



中山大學
SUN YAT-SEN UNIVERSITY



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

01

中间代码概述
Introduction

02

类型和声明
Types and
Declarations

03

表达式和语句
Assignment and
Expressions

04

类型检查
Type
Checking

05

布尔表达式
Boolean
Expressions

06

回填技术
Backpatching

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

- $E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{not } E \mid (E) \mid \text{id rop id} \mid \text{true} \mid \text{false}$
- 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

- ✓ $1 \text{ or } \underline{0 \text{ and } 1} = 1 \text{ or } \underline{0} = 1$

- 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：

- $A \text{ or } B$ if A then 1 else B

- $A \text{ and } B$ 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) $E \rightarrow E^1 \text{ or } E^2$

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

(2) $E \rightarrow E^1 \text{ and } E^2$

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

(3) $E \rightarrow \text{not } E^1$

```
{ E.place := newtemp ;
  emit ( not , E1.place , — , E.place ) }
```

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

```
{ E.place := E1.place }
```

(5) $E \rightarrow id_1 \text{ rop } id_2$

```
{ E.place := newtemp ;
```

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

```
  emit (jrop , id1.place , id2.place , nextstat+3 ) ;
```

(2) (:=, 0, -, t₁)

```
  emit ( := , 0 , - , E.place ) ;
```

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

```
  emit ( jump , — , — , nextstat+2 ) ;
```

(4) (:=, 1, -, t₁)

```
  emit ( := , 1 , - , E.place ) }
```

(5) ...

(6) $E \rightarrow \text{true}$

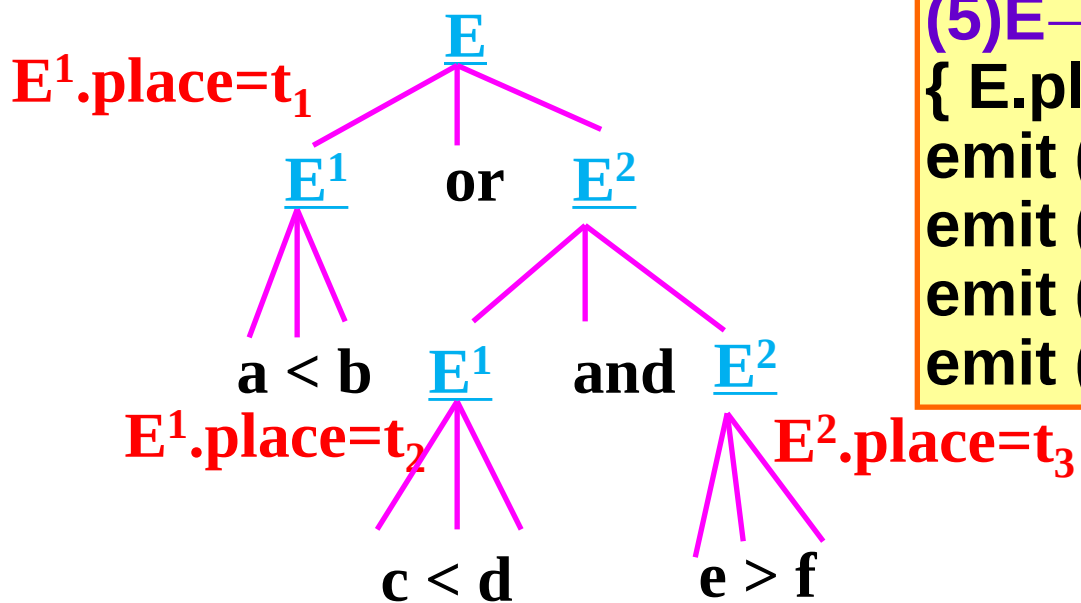
```
{ E.place := newtemp; emit(:=,1,- ,E.place) }
```

(7) $E \rightarrow \text{false}$

```
{E.place:=newtemp;emit(:=,0,- ,E.place)}
```

2. 直接计算的翻译

- 例：布尔表达式 $a < b \text{ or } c < d \text{ and } e > f$ 的翻译



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

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

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

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

(2)(:=, 0, -, t_1)(6)(:=, 0, -, t_2)(10)(:=, 0, -, t_3)

