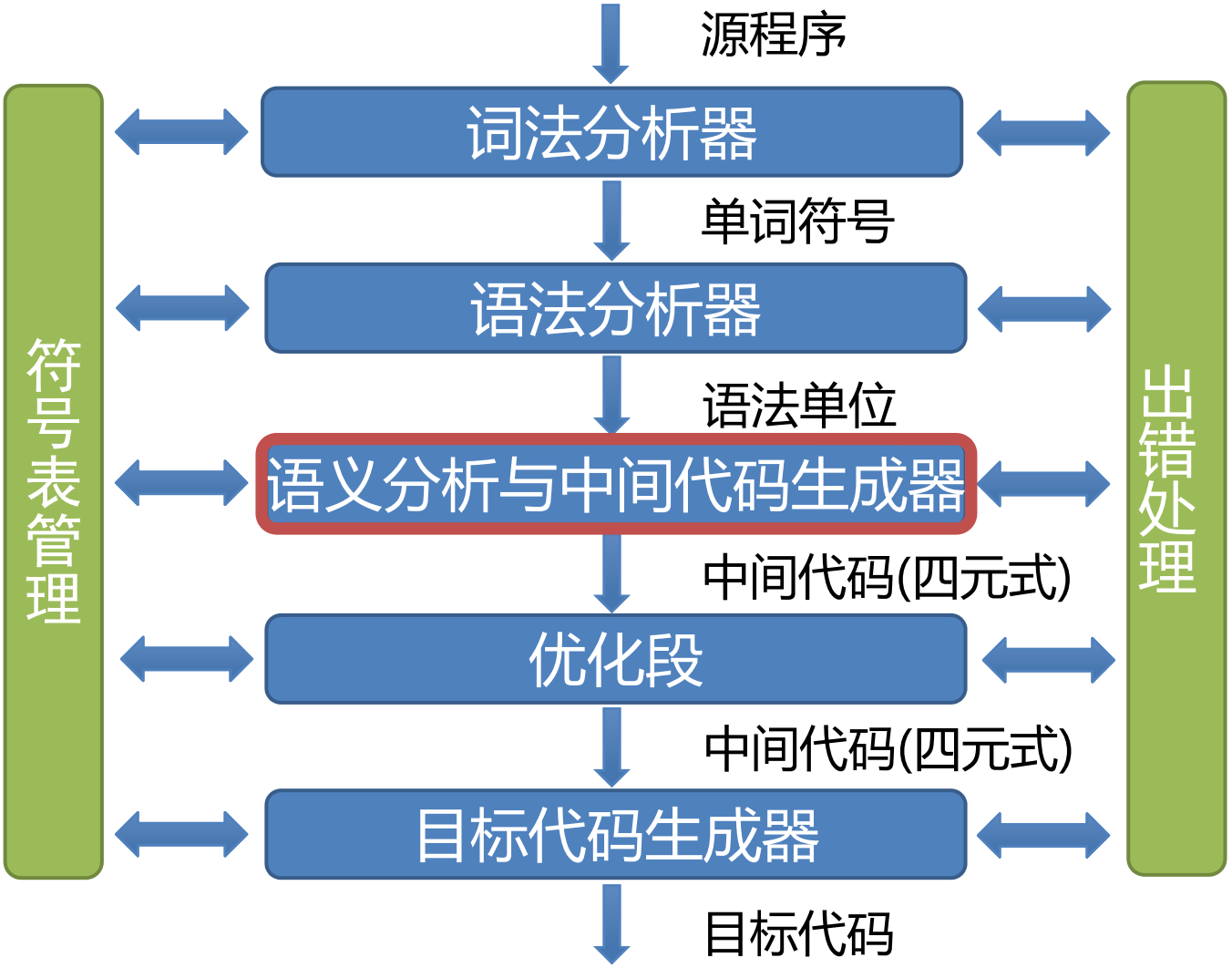
编译原理

布尔表达式的翻译

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

编译原理

布尔表达式的简介

布尔表达式及其用途

- 文法

  E → E or E | E and E | not E | (E) | i rop i | i

- 用途
  - 用于逻辑演算，计算逻辑值
  - 用于控制语句的条件式

计算布尔表达式的两种方法

- 数值表示法：如同计算算术表达式一样,一步步算

  1 or (not 0 and 0) or 0
  =1 or (1 and 0) or 0
  =1 or 0 or 0
  =1 or 0
  =1

- A or B and C>D翻译成

(1) (>, C, D, T1)
(2) (and, B, T1, T2)
(3) (or, A, T2, T3)

(1) T_1 := C>D
(2) T_2 := B and T_1
(3) T_3 := A or T_2

计算布尔表达式的两种方法

- 带优化的翻译法
  - 把A or B解释成　if A then true else B
  - 把A and B解释成　if A then B else false
  - 把not A解释成　if A then false else true
- 适合于作为条件表达式的布尔表达式使用

编译原理

按数值表示法翻译

数值表示法

- a or b and not c 翻译成

  T_1 := not c

  T_2 := b and T_1

  T_3 := a or T_2

- a<b的关系表达式可等价地写成

  if a<b then 1 else 0， 翻译成

  100: if a<b goto 103

  101: T := 0

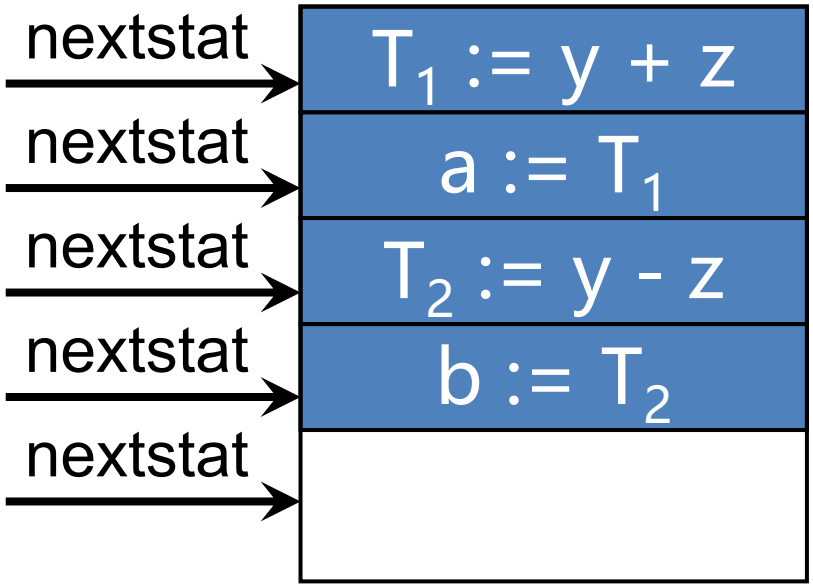
  102: goto 104

  103: T := 1

  104:

关于布尔表达式的数值表示法的翻译模式

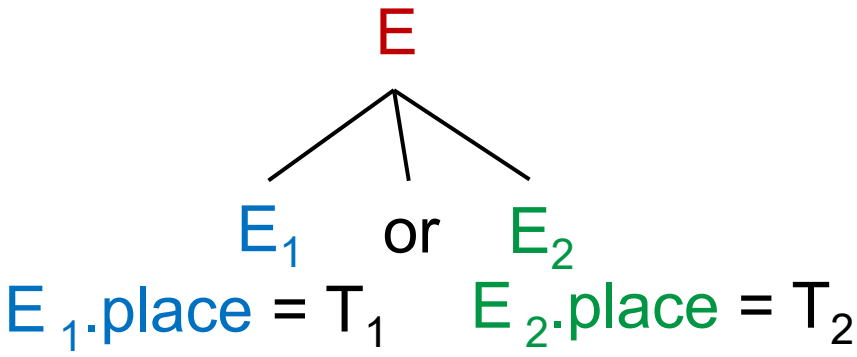
- 过程emit将三地址代码送到输出文件中
- nextstat：输出序列中下一条三地址语句的地址索引
- 过程emit每产生一条指令，nextstat加1

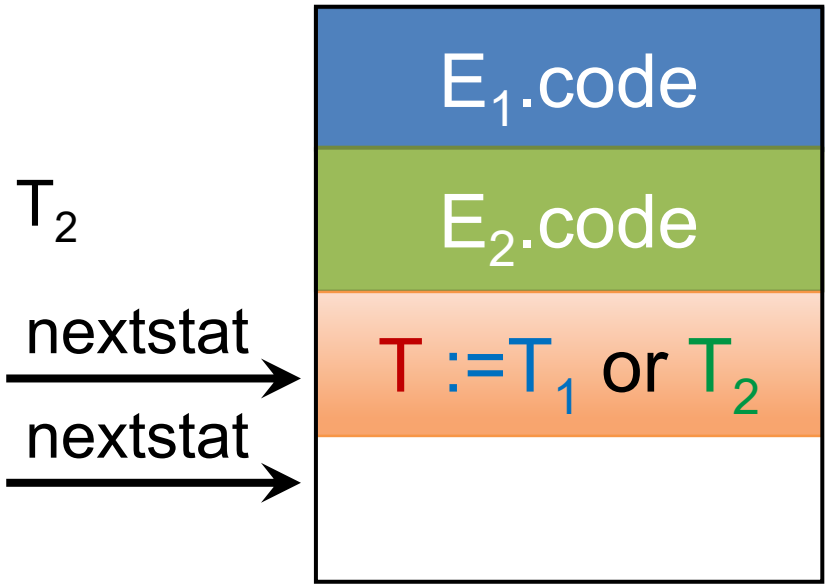


关于布尔表达式的数值表示法的翻译模式

$E \rightarrow E_1$ or E_2

{ E.place:=newtemp;
  emit(E.place ':=' E_1.place 'or' E_2.place)}





关于布尔表达式的数值表示法的翻译模式

$E \rightarrow E_1$ or E_2

{ E.place:=newtemp;
emit(E.place ':=' E_1.place 'or' E_2.place)}

$E \rightarrow E_1$ and E_2

{ E.place:=newtemp;
emit(E.place ':=' E_1.place 'and' E_2.place)}

$E \rightarrow$ not E_1

{ E.place:=newtemp;
emit(E.place ':=' 'not' E_1.place) }

$E \rightarrow (E_1)$ { E.place:=E_1.place }

关于布尔表达式的数值表示法的翻译模式

$E \rightarrow id_1$ relop id_2

{ E.place:=newtemp;

emit('if' id_1.place relop.op id_2.place 'goto' nextstat+3);

emit(E.place ':=' '0');

emit('goto' nextstat+2);

emit(E.place':=' '1') }

$E \rightarrow id$

{ E.place:=id.place }

a<b 翻译成

100: if a<b goto 103

101: T:=0

102: goto 104

103: T:=1

104:

布尔表达式a<b or c<d and e<f的翻译结果

100: if a<b goto 103
101: T_1:=0
102: goto 104
103: T_1:=1
104: if c<d goto 107
105: T_2:=0
106: goto 108
107: T_2:=1
108: if e<f goto 111
109: T_3:=0
110: goto 112
111: T_3:=1
112: T_4:=T_2 and T_3
113: T_5:=T_1 or T_4

$E \rightarrow id_1$ relop id_2
{ E.place:=newtemp;
emit('if' id_1.place relop. op id_2. place
'goto' nextstat+3);
emit(E.place ':=' '0');
emit('goto' nextstat+2);
emit(E.place':=' '1') }

$E \rightarrow id$
{ E.place:=id.place }

$E \rightarrow E_1$ or E_2
{ E.place:=newtemp;
emit(E.place ':=' E_1.place 'or' E_2.place)}

$E \rightarrow E_1$ and E_2
{ E.place:=newtemp;
emit(E.place ':=' E_1.place 'and' E_2.place) }

