国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle

编 译 原 理

第六章 中间代码生成（3）

郑馥丹
zhengfd5@mail.sysu.edu.cn

CONTENTS

目 录

1. 布尔表达式

- 布尔表达式的用途
  - 计算逻辑值
  - 控制流：作为控制语句(如if-then,while)的条件表达式
- 形式
  - **E→E or E | E and E | not E | (E) | id rop id | true | false**
  - 布尔算符的优先顺序（从高到低）为：not, and, or，且and和or都服从左结合，not服从右结合
  - rop是关系算符（<=, <, = , !=, >, >= 等）；id rop id是关系式，关系式中的id是算术量。**关系算符的优先级都相同，而且高于任何布尔算符，低于任何算术算符。**

1. 布尔表达式

- 布尔表达式的计算方法
  - 数值表示的直接计算
    - ✓ 1 or 0 and 1 = 1 or 0 = 1
  - 逻辑表示的短路计算
    - ✓ 布尔表达式计算到某一部分就可以得到结果，而**无需对布尔表达式进行完全计算**(作为条件控制的情况)，可以用if-then-else来解释：
      - A or B　　if A then 1 else B
      - A and B　if A then B else 0
      - not A　　if A then 0 else 1

2. 直接计算的翻译

- 如：A or B and not C被翻译成：

(not, C, - , t1)

(and, B, t1, t2)

(or, A, t2, t3)

- 对关系表达式a<b，可翻译成如下固定的三地址代码（四元式）序列：

a<b 等价于 if a<b then 1 else 0

(1) if a<b then goto (4)
(2) t:=false
(3) goto (5)
(4) t:=true
(5) ……

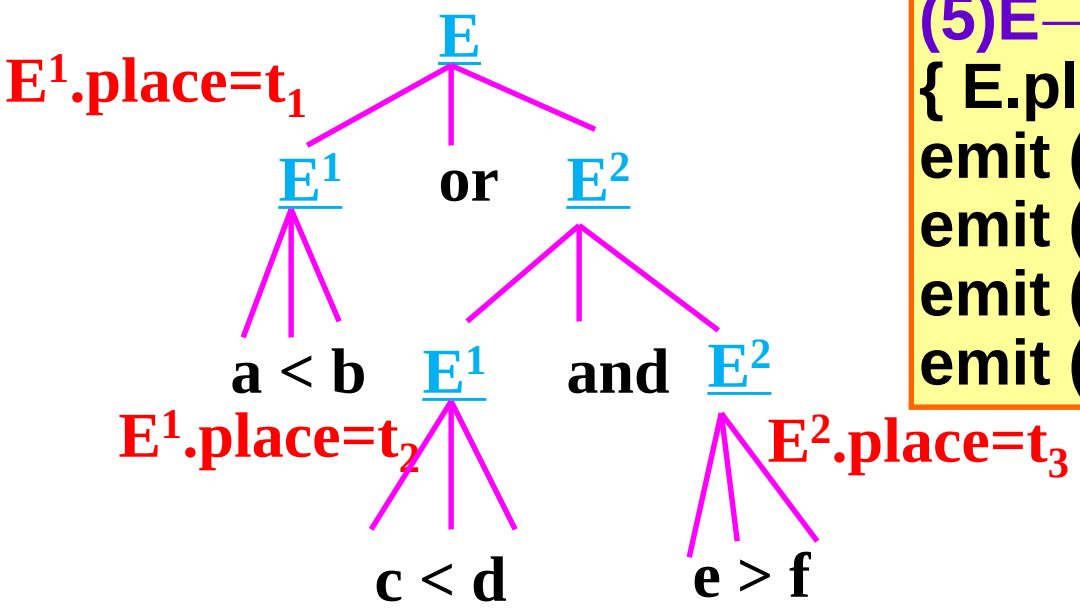
(1) (j<, a, b, (4))
(2) (:=, 0, - , t1)
(3) (jump, - , - , (5))
(4) (:=, 1, - , t1)
(5) …

2. 直接计算的翻译

(1) $E \rightarrow E^1$ or E^2	{ E.place : = newtemp ;
	emit (or , E^1.place , E^2.place , E.place) }
(2) $E \rightarrow E^1$ and E^2	{ E.place : = newtemp ;
	emit (and , E^1.place , E^2.place , E.place) }
(3) $E \rightarrow$ not E^1	{ E.place : = newtemp ;
	emit (not , E^1.place ,—, E.place) }
(4) $E \rightarrow (E^1)$	{ E.place : = E^1.place }
(5) $E \rightarrow id_1$ rop id_2	{ E.place : = newtemp ;
(1) (j<, a, b, (4)) (2) (:=, 0, - , t_1) (3) (jump, - , - , (5)) (4) (:=, 1, - , t_1) (5) …	emit (jrop , id_1.place , id_2.place , nextstat+3) ; emit (: = , 0 , - , E.place) ; emit (jump ,—,—, nextstat+2) ; emit (: = , 1 , - , E.place) }
(6) E→true	{ E.place: = newtemp;emit(:=,1,- ,E.place) }
(7) E→false	{E.place:=newtemp;emit(:=,0,- ,E.place)}