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

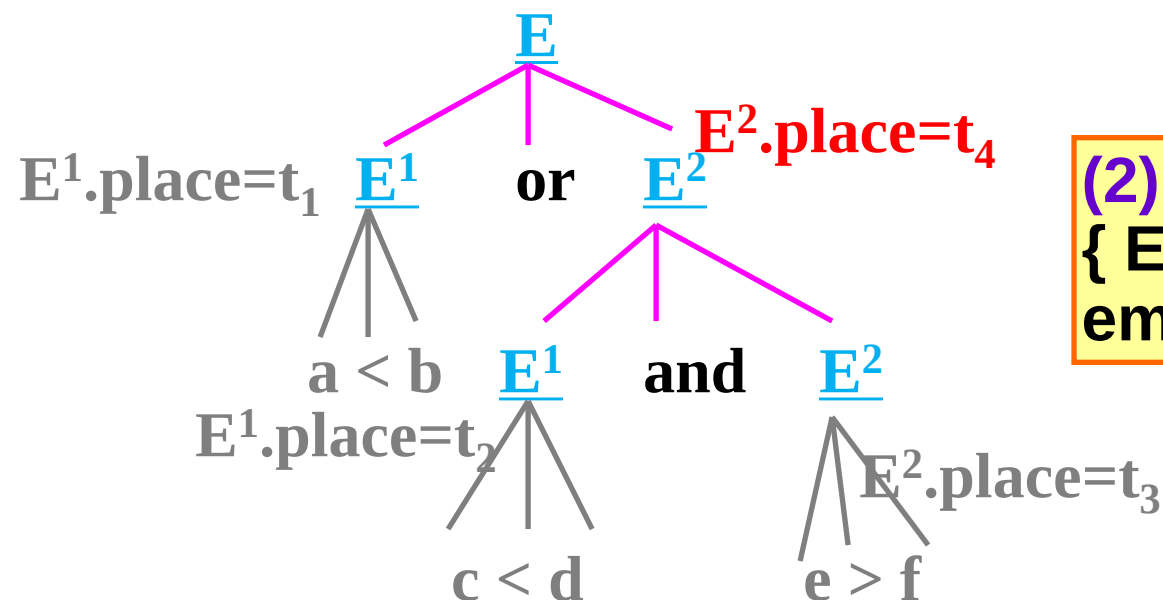
(7)(jump, -, -, (9))

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

(4)(:=, 1, -, t_1)(8)(:=, 1, -, t_2)(12)(:=, 1, -, t_3)

2. 直接计算的翻译

- 例：布尔表达式 $a < b \text{ or } c < d \text{ and } e > f$ 的翻译



(2) $E \rightarrow E1 \text{ and } E2$

{ $E.\text{place} := \text{newtemp}$;
 $\text{emit}(\text{and}, E1.\text{place}, E2.\text{place}, E.\text{place})$ }

(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)

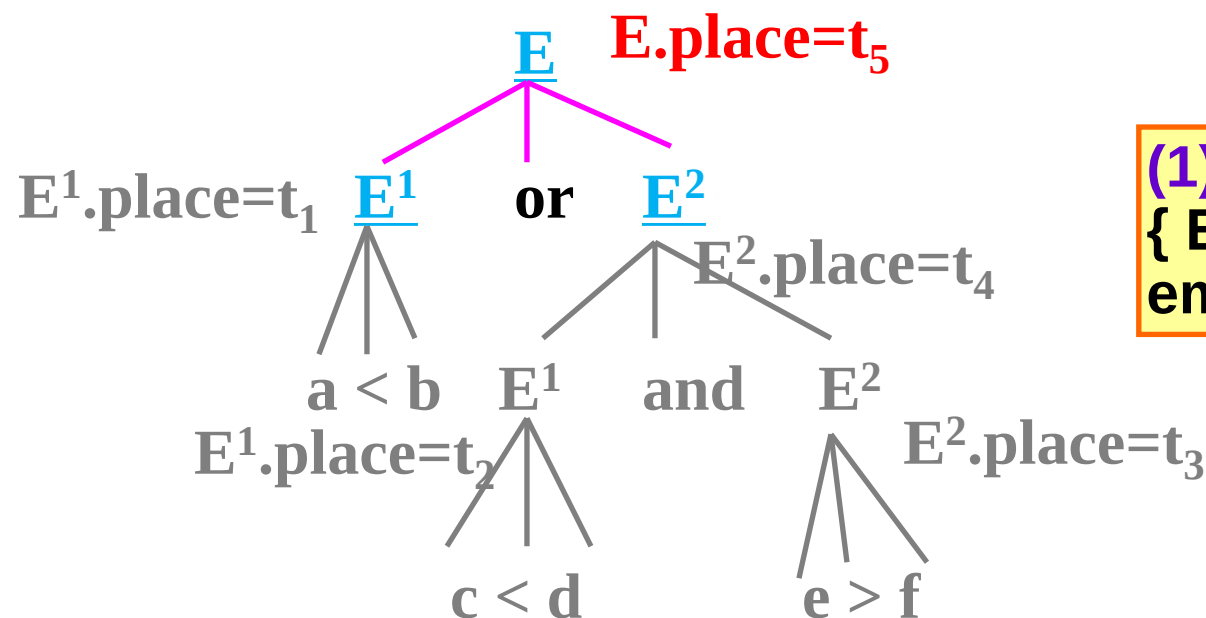
(11)(jump, -, -, (13))