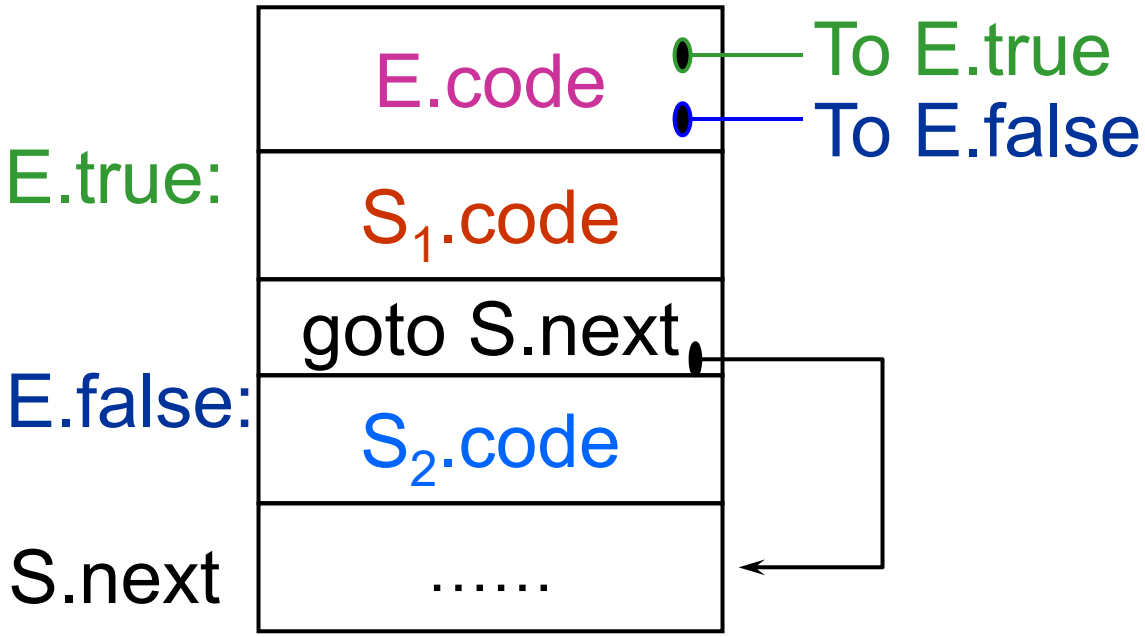
编译原理

带优化的翻译

作为条件控制的布尔式翻译

- 条件语句 if E then S_1 else S_2

  赋予 E 两种出口:一真一假



条件语句的翻译

- if a>c or b <d then S_1 else S_2 翻译成三地址代码

```
        if a>c goto L2      “真”出口
        goto L1
L1:     if b<d goto L2      “真”出口
        goto L3             “假”出口
L2:     (关于S1的三地址代码序列)
        goto Lnext
L3:     (关于S2的三地址代码序列)
Lnext:
```

编译原理

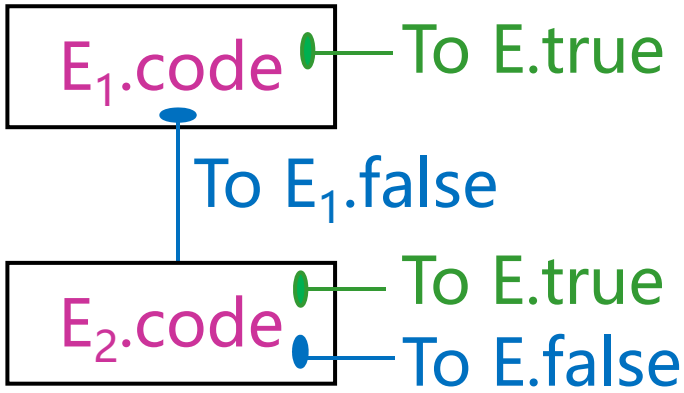
布尔表达式的属性文法

产生布尔表达式三地址代码的属性文法

- 语义函数newlabel，返回一个新的符号标号
- 对于一个布尔表达式E，设置两个继承属性
  - E.true是E为‘真’时控制流转向的标号
  - E.false是E为‘假’时控制流转向的标号
- E.code记录E生成的三地址代码序列

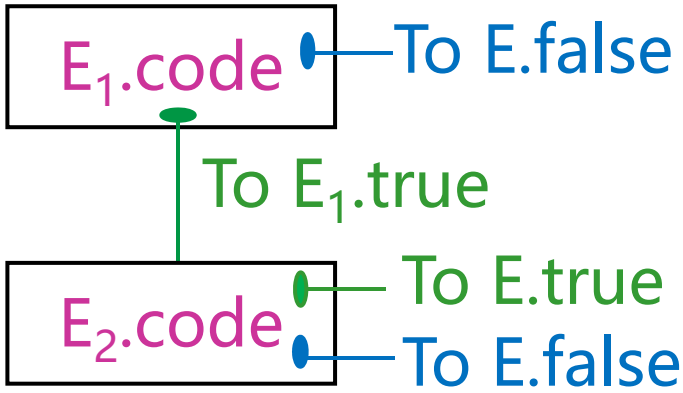
产生布尔表达式三地址代码的属性文法

产生式	语义规则
$E \rightarrow E_1$ or E_2	E_1.true:=E.true; E_1.false:=newlabel; E_2.true:=E.true; E_2.false:=E.false; E.code:=E_1.code || gen(E_1.false ':') || E_2.code



产生布尔表达式三地址代码的属性文法

产生式	语义规则
$E \rightarrow E_1$ and E_2	E_1.true:=newlabel; E_1.false:=E.false; E_2.true:=E.true; E_2.false:=E.fasle; E.code:=E_1.code || gen(E_1.true ':') || E_2.code



产生布尔表达式三地址代码的属性文法

产生式	语义规则
$E \rightarrow \text{not } E_1$	$E_1.true := E.false$; $E_1.false := E.true$; $E.code := E_1.code$
$E \rightarrow (E_1)$	$E_1.true := E.true$; $E_1.false := E.false$; $E.code := E_1.code$

产生布尔表达式三地址代码的属性文法

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)
E→true	E.code:=gen(‘goto’ E.true)
E→false	E.code:=gen(‘goto’ E.false)

根据属性文法翻译布尔表达式

- 根据属性文法翻译如下布尔表达式：

  a<b or c<d and e<f

  假定整个表达式的真假出口已分别置为Ltrue和Lfalse。

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)
$E \rightarrow E_1$ or E_2	E_1.true:=E.true; E_1.false:=newlabel; E_2.true:=E.true; E_2.false:=E.false; E.code:=E_1.code || gen(E_1.false ':') || E_2.code
$E \rightarrow E_1$ and E_2	E_1.true:=newlabel; E_1.false:=E.false; E_2.true:=E.true; E_2.false:=E.fasle; E.code:=E_1.code || gen(E_1.true ':') || E_2.code

