编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第七章 语义分析的中间代码生成

第七章 语义分析的中间代码生成

- 本章目录
- 7.1 中间语言
- 7.2 赋值语句的翻译
- 7.3 布尔表达式的翻译
- 7.4 控制语句的翻译
- 7.5 过程调用的处理

第七章 语义分析和中间代码生成

- 大纲要求
- 掌握：重点掌握两种中间语言：后缀式、三地址代码，掌握赋值语句的翻译、布尔表达式的翻译、控制语句的翻译、过程调用等语句翻译及中间代码生成方法。
- 理解：三地址中间语言的语法；语法制导定义与翻译模式的理解。
- 了解：DAG图、三地址代码的存储形式。

7.3 布尔表达式的翻译

7.3 布尔表达式的翻译

- 布尔表达式 Boolean Expression

  布尔表达式由布尔运算符号组成

  布尔运算符：&&, ||, and !

  C语言中，以上布尔运算符分别是AND, OR, 和 NOT

- 关系表达式 Relational Expression

  形如：E_1 **relop** E_2 (Relational Operation)

  其中E是数算术表达式 (Arithmetic expression)

7.3 布尔表达式的翻译

- 布尔表达式

  程序设计语言中，布尔表达式有两个基本的作用

① 计算逻辑值

布尔表达式可以表示为ture或false为值的结果。

② 条件表达式

对if-else-statements和while-statements的翻译往往是跟布尔表达式的翻译绑定的。

- 产生布尔表达式的文法:

  E→E or E | E and E | not E | (E) | id **relop** id | id

7.3 布尔表达式的翻译

- 算符的结合性和优先性

  关系运算符（ relop ）均相等且高于布尔运算符

  6种关系运算符：<, <=, =, !=, >, or >=

  or和and都是左结合

  or(||)的优先级最低，其次是and(&&)，

  not (!)的优先级最高。

7.3 布尔表达式的翻译

- 计算布尔式通常采用两种方法

第一种：如同计算算术表达式一样，一步步算。

通常用1表示真，0表示假。

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

9

7.3 布尔表达式的翻译

- 计算布尔式通常采用两种方法

第二种：采用某种优化措施如果有E_1 or E_2，当E_1的值可以确定为真时，则可以确定整个表达式的值是真，不必计算E_2

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

10

7.3.1 数值表示法

7.3.1 数值表示法

- 布尔式的**数值表示法**

a or b and not c 翻译成下列三地址序列：

T_1 := not c
T_2 := b and T_1
T_3 := a or T_2

用1表示真，用0表示假

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:

7.3.1 数值表示法

- 产生布尔式的三地址代码的翻译模式

$E \rightarrow E_1$ 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)}

emit是一个过程，作用是将三地址代码送到输出文件中

13

7.3.1 数值表示法

- 产生布尔式的三地址代码的翻译模式

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

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

E→id { E.place:=id.place}

14

7.3.1 数值表示法

- 产生布尔式的三地址代码的翻译模式

100: if a < b goto ***103***
101: T:=0
102: goto ***104***
103: T:=1
104:

E.place := newtemp;
emit('if' id_1.place *relop* id_2. place 'goto' nextstat+3);
emit(E.place ':=' '0');
emit('goto' nextstat+2);
emit(E.place':=' '1') }

nextstat给出输出序列中下一条三地址语句的位置，
nextstat初始为当前地址

每产生一条三地址语句后，过程emit便把nextstat加1
（函数emit执行之后，nextstat自动加1）

15

7.3.1 数值表示法

a<b or c<d and e<f的三地址代码的翻译结果

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

$E \rightarrow id_1$ *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') }

16

7.3.1 数值表示法

a<b or c<d and e<f的三地址代码的翻译结果

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

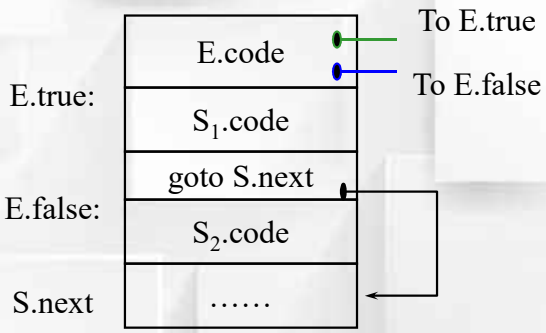
$E \rightarrow E_1$ or E_2
{ E.place := newtemp;
emit(E.place ':=' E_1.place 'or' E_2.place)}

