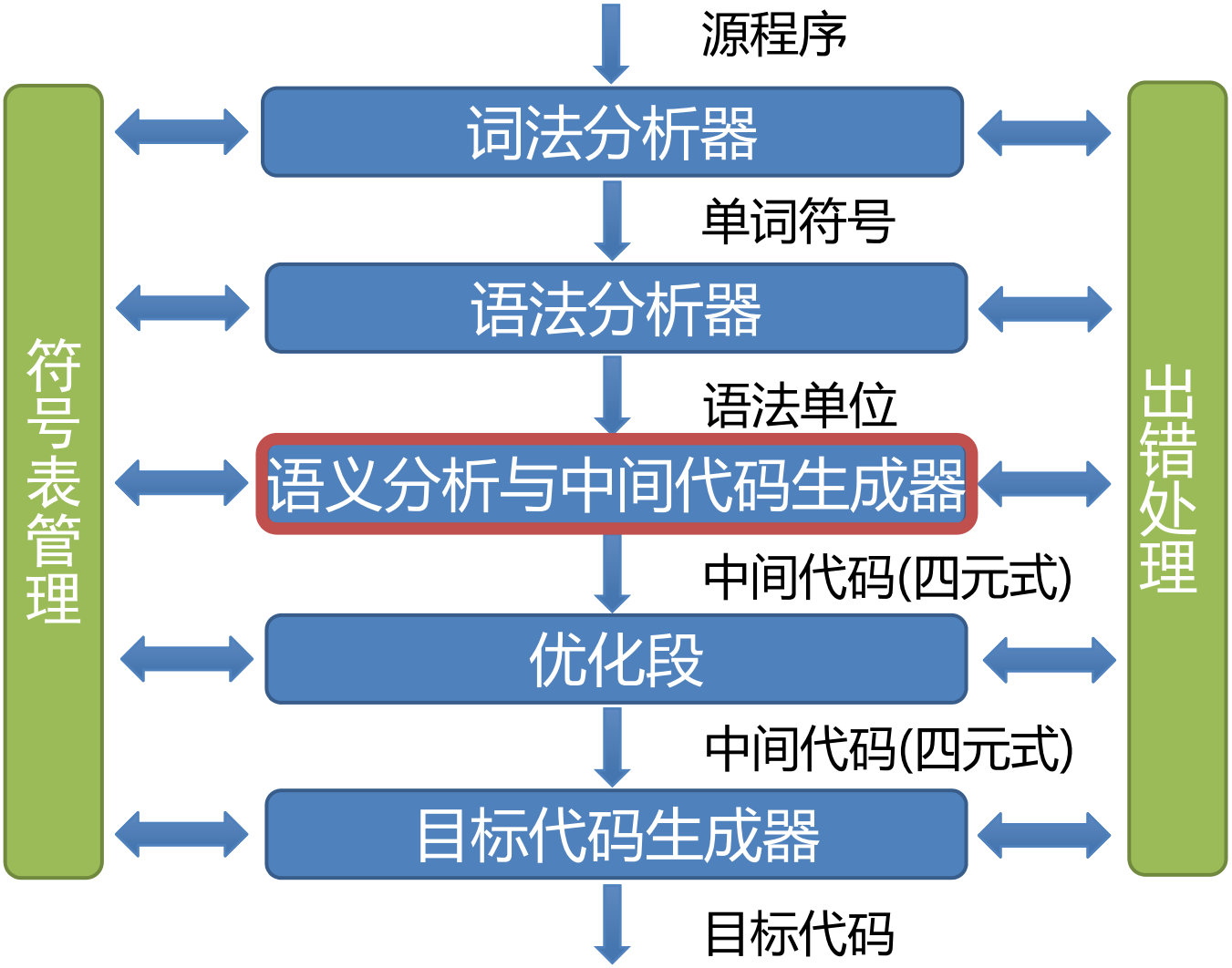
编译原理

控制语句的翻译

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

常用的控制语句

- $S \rightarrow$ if E then S_1
- $S \rightarrow$ if E then S_1 else S_2
- $S \rightarrow$ while E do S_1