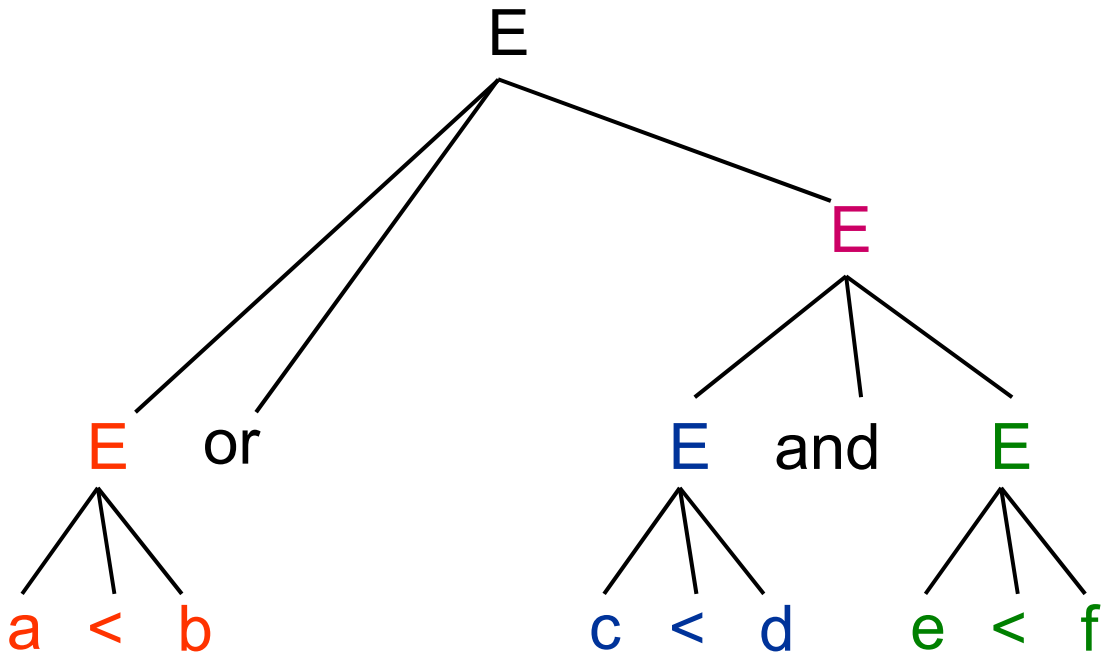
根据属性文法翻

产生式	语义规则
$E \rightarrow E_1$ or E_2	E_1.true:=E.true; E_1.false:=newlabel; E_2.true:=E.true; E_2.false:=E.false; E.code:=E_1.code || gen(E_1.false ':') || E_2.code
$E \rightarrow E_1$ and E_2	E_1.true:=newlabel; E_1.false:=E.false; E_2.true:=E.true; E_2.false:=E.fasle; E.code:=E_1.code || gen(E_1.true ':') || E_2.code

- 根据属性文法翻译如

a<b or c<d and e<f

假定整个表达式的真假出口已分别置为Ltrue和Lfalse。



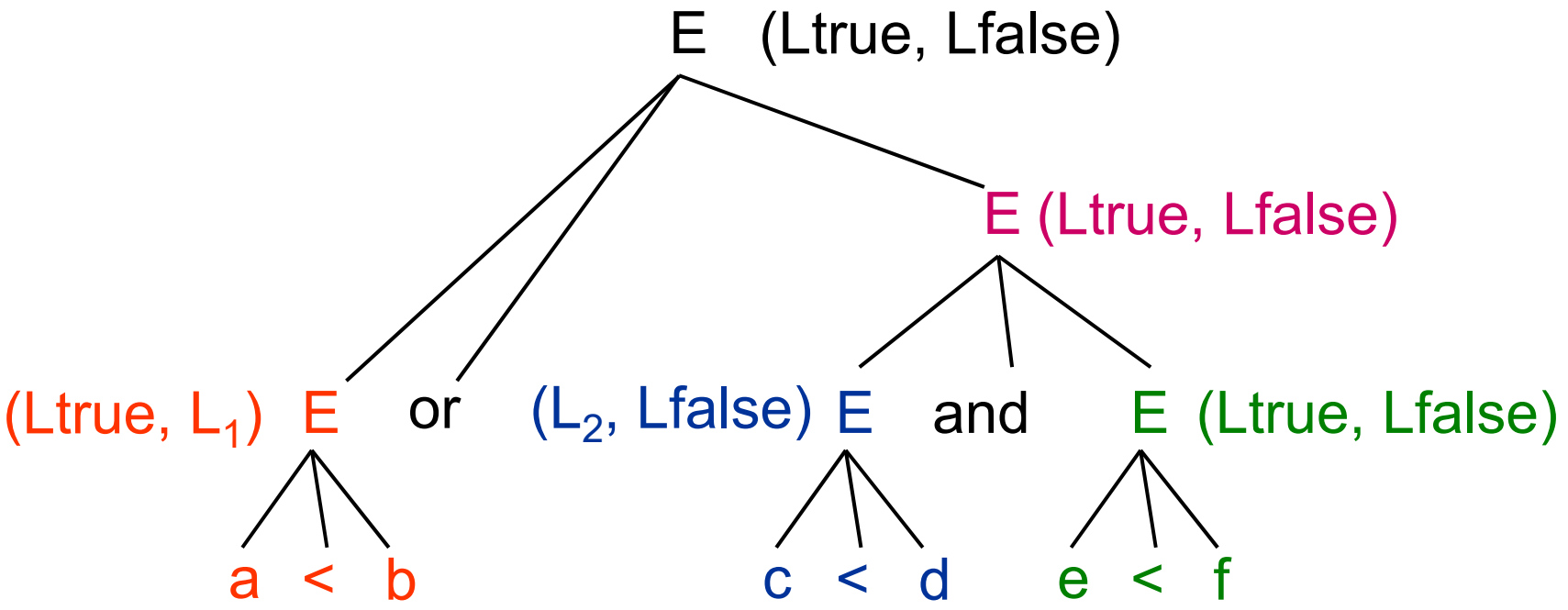
根据属性文法翻

产生式	语义规则
$E \rightarrow E_1$ or E_2	E_1.true:=E.true; E_1.false:=newlabel; E_2.true:=E.true; E_2.false:=E.false; E.code:=E_1.code || gen(E_1.false ':') || E_2.code
$E \rightarrow E_1$ and E_2	E_1.true:=newlabel; E_1.false:=E.false; E_2.true:=E.true; E_2.false:=E.fasle; E.code:=E_1.code || gen(E_1.true ':') || E_2.code