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

7.3.2 作为条件控制的布尔式翻译

7.3.2 作为条件控制的布尔式翻译

- 条件语句 if E then S_1 else S_2
  赋予作为条件控制的E两种出口:一是“真”出口，出向S_1;
  一是“假”出口，出向S_2

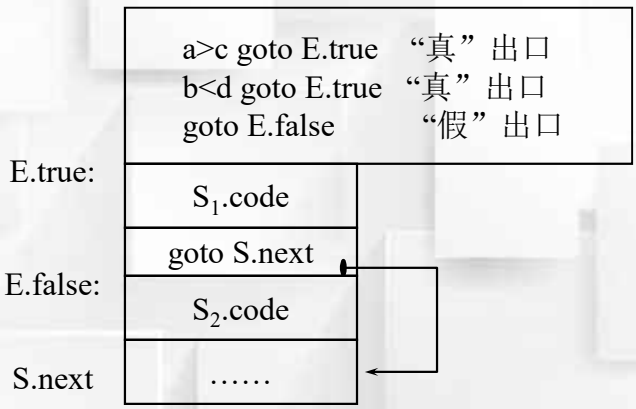


if-then-else语句的代码结构

19

7.3.2 作为条件控制的布尔式翻译

- 例：把语句: if a>c or b <d then S_1 else S_2



if-then-else语句的代码结构

20

7.3.2 作为条件控制的布尔式翻译

- 产生布尔表达式三地址代码的语义规则

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

Newlabel是存的谁的地址标号？

是存的E2的地址标号

7.3.2 作为条件控制的布尔式翻译

- 产生布尔表达式三地址代码的语义规则

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.false; E.code := E_1.code ‖ gen(E_1.true ‘:’) ‖ E_2.code

Newlabel是存的谁的地址标号？

是存的E2的地址标号

7.3.2 作为条件控制的布尔式翻译

- 产生布尔表达式三地址代码的语义规则

产生式	语义规则
• $E \rightarrow not\ E_1$	• E.true := E.false E1.false := E.true E.code := E1.code
• $E \rightarrow (E_1)$	• E1.true := E.true E1.false := E.false E.code := E.code

7.3.2 作为条件控制的布尔式翻译

- 产生布尔表达式三地址代码的语义规则

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)

7.3.2 作为条件控制的布尔式翻译

- 布尔表达式语义规则翻译
  需要两遍扫描，第一遍建立语法树，第二遍对语法树进行深度优先遍历，进行语义规则种规定的翻译。

- 通过一遍扫描产生布尔表达式的代码
  在实现三地址代码时，采用**四元式**形式实现
  把四元式存入一个数组中，**数组下标**就代表四元式的标号
  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

7.3.2 作为条件控制的布尔式翻译

- 布尔表达式语义规则翻译
  通过一遍扫描产生布尔表达式和控制流语句的代码实现的主要问题：
  当生成转移语句时，不知道该语句将要转移的标号

  解决方案：**回填技术(Backpatching)**。
  形成跳转指令时，暂时不确定跳转目标，而是建立一个链表，把转向该目标的跳转指令的标号键入这个链表。
  一旦目标确定之后，再把它填入有关的跳转指令。

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

  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

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

  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:

  (p) (x， x，x，0)　　链尾。0表示链末标志

  …

  (q) (x，x，x，p)

  …

  (r) (x，x，x，q)　　E. truelist =r

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

为了便于处理，引入下列语义变量和过程:

① 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。

② 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。

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

为了便于处理，引入下列语义变量和过程:

③ 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条链，作为函数值，回送合并后的链首（将由p_2指向的链表链接在p_1所指向的链表后，返回p_1）。

④ 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。

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

(1) E→ E_1 or M E_2
(2) | E_1 and M E_2
(3) | not E_1
(4) | (E_1)
(5) | id_1 relop id_2
(6) | id
(7) M→ε { **M.quad := nextquad** }

在文法中引入标记**非终结符M**，出现在E_1 or E_2或E_1 and E_2的E_2之前，记住下一个将要产生的四元式标号。

31

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

产生式

- E→ E_1 and M E_2

M存的E_2的四元式的标号

没回填的语义规则

- {E_1.true := newlabel;
  E_1.false := E.false;
  E_2.true := E.true;
  E_2.false := E.false;
  E.code := E_1.code || gen(E_1.true ':') || E_2.code;}

有回填的语义规则

- {backpatch(E_1.truelist, M.quad);
  E.truelist:=E_2.truelist;
  E.falselist:=merge(E_1.falselist, E_2.falselist);}

32

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

产生式

- $E \rightarrow E_1$ or M E_2

没回填的语义规则

- {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;}

有回填的语义规则

- {backpatch(E_1.falselist, M.quad);
  E.truelist:=merge(E_1.truelist, E_2.truelist);
  E.falselist:=E_2.falselist; }

33

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

(3) $E \rightarrow$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;}

34

7.3.2 作为条件控制的布尔式翻译

- 回填 Backpatching

(5) E→id_1 relop id_2
{ E.truelist:=makelist(nextquad);
E.falselist:=makelist(nextquad+1);
emit('j' relop ',' 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') ;}