2. 直接计算的翻译

- 例：布尔表达式a<b or c<d and e>f的翻译

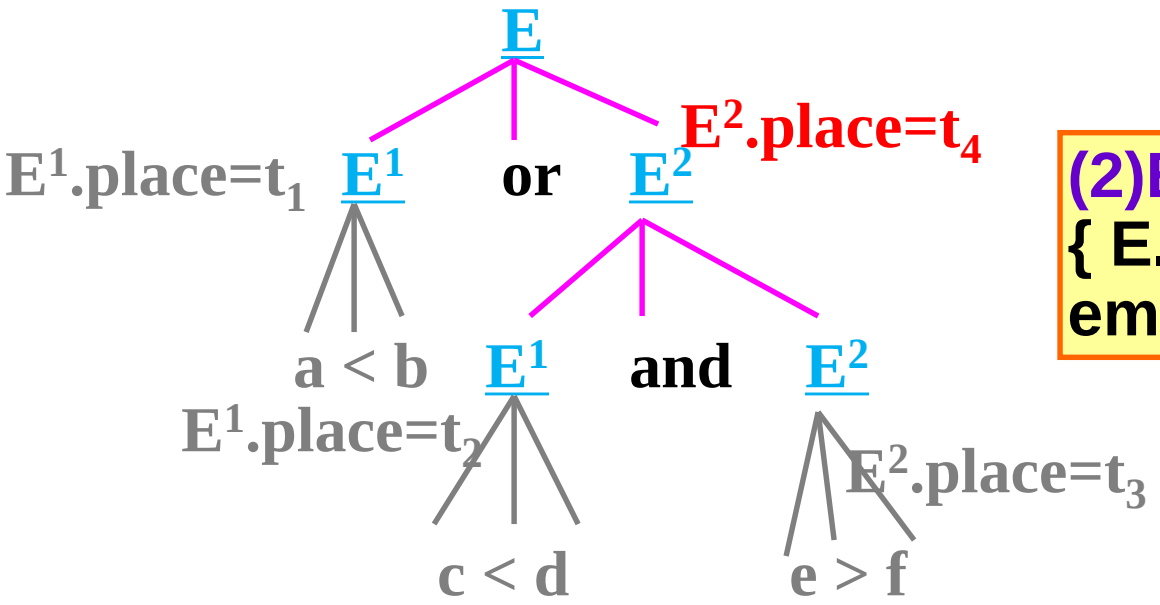


(5)$E \rightarrow id_1$ rop id_2
{ E.place : = newtemp ;
emit (jrop, id_1.place, id_2.place, nextstat+3);
emit (: = , 0 , - , E.place) ;
emit (jump , - , - , nextstat+2) ;
emit (: = , 1 , - , E.place) }

(1)(j<, a, b, (4))
(2)(:=, 0, - , t_1)
(3)(jump,- ,- ,(5))
(4)(:=, 1, - , t_1)
(5)(j<, c, d, (8))
(6)(:=, 0, - , t_2)
(7)(jump, - , - ,(9))
(8)(:=, 1, - , t_2)
(9)(j>, e, f, (12))
(10)(:=, 0, - , t_3)
(11)(jump,- ,- ,(13))
(12)(:=, 1, - , t_3)

2. 直接计算的翻译

- 例：布尔表达式a<b or c<d and e>f的翻译



(2)E→E1 and E2
{ E.place : = newtemp ;
emit(and, E1.place, E2.place , E.place)}

(1)(j<, a, b, (4))

(2)(:=, 0, - , t_1)

(3)(jump,- ,- ,(5))

(4)(:=, 1, - , t_1)

(5)(j<, c, d, (8))

(6)(:=, 0, - , t_2)

(7)(jump, - , - ,(9))

(8)(:=, 1, - , t_2)

(9)(j>, e, f, (12))

(10)(:=, 0, - , t_3)