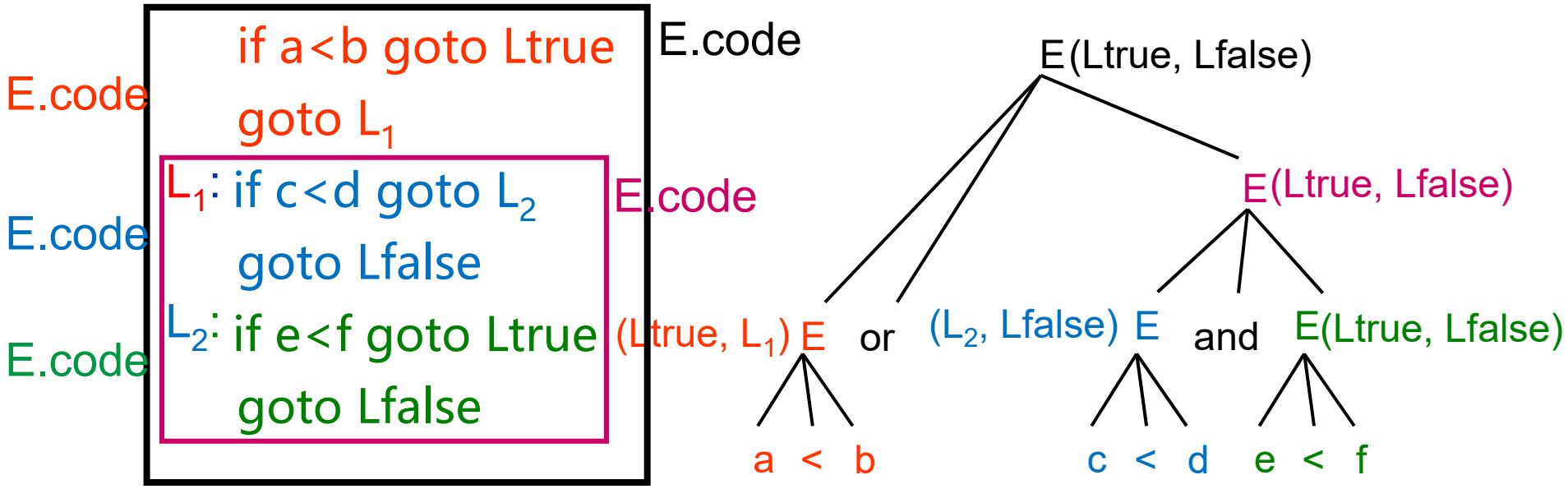
- 根据属性文法翻译如

a<b or c<d and e<f

假定整个表达式的真假出口已分别置为Ltrue和Lfalse。



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)*
$E \rightarrow E_1$ or E_2	E_1.true:=E.true; E_1.false:=newlabel; E_2.true:=E.true; E_2.false:=E.false; *$E.code:=E_1.code \|\| gen(E_1.false ':') \|\| E_2.code$*
$E \rightarrow E_1$ and E_2	E_1.true:=newlabel; E_1.false:=E.false; E_2.true:=E.true; E_2.false:=E.fasle; *$E.code:=E_1.code \|\| gen(E_1.true ':') \|\| E_2.code$*



布尔表达式的翻译

- 两(多)遍扫描
  - 为给定的输入串构造一棵语法树
  - 遍历语法树，进行语义规则中规定的翻译
- 一遍扫描

if a<b goto Ltrue

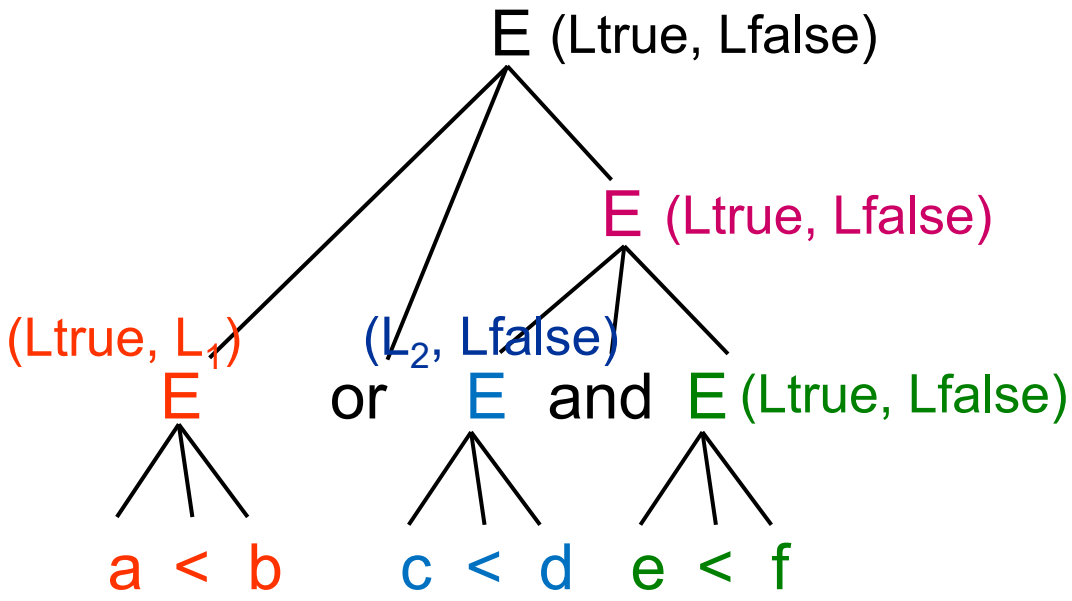
goto L_1

L_1: if c<d goto L_2.