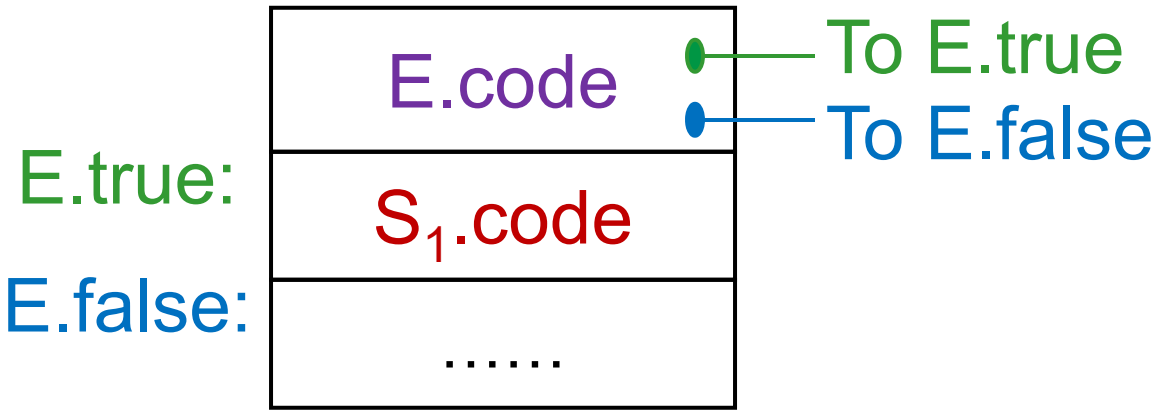
其中E为布尔表达式

编译原理

if语句的属性文法

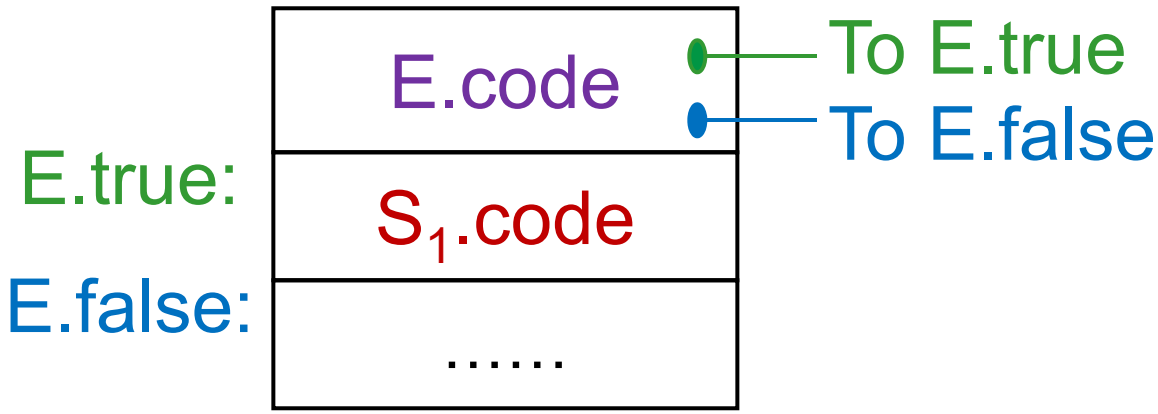
if-then语句的语义

- S → if E then S_1



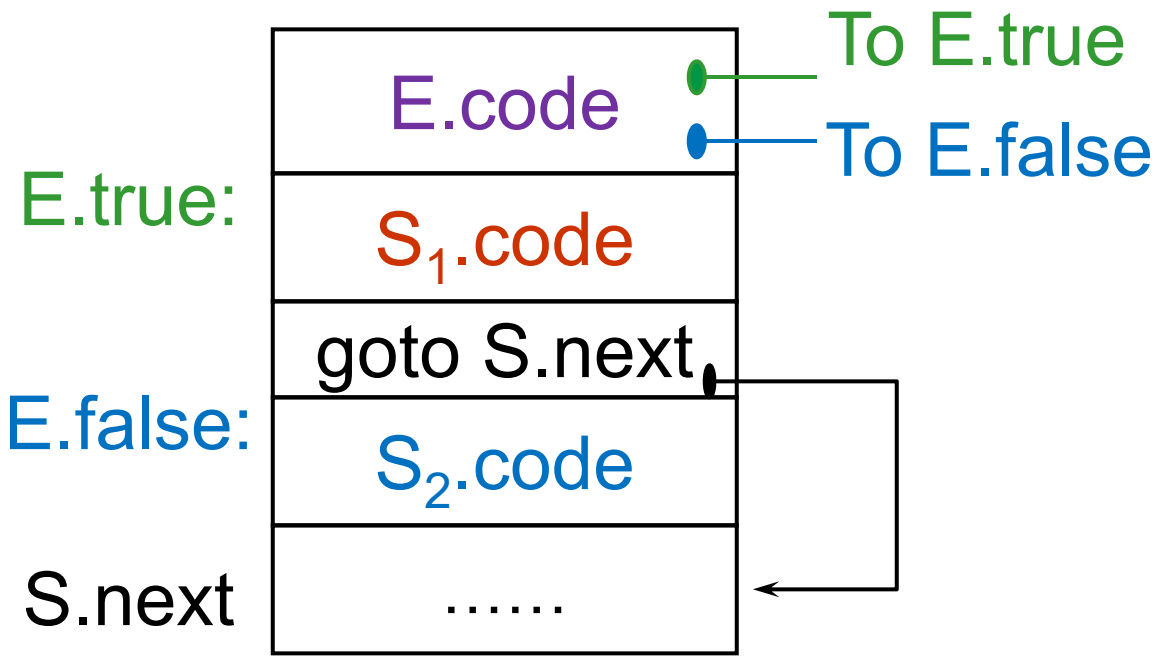
if-then语句的属性文法

产生式	语义规则
$S \rightarrow$ if E then S_1	E.true:=newlabel; E.flase:=S.next; S_1.next:=S.next S.code:=E.code || gen(E.true ':') || S_1.code



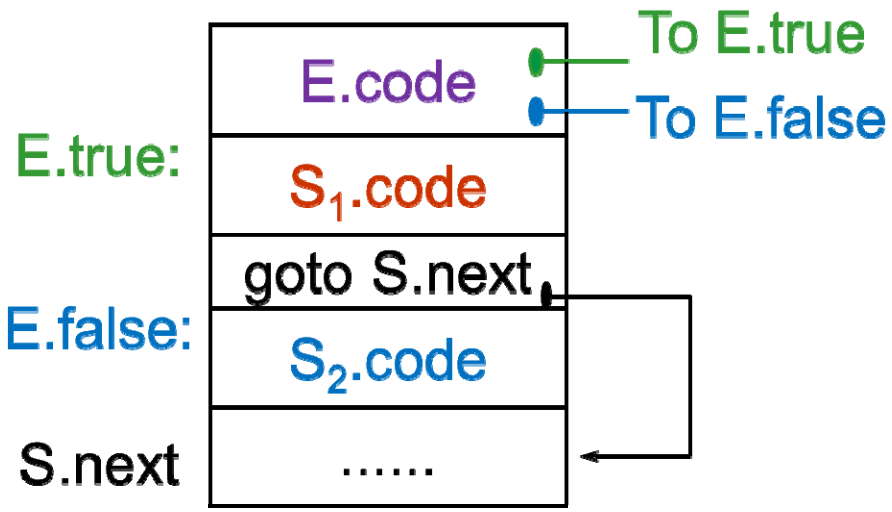
if-then-else语句的语义

- S → if E then S_1 else S_2



if-then-else语句的属性文法

产生式	语义规则
$S \rightarrow$ if E then S_1 else S_2	E.true:=newlabel; E.false:=newlabel; S_1.next:=S.next S_2.next:=S.next; S.code:=E.code || gen(E.true ':') || S_1.code || gen('goto' S.next) || gen(E.false ':') || S_2.code