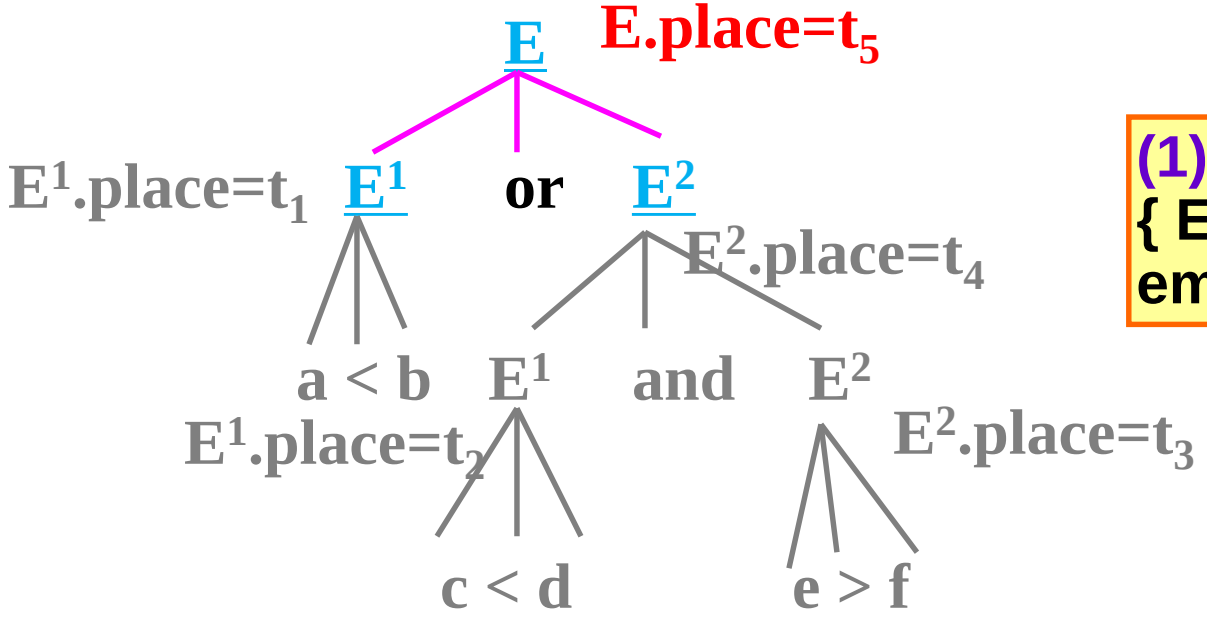
(11)(jump,- ,- ,(13))

(12)(:=, 1, - , t_3)

(13)(and, t_2, t_3, t_4)

2. 直接计算的翻译

- 例：布尔表达式a<b or c<d and e>f的翻译



(1)E→E1 or E2
{ E.place : = newtemp ;
emit (or, E1.place , E2.place , E.place)}

(1)(j<, a, b, (4))

(2)(:=, 0, - , t_1)

(3)(jump,- ,- ,(5))

(4)(:=, 1, - , t_1)

(5)(j<, c, d, (8))

(6)(:=, 0, - , t_2)

(7)(jump, - , - ,(9))

(8)(:=, 1, - , t_2)

(9)(j>, e, f, (12))

(10)(:=, 0, - , t_3)

(11)(jump,- ,- ,(13))

(12)(:=, 1, - , t_3)

(13)(and, t_2, t_3, t_4)

(14)(or, t_1, t_4, t_5)

2. 直接计算的翻译

- 例：布尔表达式a<b or c<d and e>f的翻译

(1)(j<, a, b, (4))
(2)(:=, 0, - , t_1)
(3)(jump,- ,- ,(5))
(4)(:=, 1, - , t_1)
(5)(j<, c, d, (8))
(6)(:=, 0, - , t_2)
(7)(jump, - , - ,(9))
(8)(:=, 1, - , t_2)
(9)(j>, e, f, (12))
(10)(:=, 0, - , t_3)
(11)(jump,- ,- ,(13))
(12)(:=, 1, - , t_3)
(13)(and, t_2, t_3, t_4)
(14)(or, t_1, t_4, t_5)