goto Lfalse

L_2: if e<f goto Ltrue

goto Lfalse



编译原理

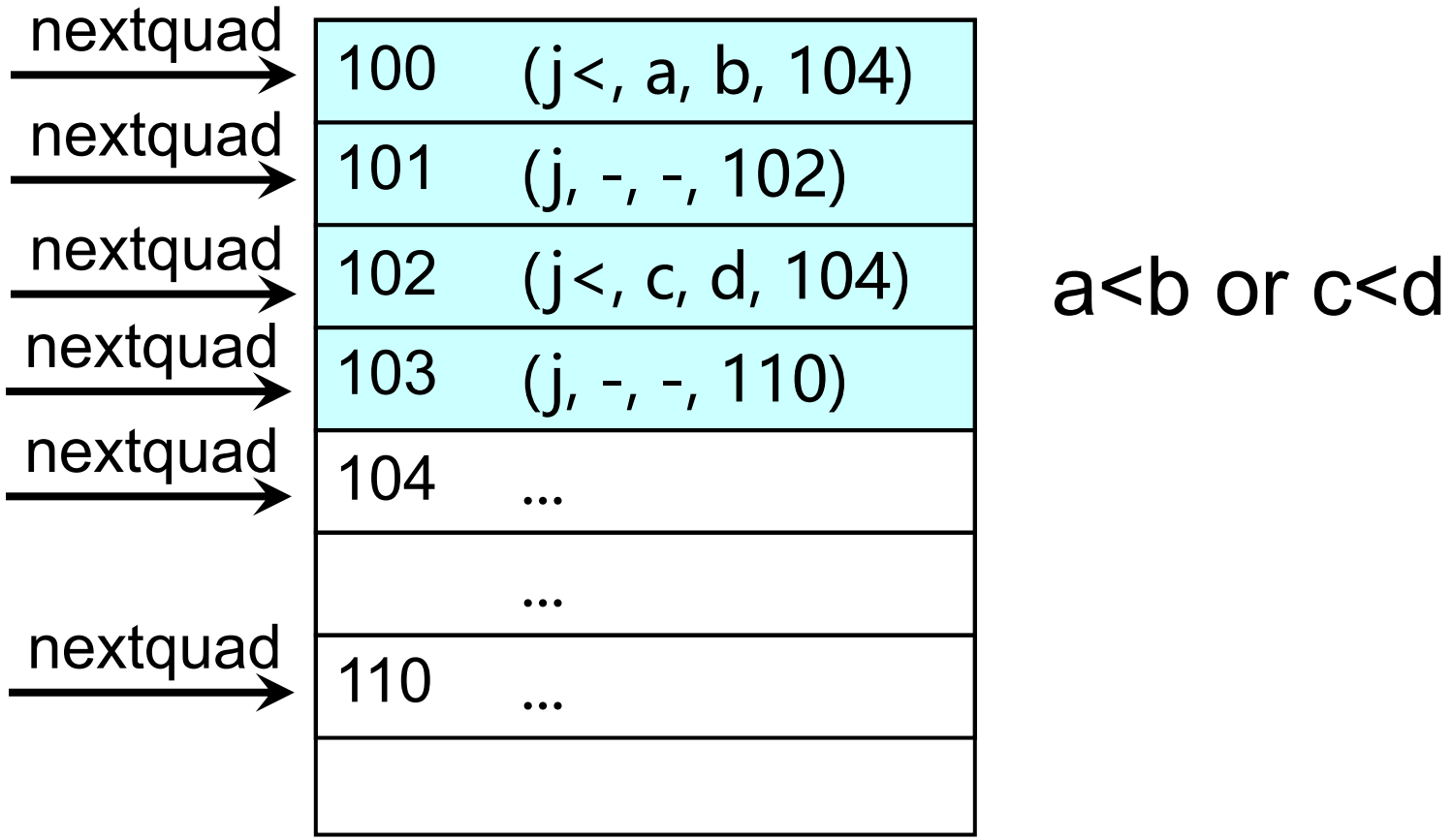
布尔表达式的一遍扫描翻译模式

一遍扫描实现布尔表达式的翻译

- 以四元式为中间语言
- 四元式存入一个数组中，数组下标代表四元式的标号
- 约定
  - 四元式(jnz, a, -, p)　表示　if a goto p
  - 四元式(jrop, x, y, p)　表示　if x rop y goto p
  - 四元式(j, -, -, p)　表示　goto p

一遍扫描实现布尔表达式的翻译

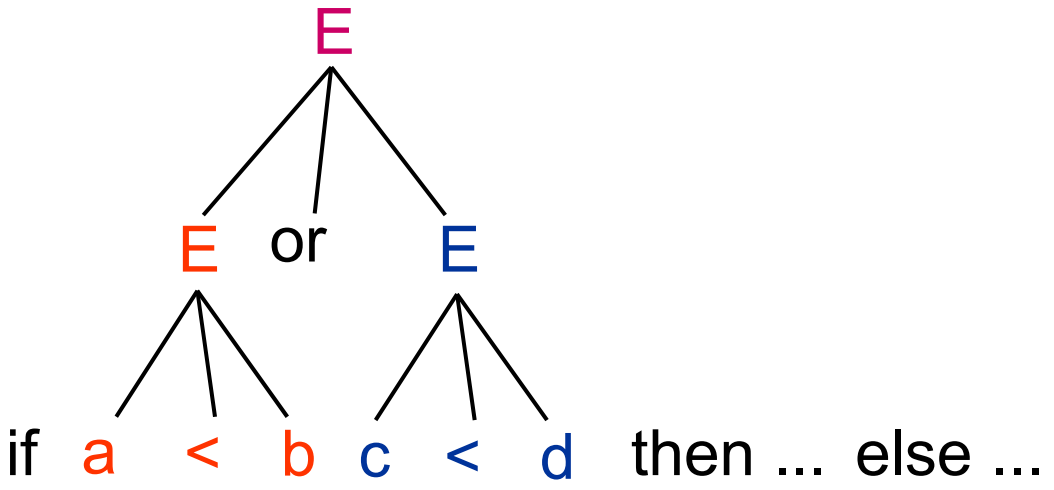
- 过程emit将四元式代码送到输出数组中



一遍扫描实现布尔表达式的翻译

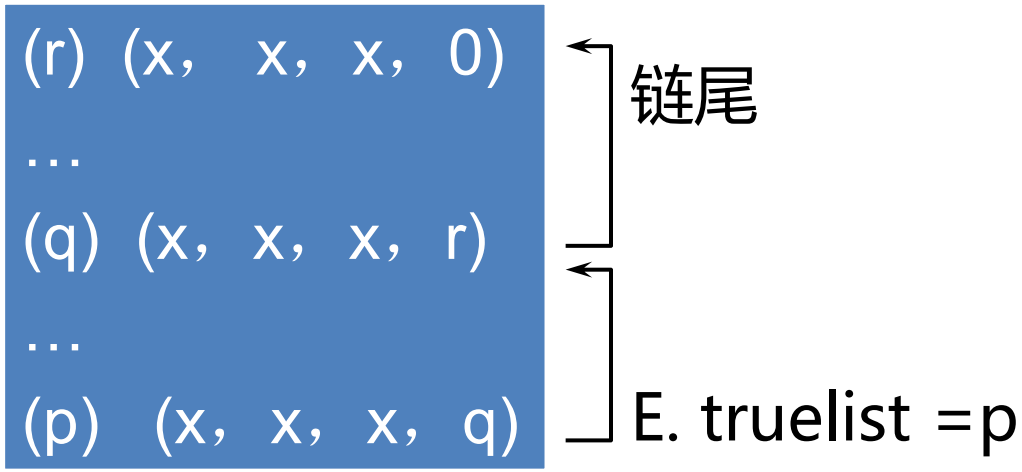
- 产生跳转四元式时，它的转移地址无法立即知道
- 需要以后扫描到特定位置时才能回过头来确定
- 把这些未完成的四元式地址作为E的语义值保存,待机“回填”

100	(j<, a, b, 104)
101	(j, -, -, 102)
102	(j<, c, d, 104)
103	(j, -, -, 110)
104	...
	...
110	...



一遍扫描实现布尔表达式的翻译

- 为非终结符E赋予两个综合属性E.truelist和E.falselist。它们分别记录布尔表达式E所对应的四元式中需回填“真”、“假”出口的四元式的标号所构成的链表
- 例如，假定E的四元式中需要回填"真"出口的p，q，r三个四元式，则E.truelist为下列链:

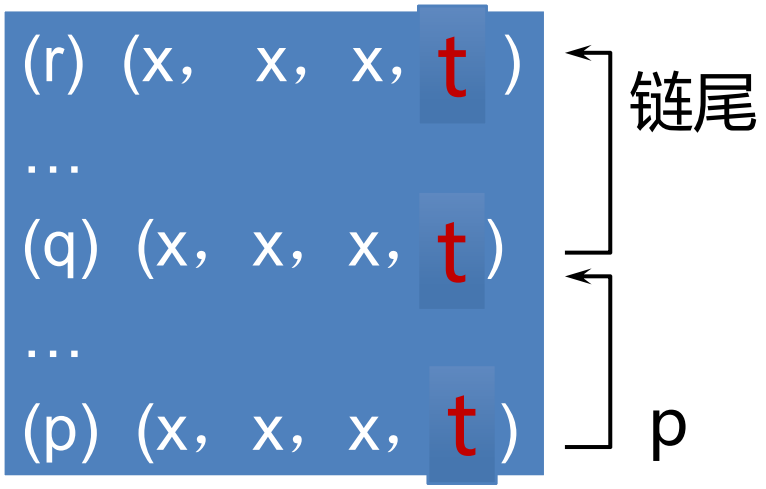


一遍扫描实现布尔表达式的翻译

- 引入语义变量和过程
  - 变量nextquad，它指向下一条将要产生但尚未形成的四元式的地址(标号)。nextquad的初值为1，每当执行一次emit之后，nextquad将自动增1。
  - 函数makelist(i)，它将创建一个仅含i的新链表，其中i是四元式数组的一个下标(标号)；函数返回指向这个链的指针。
  - 函数merge(p_1,p_2)， 把以p_1和p_2为链首的两条链合并为一，作为函数值，回送合并后的链首。
  - 过程backpatch(p, t)，其功能是完成“回填”，把p所链接的每个四元式的第四区段都填为t。