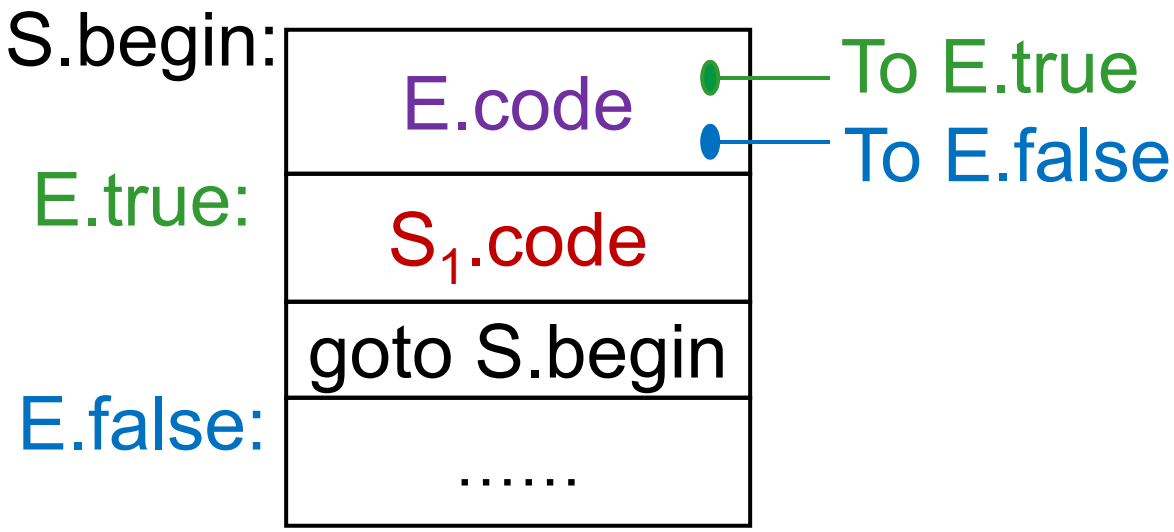


编译原理

while语句的属性文法

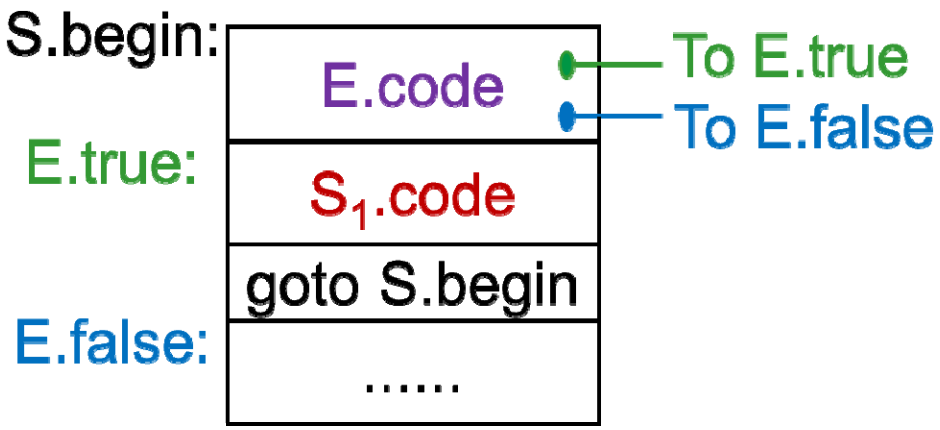
while-do语句的语义

- S → while E do S_1



while-do语句的属性文法

产生式	语义规则
S→while E do S_1	S.begin:=newlabel; E.true:=newlabel; E.false:=S.next; S_1.next:=S.begin; S.code:=gen(S.begin ':') || E.code || gen(E.true ':') || S_1.code || gen('goto' S.begin)