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

四元式编号从100开始

3. 短路计算

- 条件控制语句

  S → **if (** B **)** S1

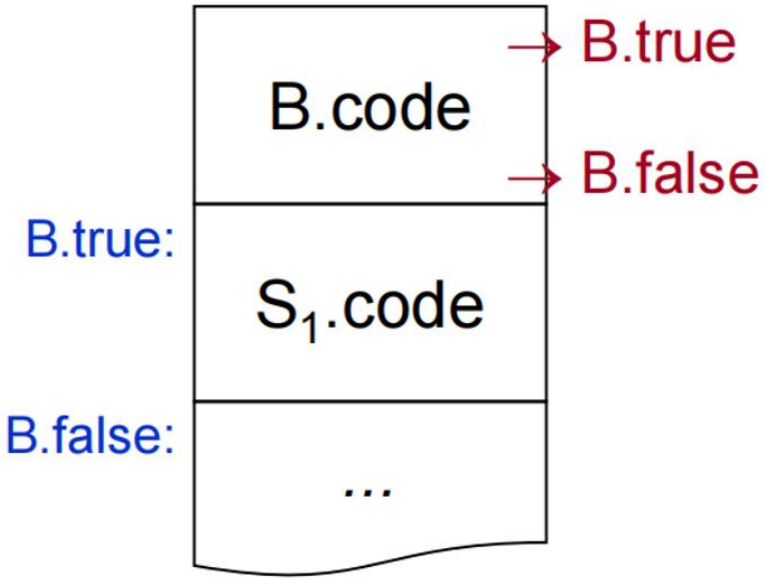
  S → **if (** B **)** S1 **else** S2

  S → **while (** B **)** S1

- 短路计算
  - 布尔表达式计算到某一部分就可以得到结果，而无需对布尔表达式进行完全计算
    - ✓ A or B　　if A then 1 else B
    - ✓ A and B　　if A then B else 0
    - ✓ not A　　if A then 0 else 1

3. 短路计算

- 为布尔表达式B引入两个新的属性：
  - B.true：表达式的真出口，它指向表达式为真时的转向
  - B.false：表达式的假出口，它指向表达式为假时的转向

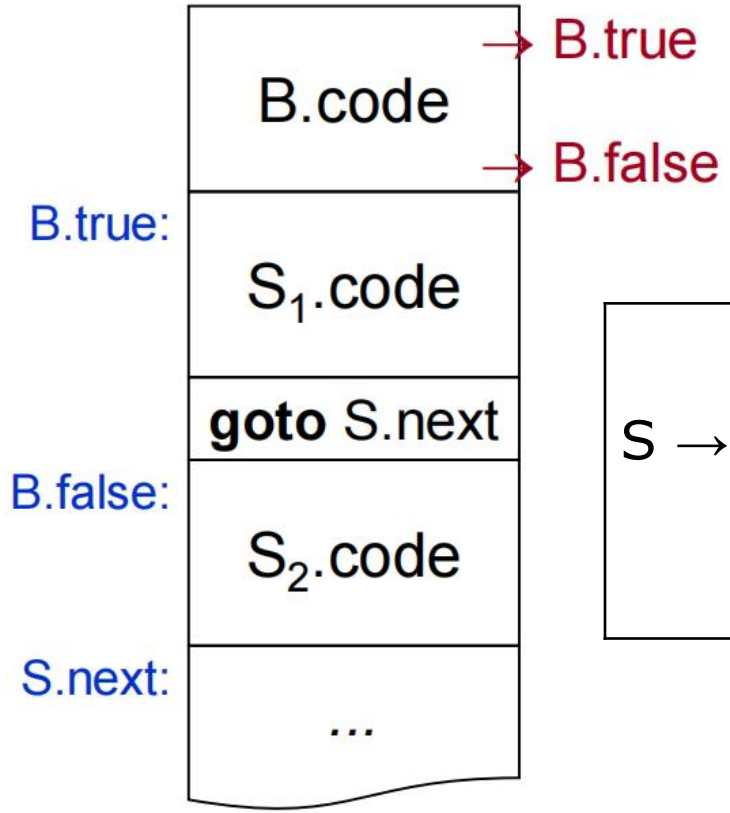


S → if (B) S1

S → **if** (B) S1	B.true = **new** Label(); B.false = S1.next = S.next; S.code = B.code || label(B.true) || S1.code

3. 短路计算

- 为布尔表达式B引入两个新的属性：
  - B.true：表达式的真出口，它指向表达式为真时的转向
  - B.false：表达式的假出口，它指向表达式为假时的转向