(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 $a < b \text{ or } c < d \text{ and } e > f$ 的翻译



(1) $E \rightarrow E1 \text{ or } E2$
 { $E.\text{place} := \text{newtemp}$;
 emit (or, $E1.\text{place}$, $E2.\text{place}$, $E.\text{place}$) }

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

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

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

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

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

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

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

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

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

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

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

(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 $a < b \text{ or } c < d \text{ and } e > f$ 的翻译

(1)(j<, a, b, (4))	(9)(j>, e, f, (12))		100: if a < b goto 103	108: if e < f goto 111
(2)(:=, 0, -, t ₁)	(10)(:=, 0, -, t ₃)		101: t ₁ = 0	109: t ₃ = 0
(3)(jump, -, -, (5))	(11)(jump, -, -, (13))		102: goto 104	110: goto 112
(4)(:=, 1, -, t ₁)	(12)(:=, 1, -, t ₃)	⇒	103: t ₁ = 1	111: t ₃ = 1
(5)(j<, c, d, (8))	(13)(and, t ₂ , t ₃ , t ₄)		104: if c < d goto 107	112: t ₄ = t ₂ and t ₃
(6)(:=, 0, -, t ₂)	(14)(or, t ₁ , t ₄ , t ₅)		105: t ₂ = 0	113: t ₅ = t ₁ or t ₄
(7)(jump, -, -, (9))			106: goto 108	
(8)(:=, 1, -, t ₂)			107: t ₂ = 1	