编译原理

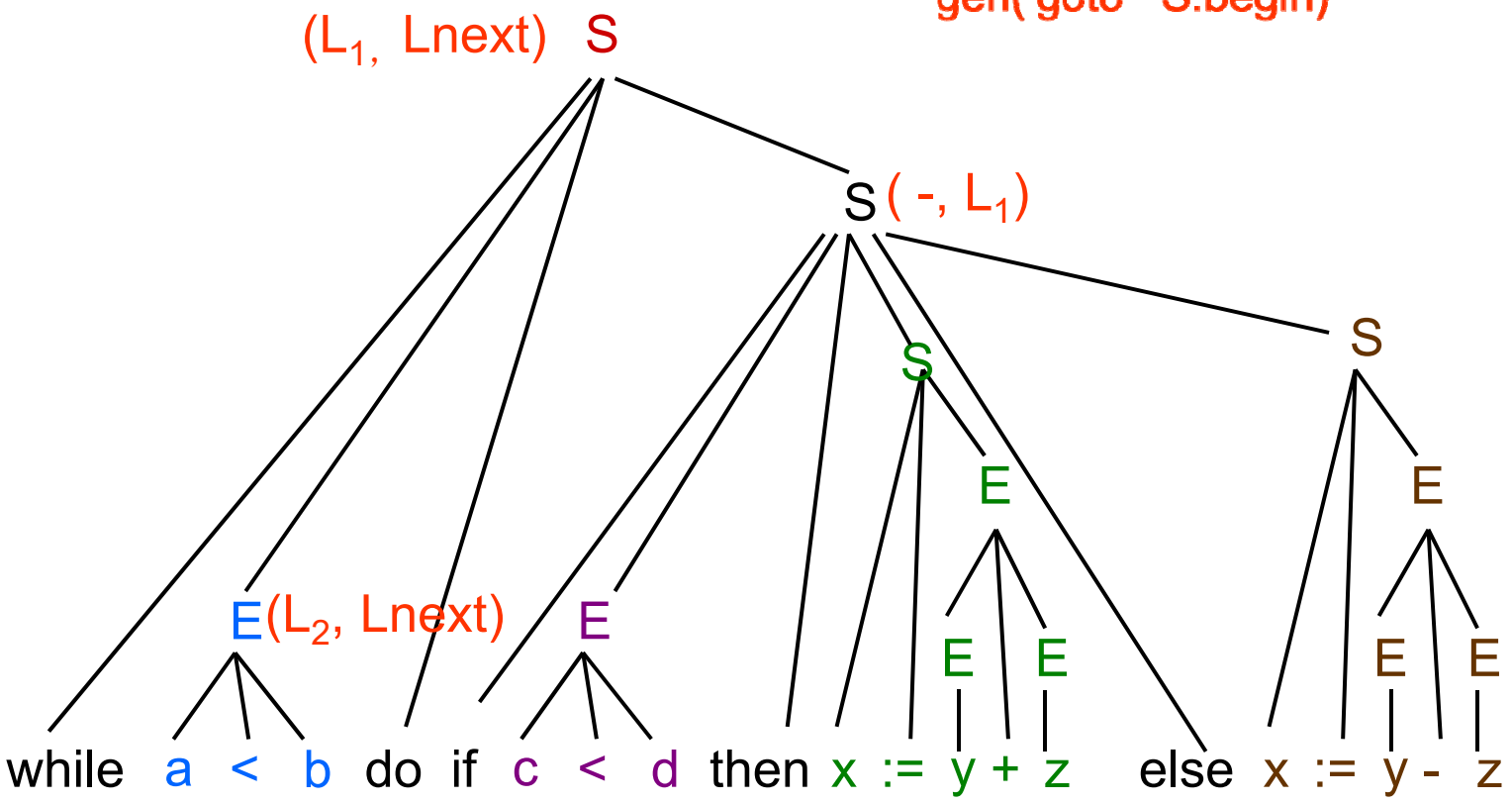
控制语句的属性计算示例

根据属性文法翻译控制语句

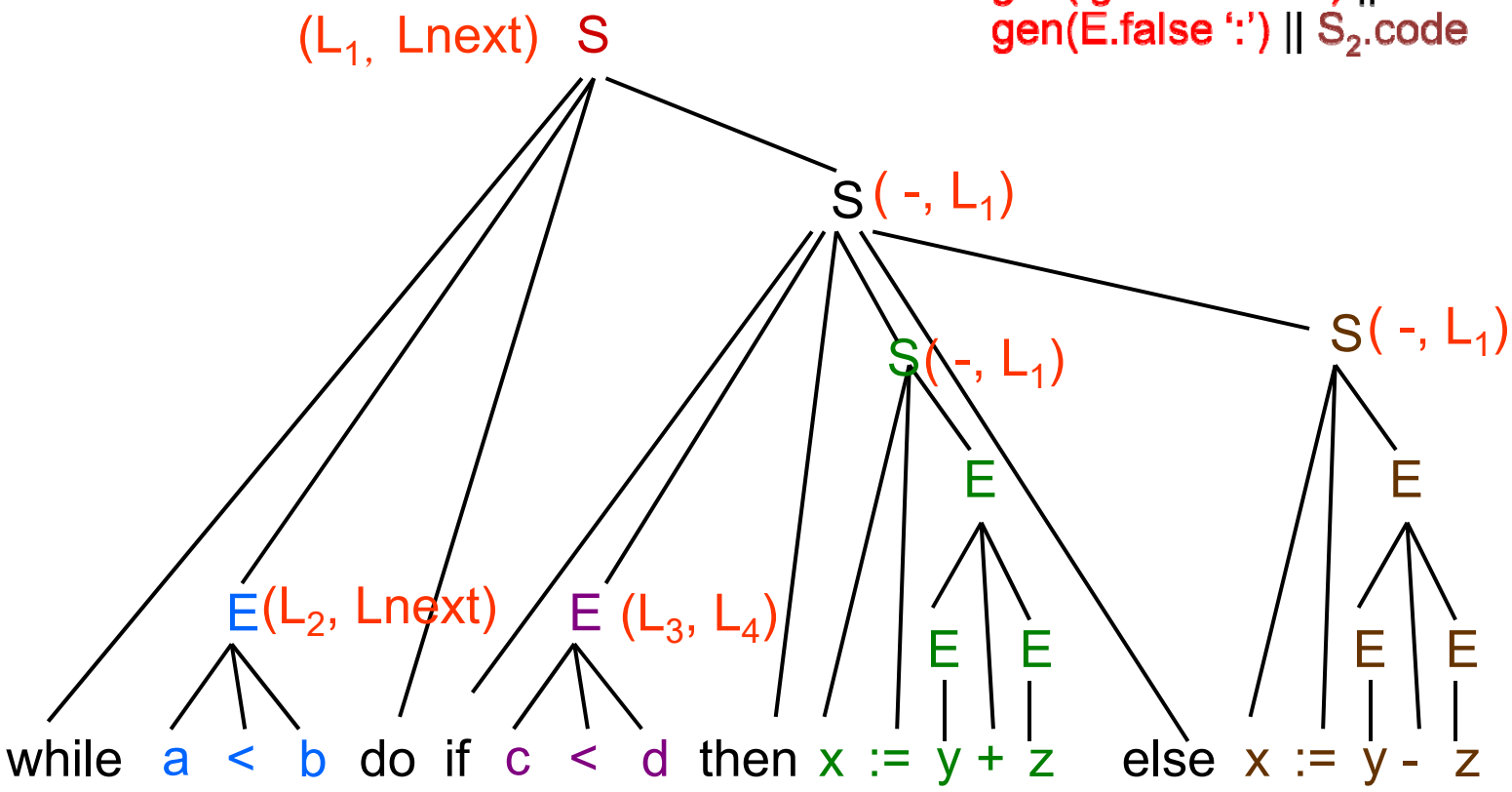
- 根据属性文法翻译如下语句：

```
while a<b do
    if c<d then   x:=y+z
          else    x:=y-z
```

产生式	语义规则
$S \rightarrow$ while E do S_1	S.begin:=newlabel; E.true:=newlabel; E.false:=S.next; S_1.next:=S.begin; S.code:=gen(S.begin ':') || E.code || gen(E.true ':') || S_1.code || gen('goto' S.begin)



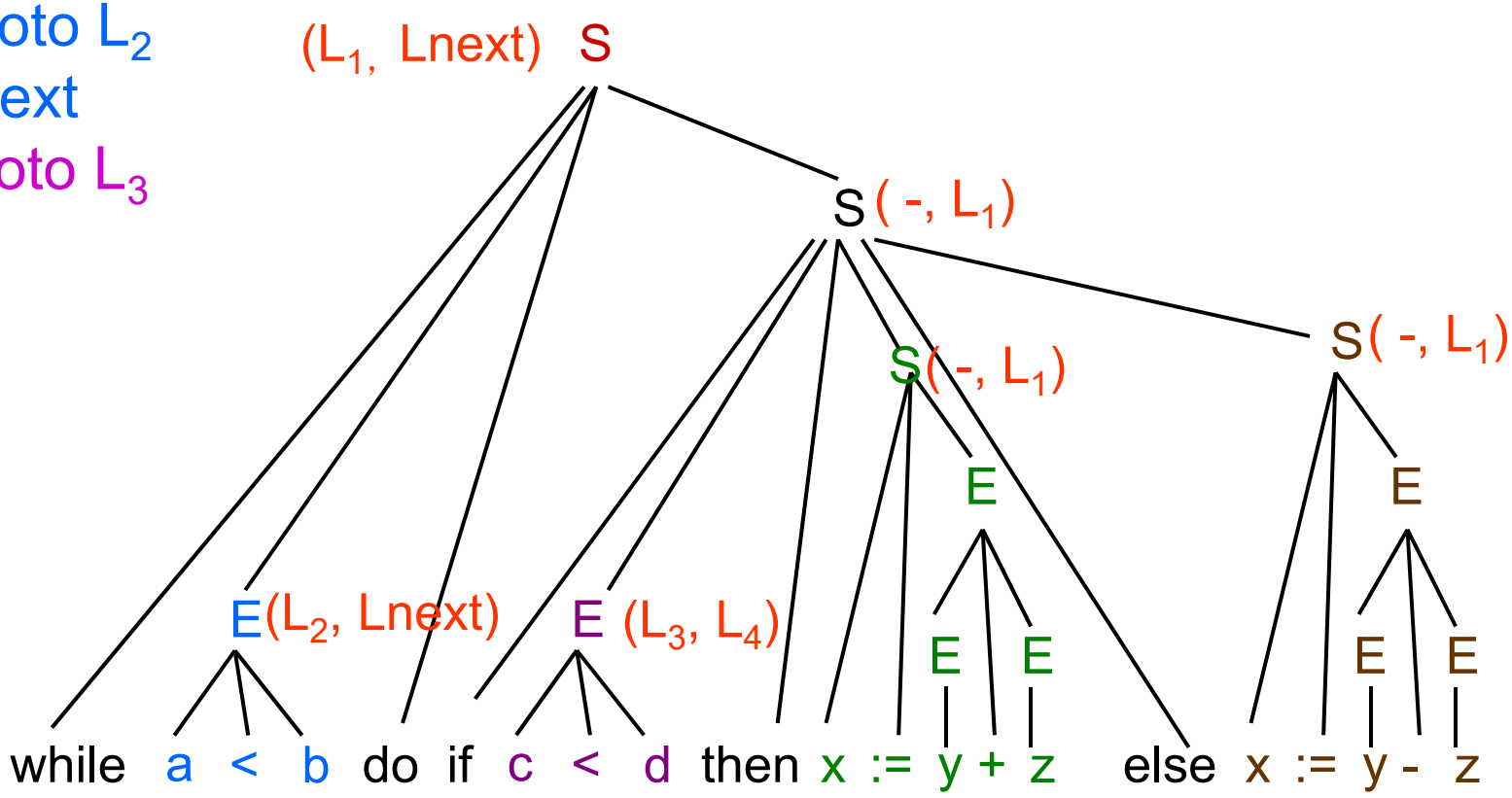
产生式	语义规则
$S \rightarrow$ if E then S_1 else S_2	E.true:=newlabel; E.false:=newlabel; S_1.next:=S.next S_2.next:=S.next; S.code:=E.code || gen(E.true ':') || S_1.code || gen('goto' S.next) || gen(E.false ':') || S_2.code



产生式　　语义规则

$E \rightarrow id_1\ relop\ id_2$　E.code:=gen('if ' id_1.place relop.op id_2.place 'goto' E.true)
|| gen('goto' E.false)