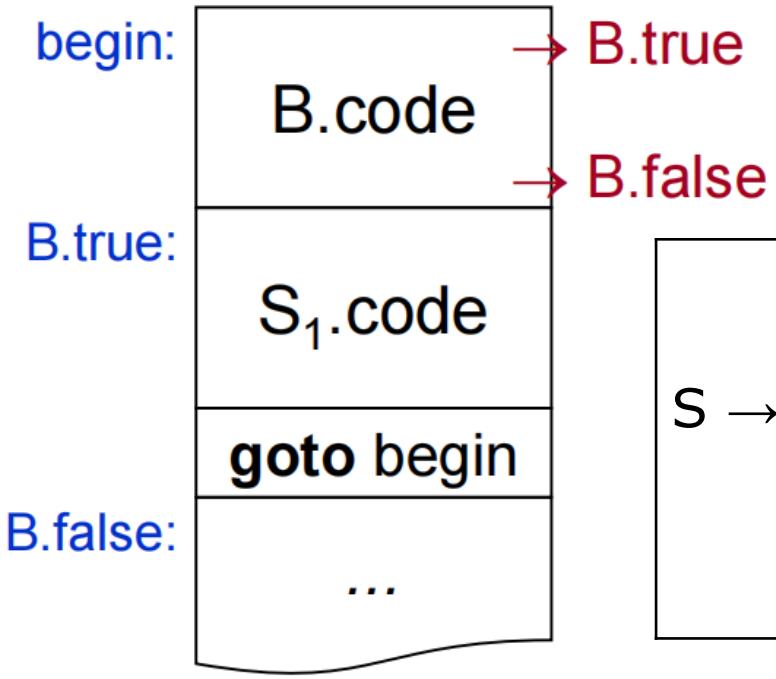
S → **if** (B) S1 **else** S2

B.true = **new** Label();
B.false = **new** Label();
S1.next = S2.next = S.next;
S.code = B.code || label(B.true) || S1.code ||
gen('**goto**' S.next) || label(B.false) || S2.code

S → if (B) S1 else S2

3. 短路计算

- 为布尔表达式B引入两个新的属性：
  - B.true：表达式的真出口，它指向表达式为真时的转向
  - B.false：表达式的假出口，它指向表达式为假时的转向



S → while (B) S1

S → **while** (B) S1	begin = **new** Label(); B.true = **new** Label(); B.false = S.next; S1.next = begin; S.code = label(begin) || B.code || label(B.true) || S1.code || gen('**goto**' begin)

3. 短路计算

Productions	Semantic Rules
P → S	S.next = **new** Label(); P.code = S.code || label(S.next)
S → **assign**	S.code = **assign**.code
S → S_1 S_2	S_1 .next = **new** Label(); S_2 .next = S.next; S.code = S_1 .code || label(S_1 .next) || S_2 .code
S → **if** (B) S_1	B.true = **new** Label(); B.false = S_1 .next = S.next; S.code = B.code || label(B.true) || S_1 .code
S → **if** (B) S_1 **else** S_2	B.true = **new** Label(); B.false = **new** Label(); S_1 .next = S_2 .next = S.next; S.code = B.code || label(B.true) || S_1 .code || gen('**goto**' S.next) || label(B.false) || S_2 .code
S → **while** (B) S_1	begin = **new** Label(); B.true = **new** Label(); B.false = S.next; S_1 .next = begin; S.code = label(begin) || B.code || label(B.true) || S1 .code || gen('**goto**' begin)

控制流语句的语法制导定义

3. 短路计算

为布尔表达式生成三地址码

Productions	Semantic Rules
B → B1 **||** B2	B_1 .true = B.true; B_1 .false = **new** Label(); B_2 .true = B.true; B_2 .false = B.false; B.code = B_1 .code || label(B_1 .false) || B_2 .code
B → B1 **&&** B2	B_1 .true = **new** Label(); B_1 .false = B.false; B_2 .true = B.true; B_2 .false = B.false; B.code = B_1 .code || label(B_1 .true) || B_2 .code
B → **!** B1	B_1 .true = B.false; B_1 .false = B.true; B.code = B_1 .code
B → E1 **relop** E2	B.code = E_1 .code || E_2 .code || gen('**if**' E_1 .addr **relop**.op E_2 .addr '**goto**' B.true) || gen('**goto**' B.false)
B → **true**	B.code = gen('**goto**' B.true)
B → **false**	B.code = gen('**goto**' B.false)

3. 短路计算

- 例：条件控制语句**if (x < 100 || x > 200 && x != y) x = 0**的翻译

```
        if x < 100 goto L2
        goto L3
L3 :    if x > 200 goto L4
        goto L1
L4 :    if x != y goto L2
        goto L1
L2 :    x = 0
L1 :    …
```

3. 短路计算

- 例：条件控制语句 a<b or c<d and e>f 的翻译

(1) (j<, a, b, E.true)
(2) (jump, - , - , (3))
(3) (j<, c , d , (5))
(4) (jump, - , - , E.false)
(5) (j>, e , f , E.true)
(6) (jump, - , - , E.false)

CONTENTS

目 录

1. 回填技术的提出

- 遇到的问题
  - 在把布尔式翻译成一串条件转和无条件转四元式时，真假出口**未能**在生成四元式时确定
  - 多个四元式可能有**相同的出口**
- 解决办法：
  - 真假出口的**拉链与回填[backpatching]**

1. 回填技术的提出

- 例：条件控制语句 **a<b or c<d and e>f** 的翻译

(1) (j<, a, b, E.true)

(2) (jump, - , - , (3))

(3) (j<, c , d , (5))

(4) (jump, - , - , E.false)

(5) (j>, e , f , E.true)

(6) (jump, - , - , E.false)

– E.true和E.false不能在产生四元式时确定，要等**将来目标明确时再回填**，为此要记录这些要回填的四元式

– 通常采用“**拉链**”的办法，把需要回填**E.true**的四元式拉成一条**“真”链**，把需要回填**E.false**的四元式拉成一条**“假”链**