四元式编号从100开始

3. 短路计算

- 条件控制语句

$S \rightarrow \text{if} (B) S1$

$S \rightarrow \text{if} (B) S1 \text{ else } S2$

$S \rightarrow \text{while} (B) S1$

- 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

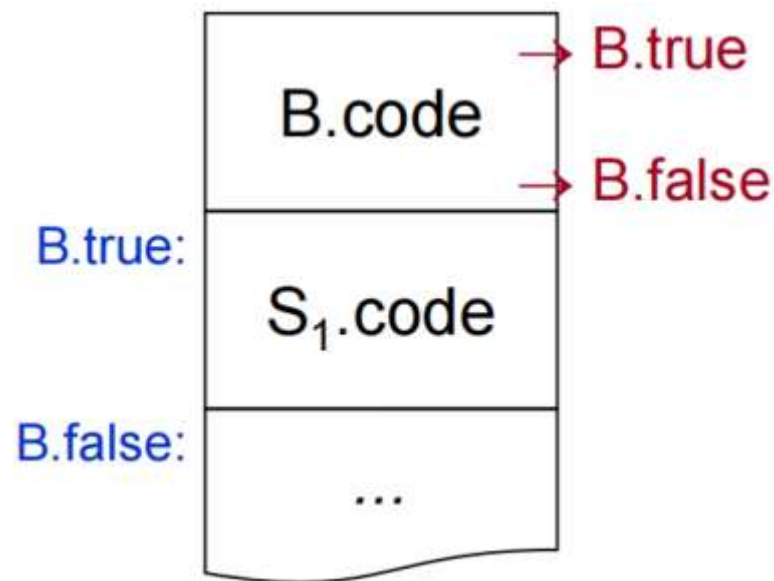
✓ A or B if A then 1 else B

✓ 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

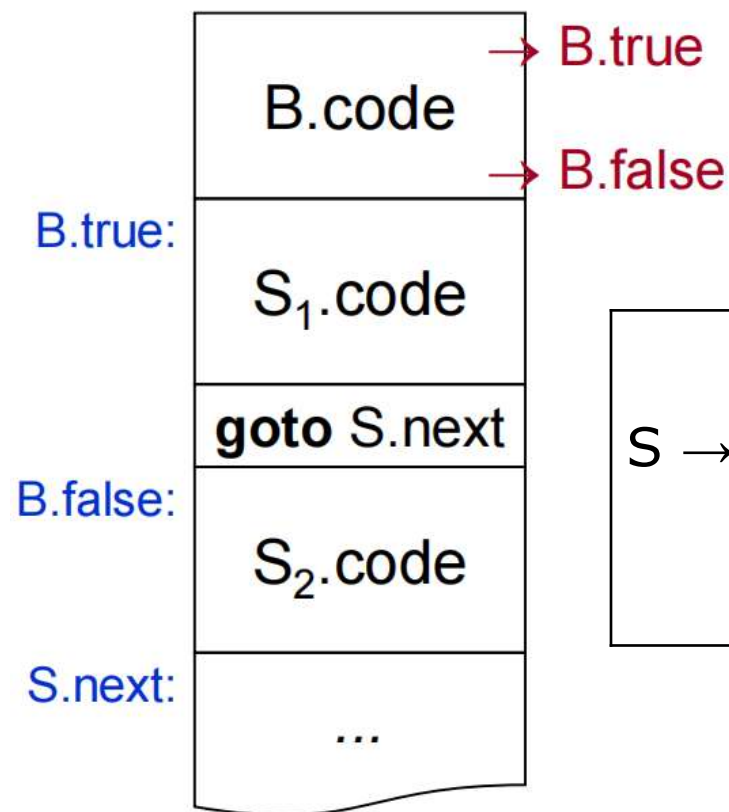


S → if (B) S1

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

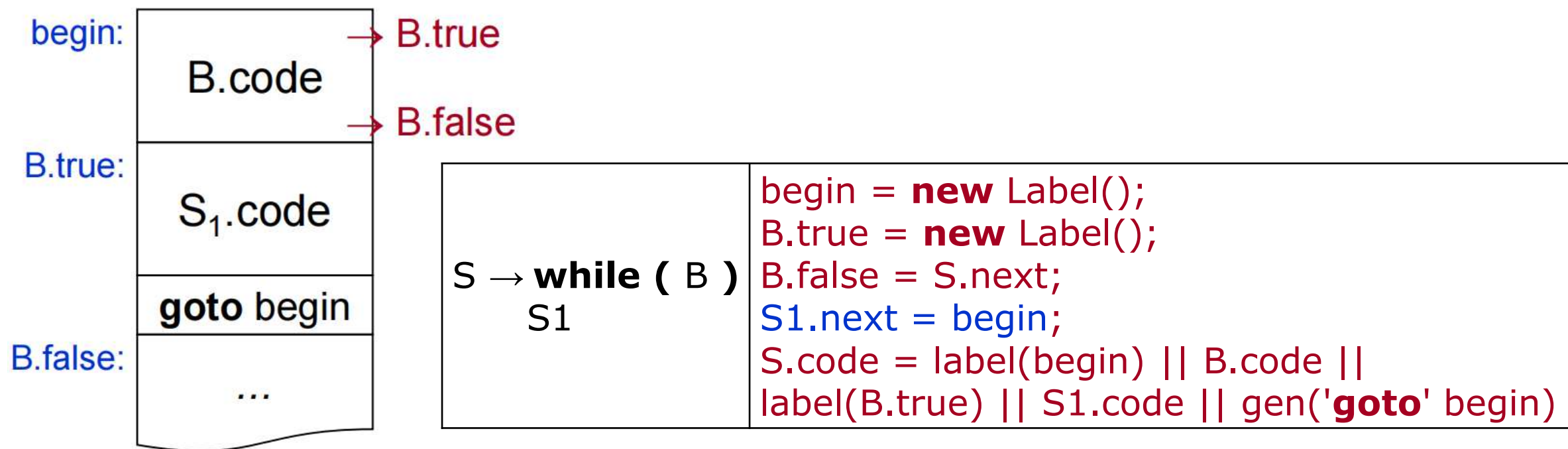


$S \rightarrow \text{if (B) } S1$ $\quad \text{else } S2$	$B.\text{true} = \text{new Label}();$ $B.\text{false} = \text{new Label}();$ $S1.\text{next} = S2.\text{next} = S.\text{next};$ $S.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel S1.\text{code} \parallel$ $\text{gen('goto' } S.\text{next}) \parallel \text{label}(B.\text{false}) \parallel S2.\text{code}$
--	--

$S \rightarrow \text{if (B) } S1 \text{ else } S2$

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



$S \rightarrow \text{while} (B) S1$

3. 短路计算

Productions	Semantic Rules
$P \rightarrow S$	$S.next = \text{new Label}();$ $P.code = S.code \parallel \text{label}(S.next)$
$S \rightarrow \text{assign}$	$S.code = \text{assign}.code$
$S \rightarrow S_1$ S_2	$S_1.next = \text{new Label}();$ $S_2.next = S.next;$ $S.code = S_1.code \parallel \text{label}(S_1.next) \parallel S_2.code$
$S \rightarrow \text{if (B) } S_1$	$B.true = \text{new Label}();$ $B.false = S_1.next = S.next;$ $S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code$
$S \rightarrow \text{if (B) } S_1$ $\text{else } S_2$	$B.true = \text{new Label}();$ $B.false = \text{new Label}();$ $S_1.next = S_2.next = S.next;$ $S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code \parallel$ $\text{gen('goto' } S.next) \parallel \text{label}(B.false) \parallel S_2.code$