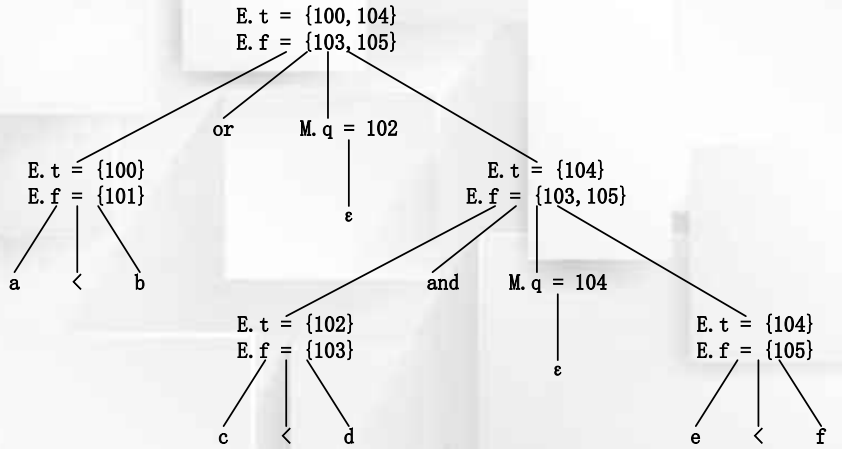
7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f

100	(j<, a, b, 0)		真出口
101	(j, -, -, 102)		
102	(j<, c, d, 104)	truelist	
103	(j, -, -, 0)		假出口
104	(j<, e, f, 100)		真出口
105	(j, -, -, 103)	falselist	假出口

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树



37

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

首先，对a < b产生

- 100： if a < b goto – (j<,a,b,0)
- 101： goto – (j,-,-,0)

(5) E→id_1 relop id_2

{ E.truelist:=makelist(nextquad);
E.falselist:=makelist(nextquad+1);
emit('j' relop ',' id_1.place ',' id_2.place',' '0');
emit('j, -, -, 0') ;}

此时，对应的E结点处的E.truelist记录了标号100，E.falselist记录了标号101，四元式的最后一位为0，表示这两处对应的标号还未生成，要通过回填来完成。

38

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树
对c < d产生

- 102： if c < d goto – (j<,c,d,0)
- 103： goto – (j,-,-,0)

此时，对应的E结点处的E.truelist记录了标号102，E.falselist记录了标号103，四元式的最后一位为0，表示这两处对应的标号还未生成，要通过回填来完成。And的优先级高于or的，此时已经分析到E→ E_1 and M E_2的E_1。

39

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树
再次，对e < f产生

- 104： if e < f goto – (j<,e,f,0)
- 105： goto – (j,-,-,0)

此时，对应的E结点处的E.truelist记录了标号104，E.falselist记录了标号105，表示这两处对应的标号还未生成，要通过回填来完成。

40

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

- 100： if a < b goto – (j<,a,b,0)
- 101： goto – (j,-,-,0)
- 102： if c < d goto 104 (j<,c,d,104) E_1.truelist
- 103： goto – (j,-,-,0)
- 104： if e < f goto – (j<,e,f,0) M.quad , E_2.truelist
- 105： goto – (j,-,-,0)

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

41

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

- 100： if a < b goto – (j<,a,b,0)
- 101： goto – (j,-,-,102) E_1.falselist
- 102： if c < d goto 104 (j<,c,d,104) M.quad , E_2.truelist
- 103： goto – (j,-,-,0)
- 104： if e < f goto – (j<,e,f,0)
- 105： goto – (j,-,-,0)

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

42

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

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

43

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

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

44

7.3.2 作为条件控制的布尔式翻译

例如：表达式a < b or c < d and e < f带注释的分析树

如果分别用L_1和L_2表示整个表达式的真、假两个出口

100： if a < b goto L_1
101： goto 102
102： if c < d goto 104
103： goto L_2
104： if e < f goto L_1
105： goto L_2

第七章 小结

第七章 小结

- 7.3 布尔表达式的翻译

Coursework

- 7.1 给出下面表达式的逆波兰表示（后缀式）

a*(-b+c)	not A or not(C or not D)
a+b*(c+d/e)	(A and B) or (not C or D)
-a+b*(-c+d)	(A or B) and (C or not D and E)
if (x+y)*z	then (a+b)↑c else a↑b↑c

- 7.2 请将表达式 – (a+b)*(c+d) – (a+b+c) 分别表示成三元式、间接三元式和四元式序列。

Coursework

- 7.3 按书上7.3节所说的办法，写出下面赋值句

  A:=B*(-C+D)

  的自下而上语法制导翻译过程。给出所产生的三地址代码。

- 7.4 写出下面赋值句的三地址代码

  A[i,j]:=B[i, j]+C[A[k, L]]+D[i+j]

  A是10 × 20的数组，即n_1=10，n_2=20，取w=4

- 7.5 按书上7.4.2节的办法，写出布尔式A or (B and not(C or D))的四元序列。

49

Coursework

- 7.6 用书上7.5.1节的办法，把下面的语句翻译成四元式序列：

```
while A< C  and B < D do
 if A=1 then C:=C+1 else
        while A≤ D do A:=A+2;
```

- 7.7 请给出

```
if A and B and C > D then
    if A <B then F:=1
   else F:=0
else G:=G+1;
```

的四元式序列，翻译过程中，采用then 与else 的最近匹配原则。

50

Coursework

7.8 对下面的文法，只利用综合属性获得类型信息。请给出该文法各个产生式的语义子程序。

D →L,id | L

L →T id

T →int | real