一遍扫描实现布尔表达式的翻译

- 引入语义变量和过程
  - 过程backpatch(p, t)，其功能是完成“回填”，把p所链接的每个四元式的第四区段都填为t。



布尔表达式的文法

- $E \rightarrow E_1$ or E_2
- $E \rightarrow E_1$ and E_2

布尔表达式的文法

(1) $E \rightarrow E_1$ or M E_2

(2) $E \rightarrow E_1$ and M E_2

(3) $E \rightarrow$ not E_1

(4) $E \rightarrow (E_1)$

(5) $E \rightarrow id_1$ relop id_2

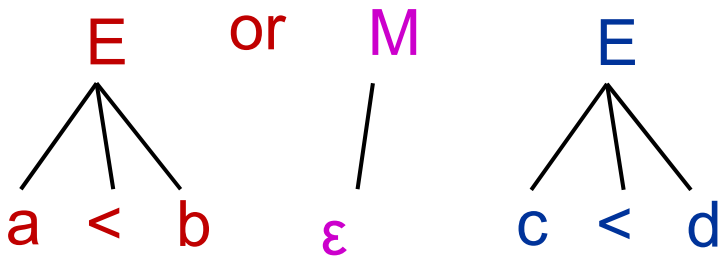
(6) $E \rightarrow$ id

(7) $M \rightarrow \varepsilon$

布尔表达式的翻译模式

(7) M→ε　　{ M.quad:=nextquad }

(1) E → E_1 or M E_2
(2) E → E_1 and M E_2
(3) E → not E_1
(4) E → (E_1)
(5) E → id_1 relop id_2
(6) E → id



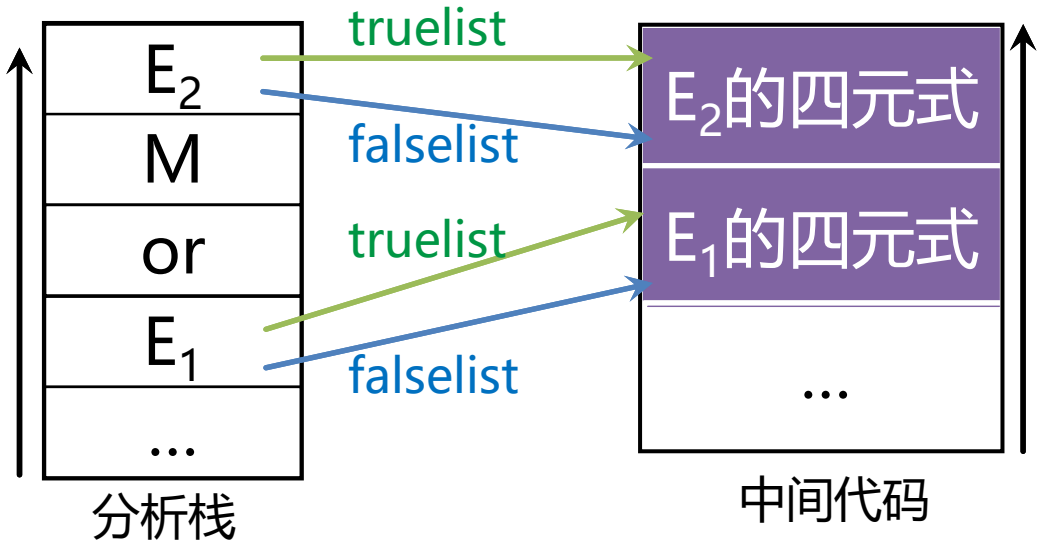
布尔表达式的翻译模式

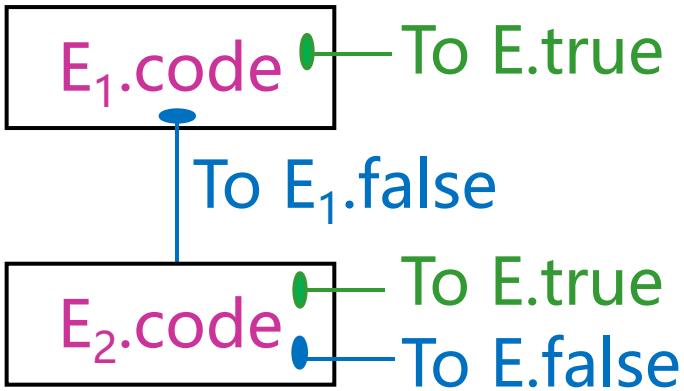
(1) $E \rightarrow E_1$ or M E_2

{ backpatch(E_1.falselist, M.quad);

E.truelist:=merge(E_1.truelist, E_2.truelist);