$S \rightarrow \text{while (B)}$ S_1	$begin = \text{new Label}();$ $B.true = \text{new Label}();$ $B.false = S.next;$ $S_1.next = begin;$ $S.code = \text{label}(begin) \parallel B.code \parallel \text{label}(B.true) \parallel S_1.code \parallel \text{gen('goto' } begin)$

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
$B \rightarrow B_1 \parallel B_2$	$B_1.\text{true} = B.\text{true};$ $B_1.\text{false} = \text{new Label}();$ $B_2.\text{true} = B.\text{true};$ $B_2.\text{false} = B.\text{false};$ $B.\text{code} = B_1.\text{code} \parallel \text{label}(B_1.\text{false}) \parallel B_2.\text{code}$
$B \rightarrow B_1 \&\& B_2$	$B_1.\text{true} = \text{new Label}();$ $B_1.\text{false} = B.\text{false};$ $B_2.\text{true} = B.\text{true};$ $B_2.\text{false} = B.\text{false};$ $B.\text{code} = B_1.\text{code} \parallel \text{label}(B_1.\text{true}) \parallel B_2.\text{code}$
$B \rightarrow ! B_1$	$B_1.\text{true} = B.\text{false};$ $B_1.\text{false} = B.\text{true};$ $B.\text{code} = B_1.\text{code}$
$B \rightarrow E_1 \text{ relop } E_2$	$B.\text{code} = E_1.\text{code} \parallel E_2.\text{code}$ $\parallel \text{gen}(\text{'if' } E_1.\text{addr } \text{relop.op } E_2.\text{addr 'goto' } B.\text{true})$ $\parallel \text{gen}(\text{'goto' } B.\text{false})$
$B \rightarrow \text{true}$	$B.\text{code} = \text{gen}(\text{'goto' } B.\text{true})$
$B \rightarrow \text{false}$	$B.\text{code} = \text{gen}(\text{'goto' } B.\text{false})$

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

01

中间代码概述
Introduction

02

类型和声明
Types and
Declarations

03

表达式和语句
Assignment and
Expressions

04

类型检查
Type
Checking

05

布尔表达式
Boolean
Expressions

06

回填技术
Backpatching

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：

(10) (*, *, *, E.true)

.....

(20) (*, *, *, E.true)

.....

(30) (*, *, *, E.true)

则链接成为：

(10) (*, *, *, **0**)

.....

(20) (*, *, *, **10**)

.....