2. 拉链方式

若有四元式序列：	则链接成为：
(10) (*, *, *, E.true)	(10) (*, *, *, **0**)
……	……
(20) (*, *, *, E.true)	(20) (*, *, *, **10**)
……	……
(30) (*, *, *, E.true)	(30) (*, *, *, **20**)

– 把地址（30）作为链首，地址（10）作为链尾，0为链尾标志。

– 四元式的第四个区段存放链指针。

– **E.true 和E.false用于存放“真”链和“假”链的链首。**

3. 回填的翻译方案

- 语义：
  - 函数**merge (p1, p2)** 用于把p1和p2为链首的两条链合并成1条，返回合并后的链首值。
    - ✓ 当p2为空链时，返回p1；
    - ✓ 当p2不为空链时，把p2的链尾第四区段改为p1，返回p2。
  - 函数**backpatch (p, t)** 用于把链首p所链接的每个四元式的第四区段都填为转移目标t【**在知道t具体在哪里之后**】

3. 回填的翻译方案

- 对布尔表达式

$B \rightarrow B_1$ **||** $M\ B_2$ { backpatch(B_1.falseList, M.instruction);
B.trueList = merge(B_1.trueList, B_2.trueList);
B.falseList = B_2.falseList; }

$B \rightarrow B_1$ **&&** $M\ B_2$ { backpatch(B_1.trueList, M.instruction);
B.trueList = B_2.trueList;
B.falseList = merge(B_1.falseList, B_2.falseList); }

$B \rightarrow$ **!** B_1 { B.trueList = B_1.falseList;
B.falseList = B_1.trueList; }

$B \rightarrow$ **(** B_1 **)** { B.trueList = B_1.trueList;
B.falseList = B_1.falseList; }

$B \rightarrow E_1$ **relop** E_2 { B.trueList = **new** List(nextInstruction);
B.falseList = **new** List(nextInstruction + 1);
emit('**if**' E_1.addr relop.op E_2.addr '**goto** __');
emit('**goto** __'); }

$B \rightarrow$ **true** { B.trueList = **new** List(nextInstruction);
emit('**goto** __'); }

$B \rightarrow$ **false** { B.falseList = **new** List(nextInstruction);
emit('**goto** __'); }

$M \rightarrow \varepsilon$ { M.instruction = nextInstruction; }

3. 回填的翻译方案

- 对布尔表达式

1) $E \rightarrow E^1$ or E^2

{ E.codebegin: = E^1.codebegin ;
backpatch (E^1.false , E^2.codebegin) ;
E.true: = merge (E^1.true , E^2.true) ;
E.false: = E2.false }

2) $E \rightarrow E^1$ and E^2

{ E.codebegin:＝E^1.codebigin ;
backpatch (E^1.true , E^2.codebegin) ;
E.true:＝E^2.true ;
E.false:＝merge (E^1.fasle , E^2.false) }

3) $E \rightarrow$ not E^1

{ E.codebegin:＝E^1.codebegin ;
E.true:＝E^1.false ;
E.false:＝E^1.true }

4) $E \rightarrow (E^1)$

{ E.codebegin:＝E^1.codebegin ;
E.true:＝E^1.true ;
E.false:＝E^1.false }

3. 回填的翻译方案

- 对布尔表达式

5) $E \rightarrow id_1\ rop\ id_2$

{ E.codebegin:＝nextstat ;
E.true:＝nextstat ;
E.false:＝nextstat+1;
emit (jrop , id_1.place , id_2.place , 0) ;
emit (jump ,－,－, 0) }

6) E→true

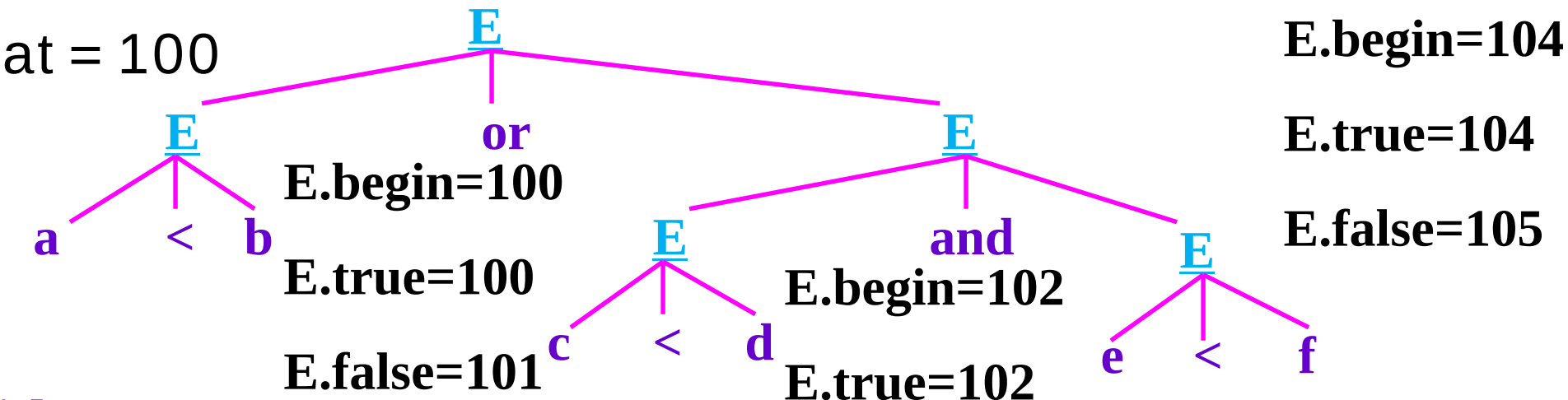
{ E.codebegin:＝nextstat ;
E.true:＝nextstat ;
E.false:＝0;
emit (jump ,－,－,0) }