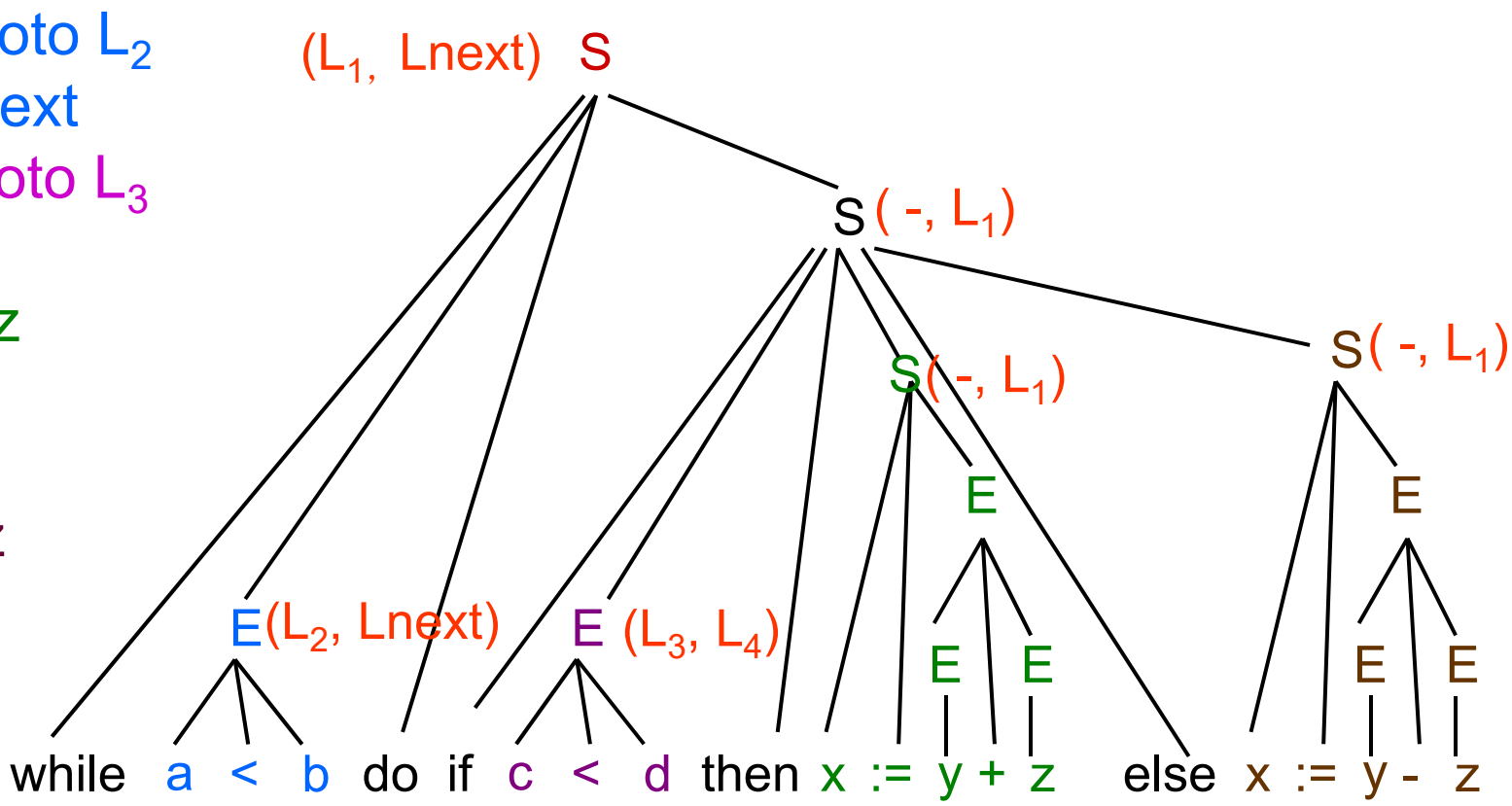
if a<b goto L_2
goto Lnext
if c<d goto L_3
goto L_4



产生式	语义规则
$S \rightarrow id := E$	S.code:=E.code || gen(id.place ':=' E.place)
$E \rightarrow E_1 + E_2$	E.place:=newtemp; $E.code := E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '+' E_2.place)