(30) (*, *, *, **20**) **E.true=30**

- 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

$B \rightarrow B_1 \parallel M B_2$ { `backpatch(B1.falseList, M.instruction);`
`B.trueList = merge(B1.trueList, B2.trueList);`
`B.falseList = B2.falseList; }`

$B \rightarrow B_1 \&\& M B_2$ { `backpatch(B1.trueList, M.instruction);`
`B.trueList = B2.trueList;`
`B.falseList = merge(B1.falseList, B2.falseList); }`

$B \rightarrow ! B_1$ { `B.trueList = B1.falseList;`
`B.falseList = B1.trueList; }`

$B \rightarrow (B_1)$ { `B.trueList = B1.trueList;`
`B.falseList = B1.falseList; }`

$B \rightarrow E_1 \text{ relop } E_2$ { `B.trueList = new List(nextInstruction);`
`B.falseList = new List(nextInstruction + 1);`
`emit('if' E1.addr relop.op E2.addr 'goto __');`
`emit('goto __');` }

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

$B \rightarrow \text{false}$ { `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) $E \rightarrow E^1 \text{ or } E^2$

```
{ E.codebegin := E1.codebegin ;  
  backpatch ( E1.false , E2.codebegin ) ;  
  E.true := merge ( E1.true , E2.true ) ;  
  E.false := E2.false }
```

3) $E \rightarrow \text{not } E^1$

```
{ E.codebegin := E1.codebegin ;  
  E.true := E1.false ;  
  E.false := E1.true }
```

2) $E \rightarrow E^1 \text{ and } E^2$

```
{ E.codebegin := E1.codebegin ;  
  backpatch ( E1.true , E2.codebegin ) ;  
  E.true := E2.true ;  
  E.false := merge ( E1.false , E2.false ) }
```

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

```
{ E.codebegin := E1.codebegin ;  
  E.true := E1.true ;  
  E.false := E1.false }
```

3. 回填的翻译方案

- 对布尔表达式

5) $E \rightarrow id_1 \text{ rop } id_2$

```
{ E.codebegin:=nextstat ;  
  E.true:=nextstat ;  
  E.false:=nextstat+1;  
  emit ( jrop , id1.place , id2.place , 0 ) ;  
  emit ( jump , — , — , 0 ) }
```

6) $E \rightarrow \text{true}$

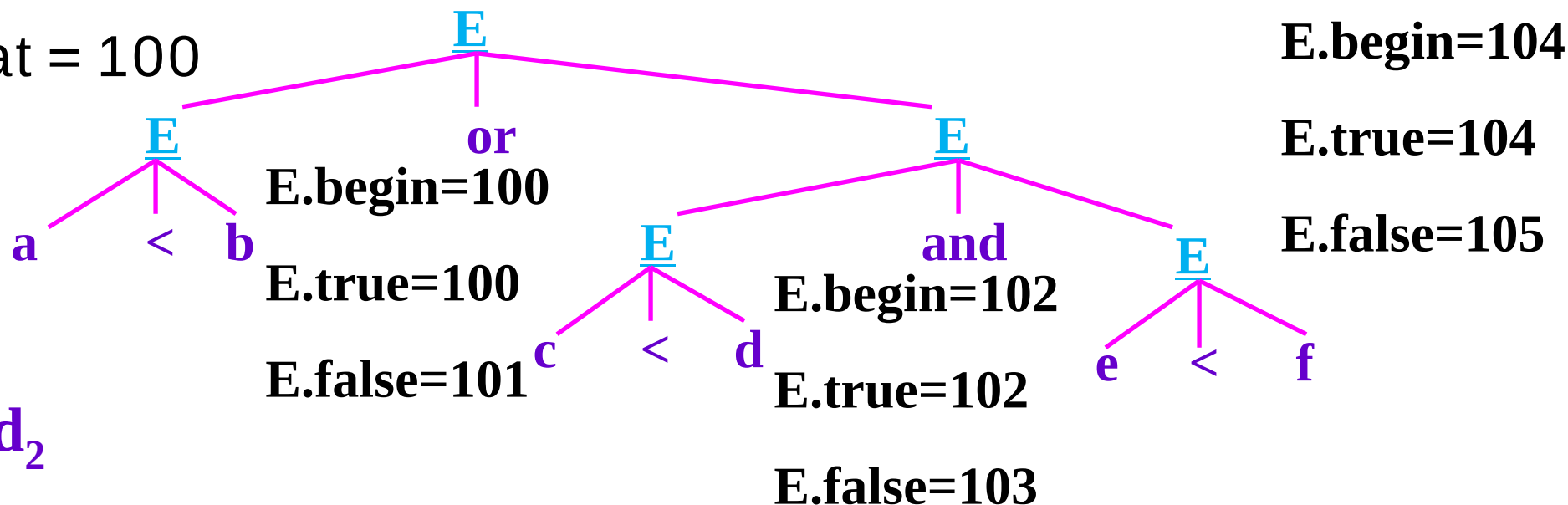
```
{ E.codebegin:=nextstat ;  
  E.true:=nextstat ;  
  E.false:=0;  
  emit ( jump , — , — , 0 ) }
```