7) E→false

{ E.codebegin:＝nextstat ;
E.false:＝nextstat ;
E.true:＝0;
emit (jump ,－,－,0) }

3. 回填的翻译方案

- 例：a<b or c<d and e<f 的翻译过程，假定四元式编号从100开始，即开始时nextstat = 100



5) $E \rightarrow id_1\ rop\ id_2$

{ E.codebegin:＝nextstat ;

E.true:＝nextstat ;

E.false:＝nextstat+1;

emit (jrop , id_1.place , id_2.place , 0) ;

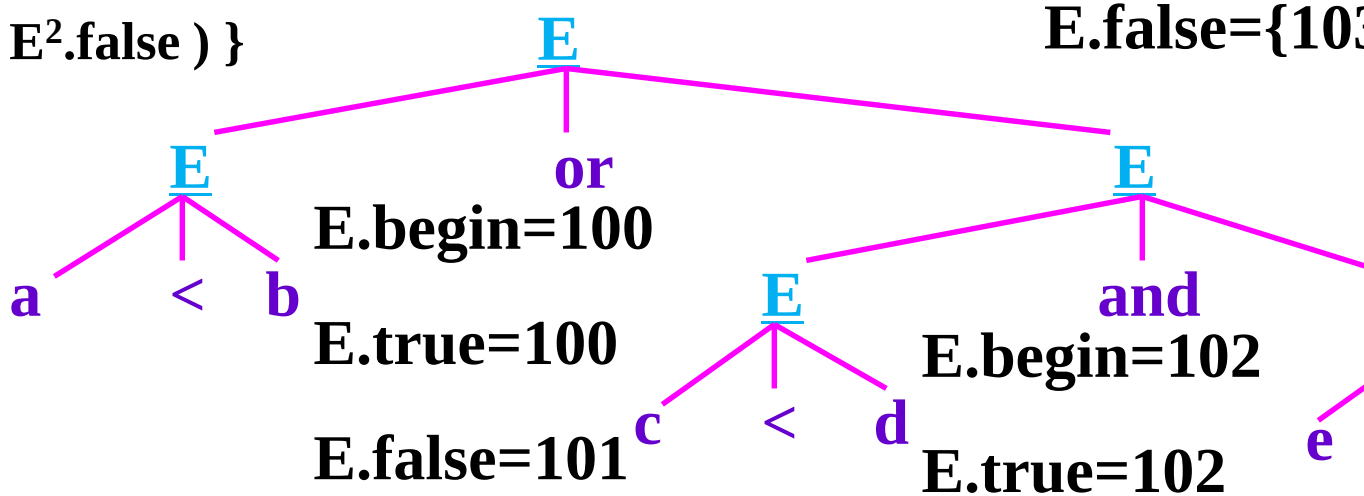
emit (jump ,－,－, 0) }

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

3. 回填的翻译方案

2) $E \rightarrow E^1$ and E^2

{ E.codebegin:＝E^1.codebigin ;
backpatch (E^1.true , E^2.codebegin) ;
E.true:＝E^2.true ;
E.false:＝merge (E^1.fasle , E^2.false) }



E.begin=102

E.true=104

E.false={103,105}=105

E.begin=100

E.true=100

E.false=101

E.begin=102

E.true=102

E.false=103

E.begin=104

E.true=104

E.false=105

100 : (j< , a , b , 0)
101: (jump ,—,—,0)
102: (j< , c , d , 0) ⟶ 102 (j< , c , d ,104)
103: (jump ,—,—, 0)
104: (j< , e , f , 0)
105: (jump ,—,—,0) ⟶ 105 (jump ,—,—,103)