if a<b goto L_2
goto Lnext
if c<d goto L_3
goto L_4
T_1 :=y+z
x:=T_1

T_2 :=y-z
x:=T_2



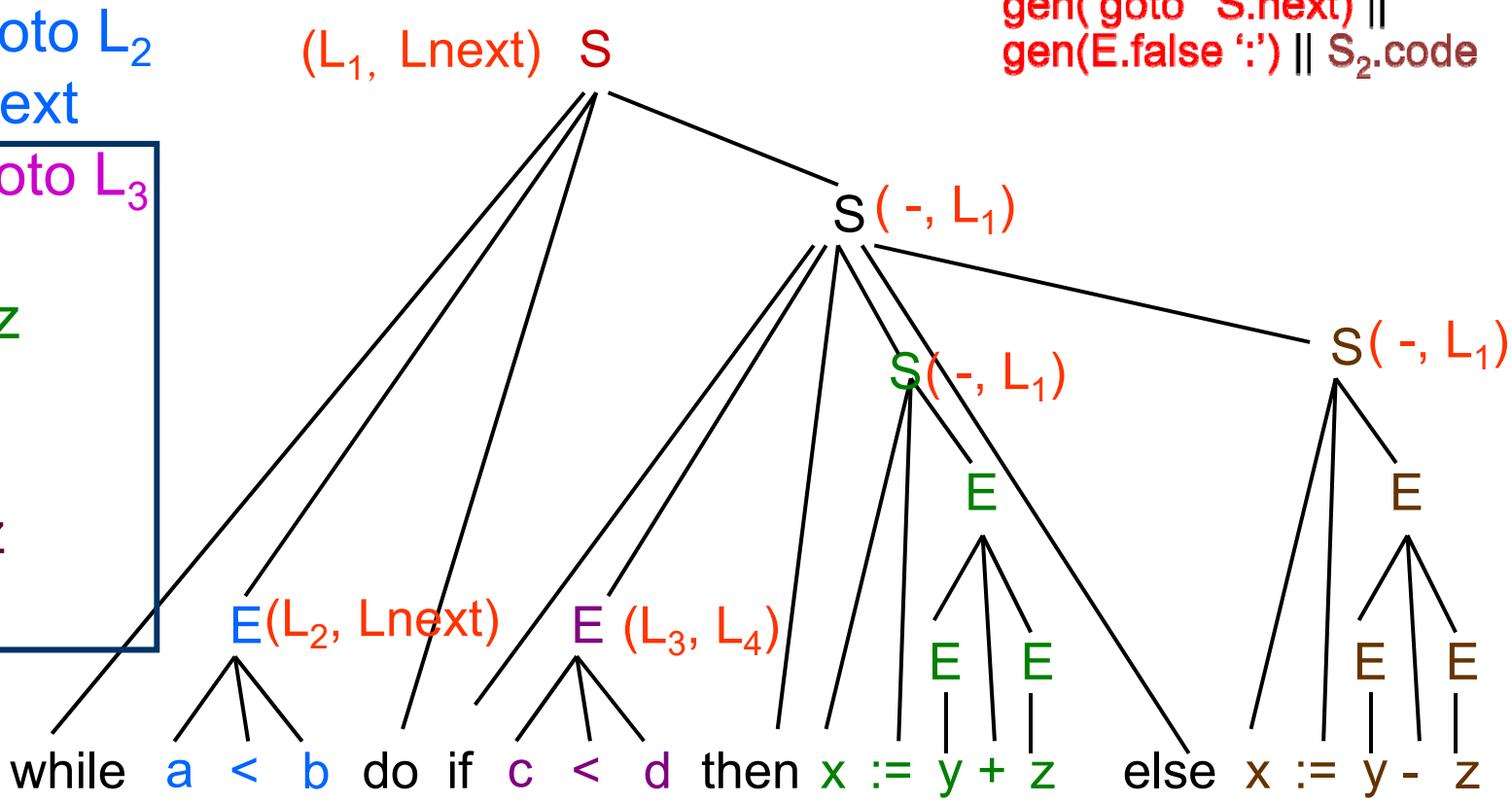
产生式

$S \rightarrow$ if E then S_1 else S_2

语义规则

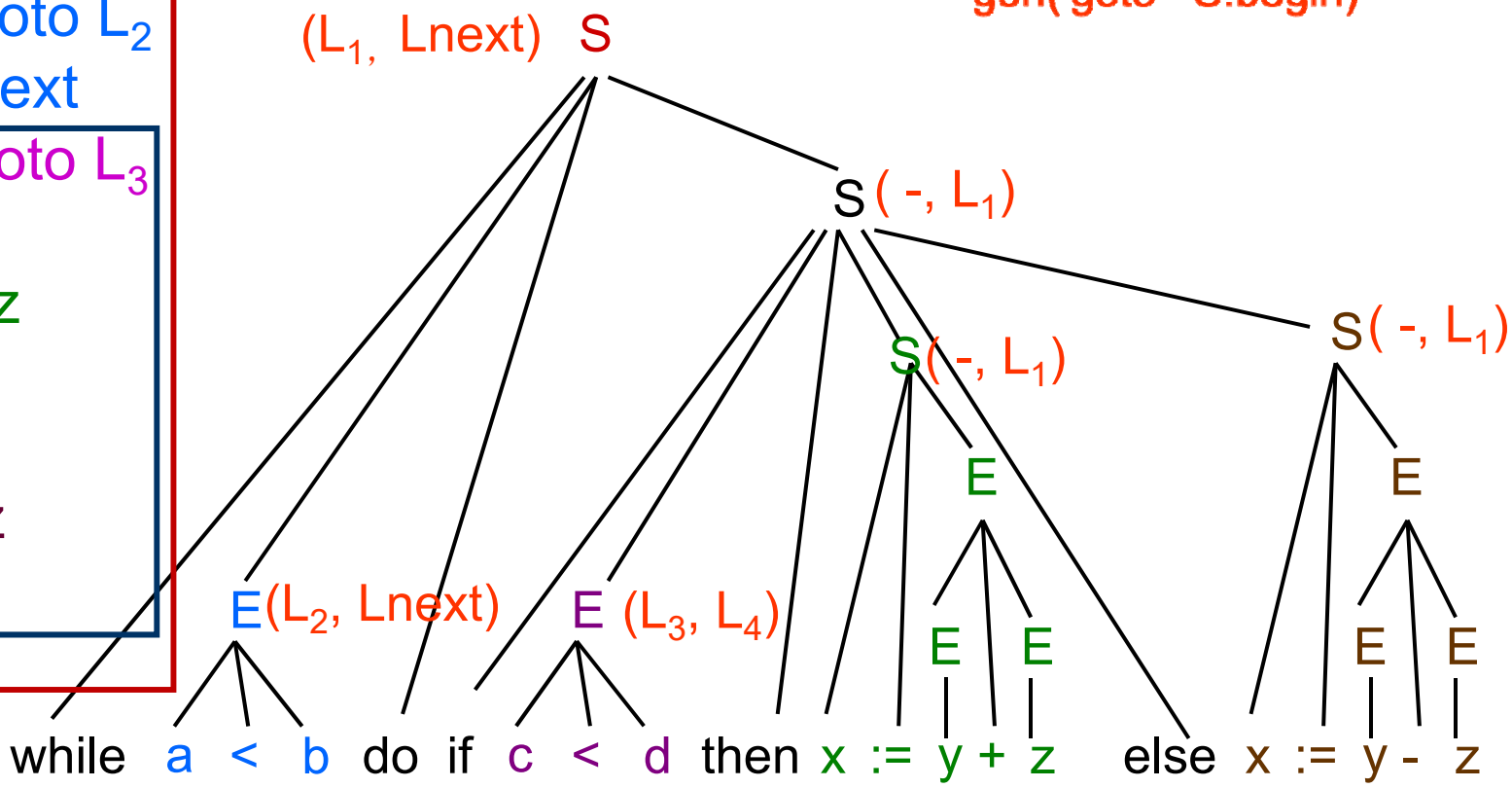
E.true:=newlabel;
E.false:=newlabel;
S_1.next:=S.next
S_2.next:=S.next;
S.code:=E.code ||
gen(E.true ':') || S_1.code ||
gen('goto' S.next) ||
gen(E.false ':') || S_2.code

```
    if a<b goto L2
    goto Lnext
    if c<d goto L3
    goto L4
L3: T1 :=y+z
    x:=T1
    goto L1
L4: T2 :=y-z
    x:=T2
```



产生式	语义规则
$S \rightarrow$ while E do S_1	S.begin:=newlabel; E.true:=newlabel; E.false:=S.next; S_1.next:=S.begin; S.code:=gen(S.begin ':') || E.code || gen(E.true ':') || S_1.code || gen('goto' S.begin)

L_1: if a<b goto L_2
goto Lnext
L_2: if c<d goto L_3
goto L_4
L_3: T_1 :=y+z
x:=T_1
goto L_1
L_4: T_2 :=y-z
x:=T_2
goto L_1



根据属性文法翻译控制语句

- 根据属性文法翻译如下语句：

```
while a<b do
    if c<d then   x:=y+z
          else    x:=y-z
```

L_1: if a<b goto L_2
goto Lnext
L_2: if c<d goto L_3
goto L_4
L_3: T_1 :=y+z
x:=T_1
goto L_1
L_4: T_2 :=y-z
x:=T_2
goto L_1

编译原理

一遍扫描翻译控制语句

控制语句的文法

(1) S → if E then S
(2) S → if E then S else S
(3) S → while E do S
(4) S → begin L end
(5) S → A
(6) L → L;S
(7) L → S