7) $E \rightarrow \text{false}$

```
{ 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) $E \rightarrow id_1 \text{ rop } id_2$

```

{ E.codebegin:=nextstat ;
  E.true:=nextstat ;
  E.false:=nextstat+1;
  emit ( jrop , id1.place , id2.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) $E \rightarrow E^1 \text{ and } E^2$

```

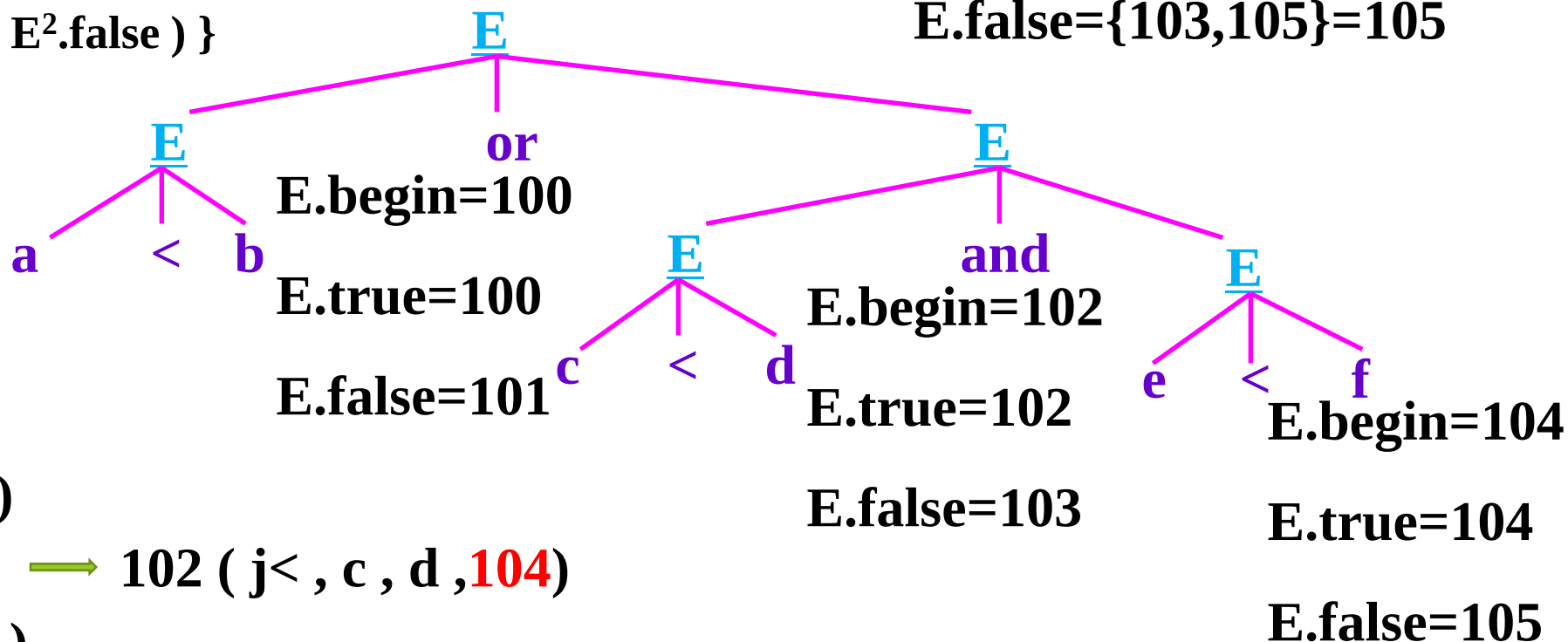
{ E.codebegin := E1.codebegin ;
  backpatch ( E1.true , E2.codebegin ) ;
  E.true := E2.true ;
  E.false := merge ( E1.false , E2.false ) }