3. 回填的翻译方案

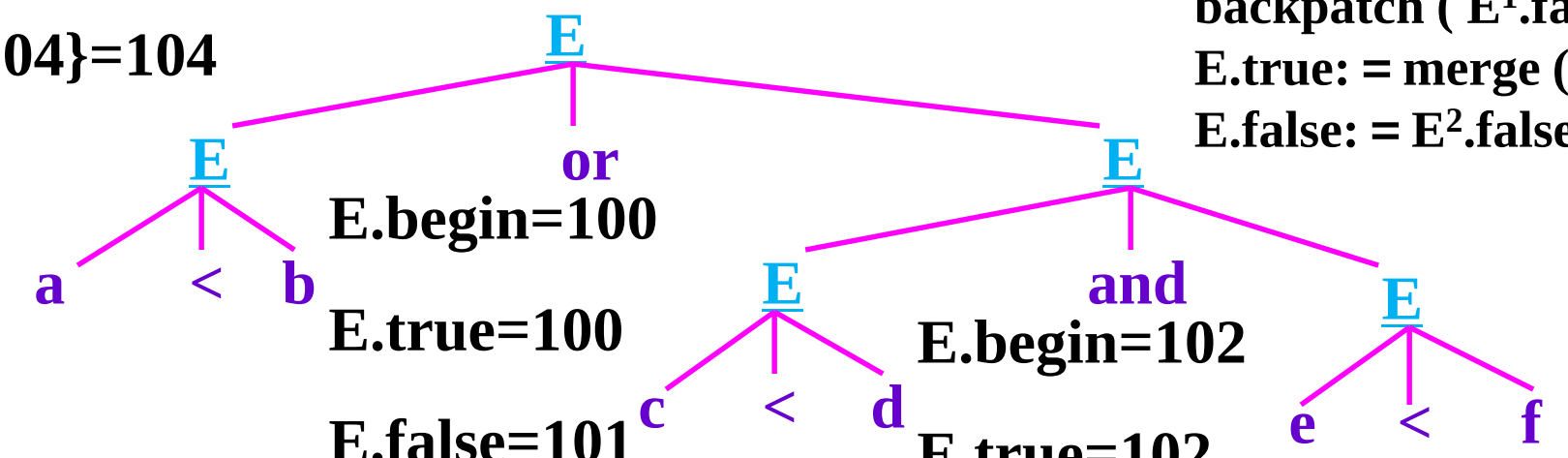
1) $E \rightarrow E^1$ or E^2

{ E.codebegin: = E^1.codebegin ;
backpatch (E^1.false , E^2.codebegin) ;
E.true: = merge (E^1.true , E^2.true) ;
E.false: = E^2.false }

E.begin=100

E.true={100,104}=104

E.false=105



100 : (j< , a , b , 0)

101: (jump ,—,—,0) ⟶ 101(jump,—,—,102)

102: (j< , c , d , 0) ⟶ 102 (j< , c , d ,104)

103: (jump ,—,—, 0)

104: (j< , e , f , 0) ⟶ 104 (j< , e , f ,100)

105: (jump ,—,—,0) ⟶ 105 (jump ,—,—,103)

3. 回填的翻译方案

- 例：a<b or c<d and e<f 的翻译过程，假定四元式编号从100开始，即开始时nextstat = 100
- 最终结果：

100：(j< , a , b , 0)

101：(jump,—,—,102)

102：(j< , c , d , 104)

103：(jump, - , - , 0)

104：(j< , e , f , 100)

105：(jump, - , - ,103)

“真”链首E.true = 104，“假”链首E.false = 105。

3. 回填的翻译方案

- 对条件控制语句

$S \rightarrow$ **if (** B **)** M S_1 { backpatch(B.trueList, M.instruction);
S.nextList = merge(B.falseList, S_1.nextList); }

$S \rightarrow$ **if (** B **)** M_1 S_1 N **else** M_2 S_2
{ backpatch(B.trueList, M_1.instruction);
backpatch(B.falseList, M_2.instruction);
S.nextList = merge(S_1.nextList, N.nextList, S_2.nextList); }

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1 { backpatch(B.trueList, M_2.instruction);
backpatch(S_1.nextList, M_1.instruction);
S.nextList = B.falseList;
emit('**goto**' M_1.instruction); }

$S \rightarrow$ **{** L **}** { S.nextList = L.nextList; }

$S \rightarrow$ A **;** { S.nextList = **new** List(); // Assignment or Atom }

$M \rightarrow \varepsilon$ { M.instruction = nextInstruction; }

$N \rightarrow \varepsilon$ { N.nextList = **new** List(nextInstruction);
emit('**goto** __'); }

$L \rightarrow L_1$ M S { backpatch(L_1.nextList, M.instruction);
L.nextList = S.nextList; }

$L \rightarrow$ S { L.nextList = S.nextList; }