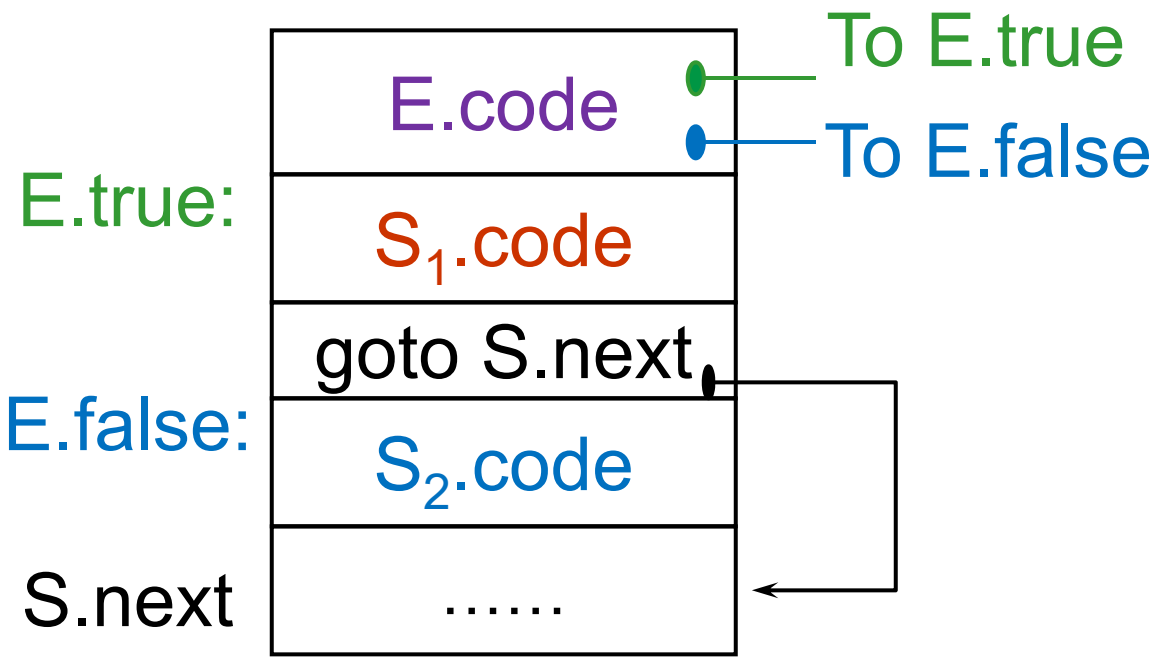
- S表示语句，L表示语句表，A为赋值语句，E表示布尔表达式

if 语句的文法

- 相关产生式
  $S \rightarrow \text{if } E \text{ then } S_1$
  $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

if 语句的文法

- 相关产生式
  $S \rightarrow$ if E then S_1
  $S \rightarrow$ if E then S_1 else S_2

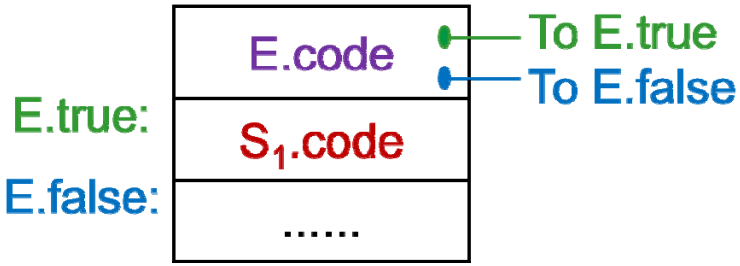


if 语句的文法

- 相关产生式
  $S \rightarrow$ if E then S_1
  $S \rightarrow$ if E then S_1 else S_2
- 改写后的产生式
  $S \rightarrow$ if E then M S_1
  $S \rightarrow$ if E then M_1 S_1 N else M_2 S_2
  $M \rightarrow \varepsilon$
  $N \rightarrow \varepsilon$

if 语句的翻译模式

1. S→if E then M S_1
{ backpatch(E.truelist, M.quad);
S.nextlist:=merge(E.falselist, S_1.nextlist) }

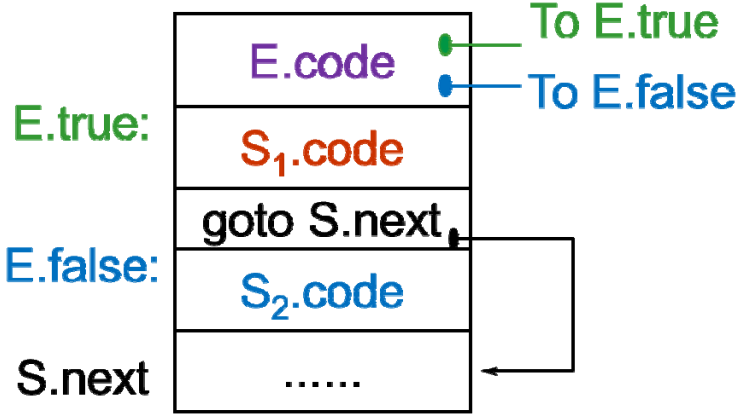


if-then的语义

2. S→if E then M_1 S_1 N else M_2 S_2
{ backpatch(E.truelist, M_1.quad);
backpatch(E.falselist, M_2.quad);
S.nextlist:=merge(S_1.nextlist, N.nextlist, S_2.nextlist) }