```

E.begin=102

E.true=104

E.false={103,105}=105



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

101: (jump , — , — , 0)

102: (j< , c , d , 0) $\longrightarrow$ 102 (j< , c , d , 104)

103: (jump , — , — , 0)

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

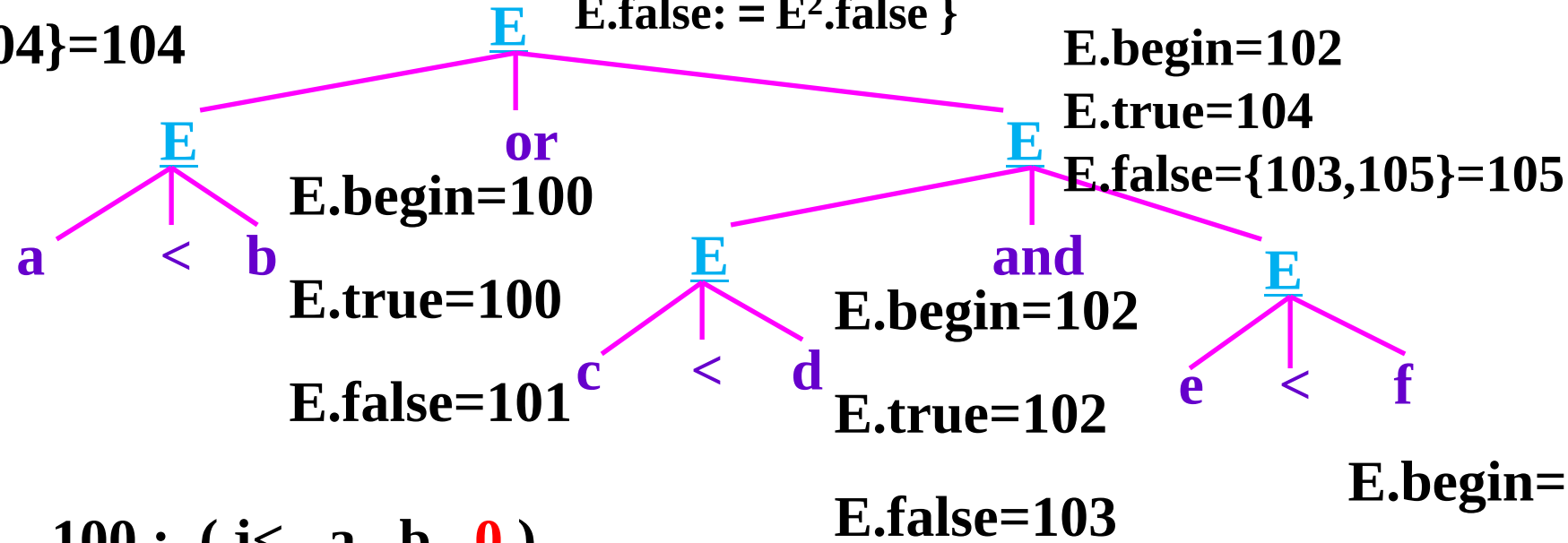
105: (jump , — , — , 0) $\longrightarrow$ 105 (jump , — , — , 103)

3. 回填的翻译方案

E.begin=100

E.true={100,104}=104

E.false=105



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

E.begin=104
E.true=104
E.false=105

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

$S \rightarrow \text{if } (B) M S_1$	{ backpatch(B.trueList, M.instruction); S.nextList = merge(B.falseList, S ₁ .nextList); }
$S \rightarrow \text{if } (B) M_1 S_1 N \text{ else } M_2 S_2$	{ backpatch(B.trueList, M ₁ .instruction); backpatch(B.falseList, M ₂ .instruction); S.nextList = merge(S ₁ .nextList, N.nextList, S ₂ .nextList); }
$S \rightarrow \text{while } M_1 (B) M_2 S_1$	{ 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; }