E.falselist:=E_2.falselist }





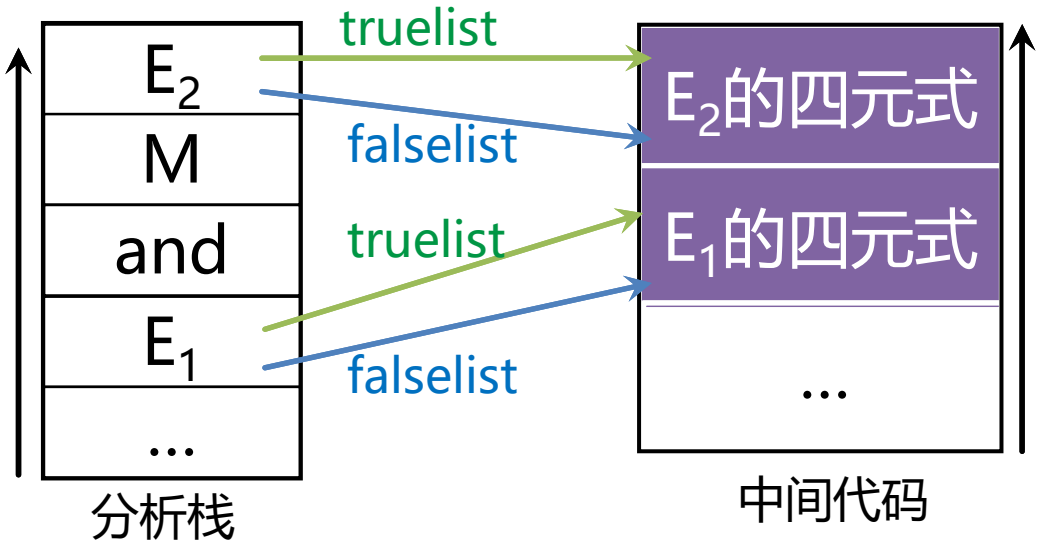
布尔表达式的翻译模式

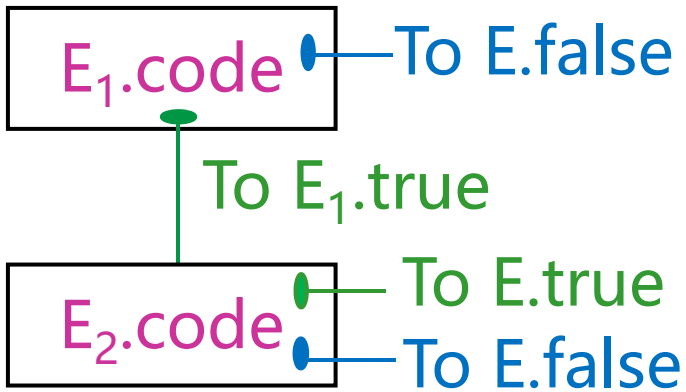
(2) $E \rightarrow E_1$ and M E_2

{ backpatch(E_1.truelist, M.quad);

E.truelist:=E_2.truelist;

E.falselist:=merge(E_1.falselist,E_2.falselist) }





布尔表达式的翻译模式

(3) $E \rightarrow \text{not } E_1$

{ E.truelist:=E_1.falselist;

E.falselist:=E_1.truelist}

(4) $E \rightarrow (E_1)$

{ E.truelist:=E_1.truelist;

E.falselist:=E_1.falselist}

布尔表达式的翻译模式

(5) $E \rightarrow id_1$ 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') }

(6) E→id

{ E.truelist:=makelist(nextquad);
E.falselist:=makelist(nextquad+1);
emit('jnz' ',' id .place ',' ' - ' ',' '0');
emit(' j, -, -, 0') }

布尔表达式的翻译模式

(1) $E \rightarrow E_1$ or M E_2

(2) $E \rightarrow E_1$ and M E_2

(3) $E \rightarrow$ not E_1

(4) $E \rightarrow (E_1)$

(5) $E \rightarrow id_1$ relop id_2

(6) $E \rightarrow$ id

(7) $M \rightarrow \varepsilon$

作为整个布尔表达式的“真”、“假”出口的四元式的转移目标仍待回填，分别记录在E.truelist和E.falselist中。

a<b or c<d and e<f

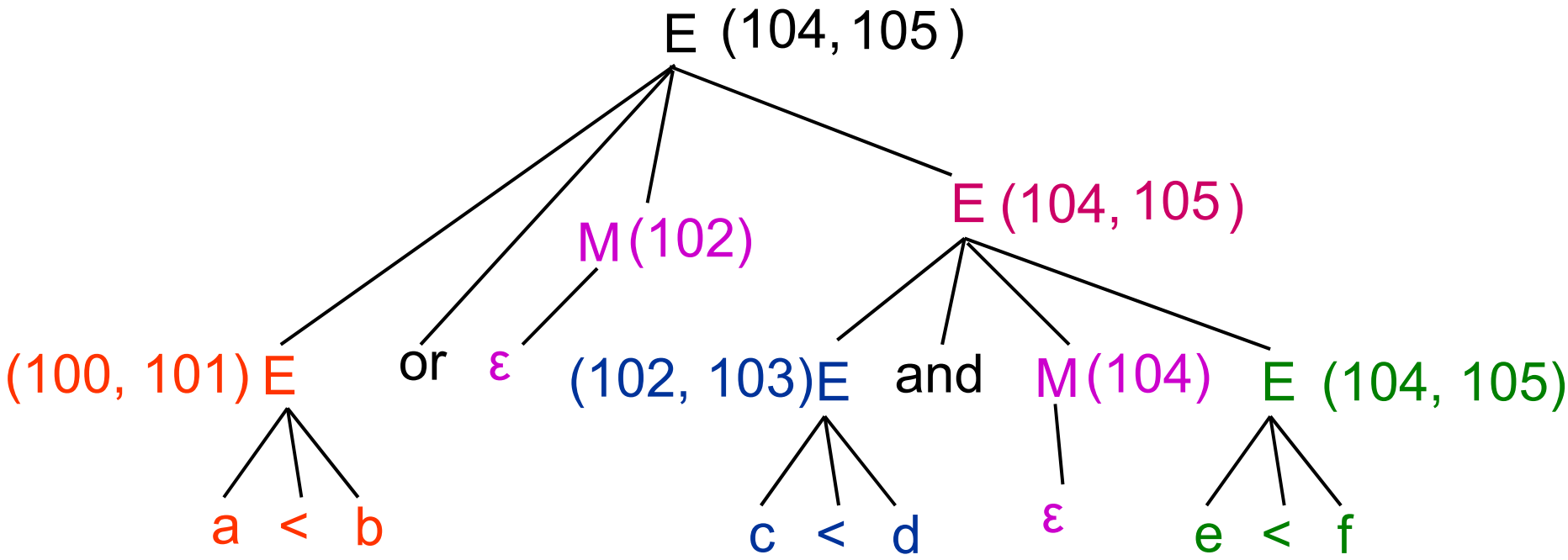
100	(j<, a, b, 0)
101	(j, -, -, 102)
102	(j<, c, d, 104)
103	(j, -, -, 0)
104	(j<, e, f, 100)
105	(j, -, -, 103)

(3) $E \rightarrow E_1$ or M E_2
{ backpatch(E_1.falselist, M.quad);
E.truelist := merge(E_1.truelist, E_2.truelist);
E.falselist := E_2.falselist }

(4) $E \rightarrow E_1$ and M E_2
{ backpatch(E_1.truelist, M.quad);
E.truelist := E_2.truelist;
E.falselist := merge(E_1.falselist, E_2.falselist) }

(5) E → id relop id
{ E.truelist := makelist(nextquad);
E.falselist := makelist(nextquad+1);
emit('j' relop.op id.addr id.addr '0');
emit('j, - , - , 0') }

(7) M→ε { M.quad:=nextquad }



小结

- 布尔表达式的翻译
  - 按数值表示法的表达式翻译
  - 带优化的表达式翻译
  - 语义的属性文法描述
  - 一遍扫描的翻译模式