3. M→ε { M.quad:=nextquad }

4. N→ε { N.nextlist:=makelist(nextquad);
emit('j,－,－,－') }



if-then-else的语义

while语句的文法

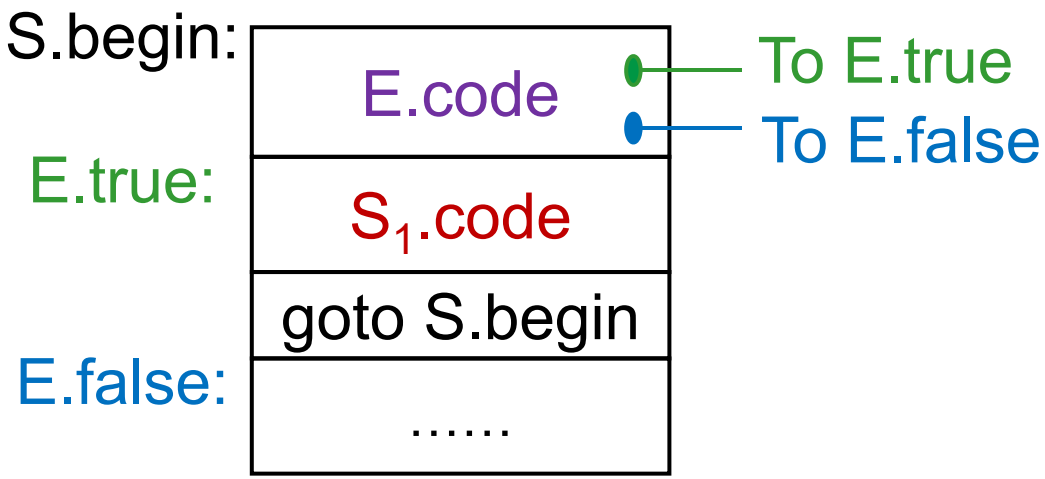
- 相关产生式

  $S \rightarrow$ while E do S_1

- 改写后的产生式

  $S \rightarrow$ while M_1 E do M_2 S_1

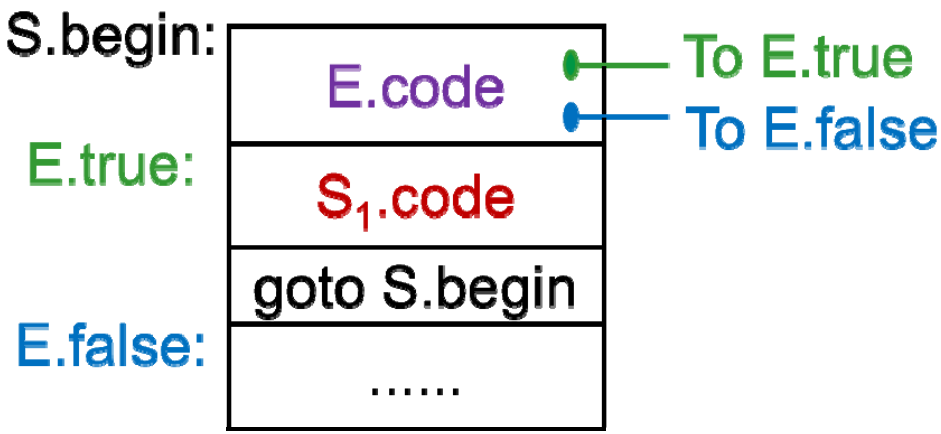
  $M \rightarrow \varepsilon$



while-do语句的翻译模式

1. $S \rightarrow$ while M_1 E do M_2 S_1
 { backpatch(E.truelist, M_2.quad);
 backpatch(S_1.nextlist, M_1.quad);
 S.nextlist := E.falselist;
 emit(‘j, - , - ,’ M_1.quad) }

2. $M \rightarrow \varepsilon$
 { M.quad := nextquad }



复合语句的文法

- 相关产生式

  $S \rightarrow$ begin L end

  $L \rightarrow L$; S | S

- 改写后的产生式

  $S \rightarrow$ begin L end

  $L \rightarrow L_1$; M S | S

  $M \rightarrow \varepsilon$

复合语句的翻译模式

1. $S \rightarrow$ begin L end
   { S.nextlist := L.nextlist }

2. $L \rightarrow L_1$; M S
   { backpatch(L_1.nextlist, M.quad);
     L.nextlist := S.nextlist }

3. $M \rightarrow \varepsilon$
   { M.quad := nextquad }

其它几个语句的翻译

1. S → A　　{ S.nextlist := makelist( ) }

2. L → S　　{ L.nextlist := S.nextlist }

编译原理

一遍扫描翻译控制语句示例

示例：翻译语句

- 将下面的语句翻译为四元式

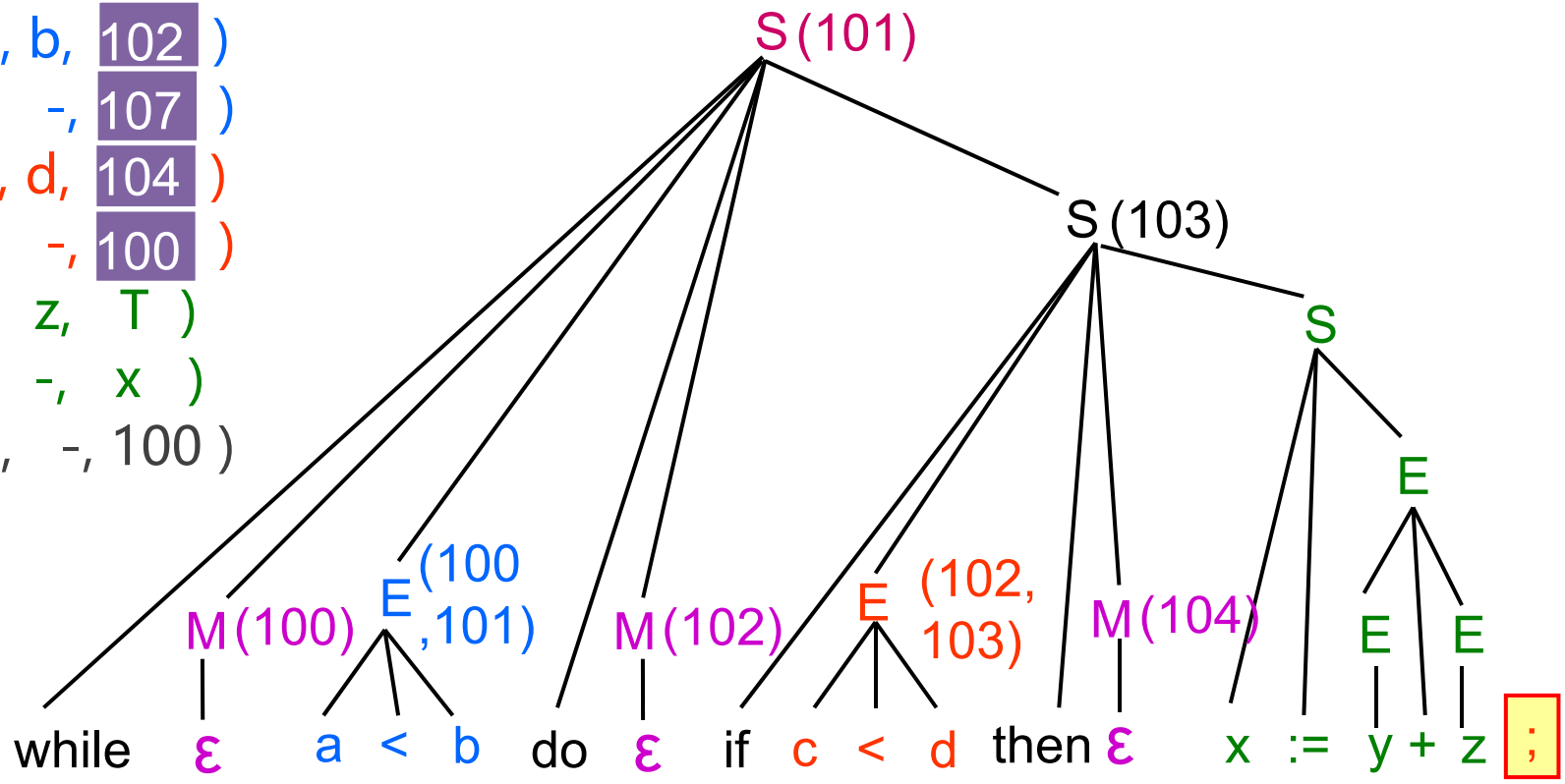
```
while a<b do
    if c<d then x:=y+z;
```

$E \rightarrow id_1$ relop id_2
{ E.truelist:=makelist(nextquad);
E.falselist:=makelist(nextquad+1);
emit('j' relop.op ',' id $_1$.place ',' id $_2$.place',' '0');
emit('j, - , - , 0') }

$M \rightarrow \varepsilon$ { M.quad:=nextquad }

do if c<d then x:=y+z;

100 (j<, a, b, 102)
101 (j, -, -, 107)
102 (j<, c, d, 104)
103 (j, -, -, 100)
104 (+, y, z, T)
105 (:=, T, -, x)
106 (j, -, -, 100)
107



小结

- 控制语句
  - S → if E then S
  - S → if E then S else S
  - S → while E do S
  - S → begin L end
- 控制语句的翻译
  - 用属性文法描述语义
  - 设计一遍扫描的翻译模式