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。

7.4 控制语句的翻译

7.4 控制语句的翻译

- 控制语句的翻译Flow-of-Control Statement

  针对if – then, if – then – else, while –语句的翻译

  例如：通过一遍扫描对如下控制语句进行翻译

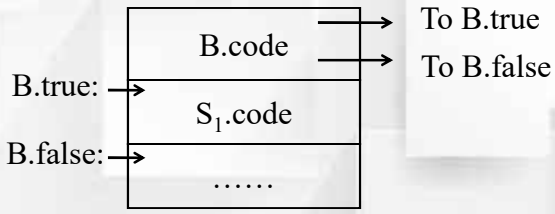
  $S \rightarrow$ if (B) then S_1
  | if (B) then S_1 else S_2
  | while (B) S_1

  其中B为布尔表达式。非终结符S是陈述statement

7.4 控制语句的翻译

- If – then 语句的代码结构

  S → if (B) then S_1

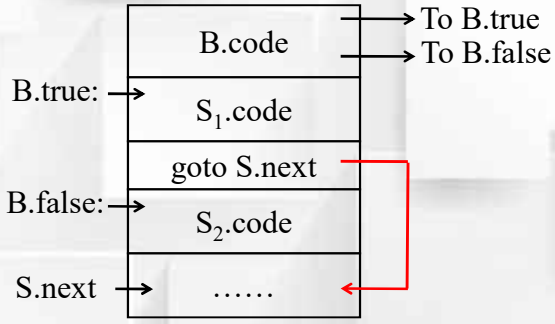


if-then语句的代码结构

7.4 控制语句的翻译

- If – then – else 语句的代码结构

  S → if (B) then S_1 else S_2

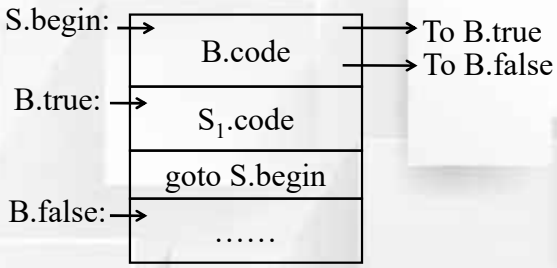


if-then-else语句的代码结构

7.4 控制语句的翻译

- While – do 语句的代码结构

  S → While (B) S_1

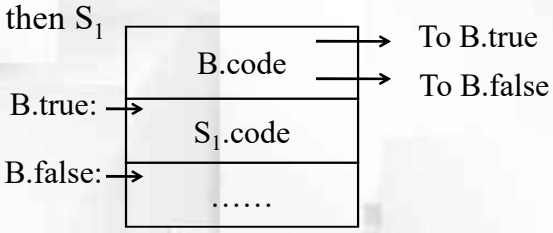


while-语句的代码结构

7.4 控制语句的翻译

- 控制流语句的属性文法　S → if (B) then M S_1

  产生式：S → if (B) then S_1



语义规则：

{ B.true := newlabel();
  B.false := S.next;　　S_1.next := S.next;
  S.code := B.code || gen(B.true ':') || S_1.code;}

建立一个新标号B.true，用它标识S_1的代码的第一条指令

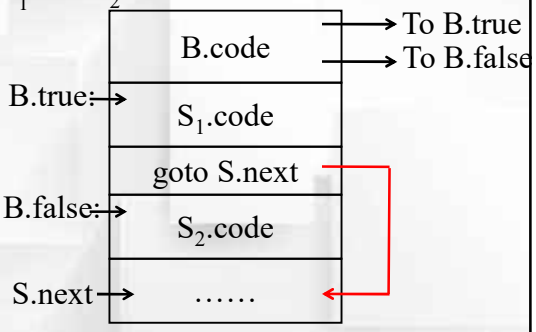
7.4 控制语句的翻译

- 控制流语句的属性文法　S → if (B) then M S_1 N else M S_2

  产生式：S → if (B) then S_1 else S_2

  语义规则：

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



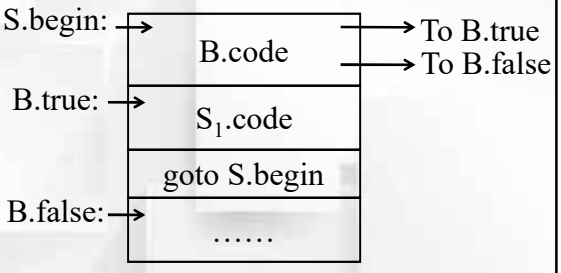
7.4 控制语句的翻译

- 控制流语句的属性文法　S → while M (B) do M S_1

  产生式：S → while (B) do S_1

  语义规则：

{ S.begin := newlabel();
B.true := newlabel();
B.false := S.next;
S_1.next := S.begin;
S.code := gen(S.begin ':')
|| B.code || gen(B.true ':')
|| S_1.code || gen('goto' S.begin);}



7.4 控制语句的翻译

- 控制流语句的属性文法

产生式：$S \rightarrow S_1\ S_2$

语义规则：{S_1.next := newlabel();

S_2.next := S.next;

S.code := S_1.code || gen(S_1.next ':') || S_2.code;}

产生式：$P \rightarrow S$

语义规则：{S.next := newlabel();

P.code := S.code || gen(S.next ':')

7.4 控制语句的翻译

- 控制流语句带回填技术的翻译

产生式：S → if (B) then M S_1

不带回填的语义规则{ B.true := newlabel();

B.false := S_1.next = S.next;

S.code := B.code || gen(B.true ':') || S_1.code;}

带回填的语义规则：{backpatch(B.truelist, M.quad);

S.nextlist := merge(B.falselist, S_1.nextlist);}

M → ε　M属性 M.quad, 把下一个四元式的标号赋给属性M.quad

7.4 控制语句的翻译

- 控制流语句带回填技术的翻译

产生式：$S \rightarrow$ if (B) then M_1 S_1 N else M_2 S_2

不带回填的语义规则{ B.true := newlabel();

B.false := newlabel(); S_1.next := S_2.next = S.next;

S.code := B.code || gen(B.true ':') || S_1.code|| gen('goto' S.next)

|| gen(B.false ':') || S_2.code;}

$N \rightarrow \varepsilon$ N属性 N.nextlist，N中是跳转指令

带回填的语义规则：{backpatch(B.truelist, M_1.quad);

backpatch(B.falselist, M_2.quad);

temp := merge(S_1.nextlist, N.nextlist);

S.nextlist := merge(temp, S_2.nextlist);}

7.4 控制语句的翻译

- 控制流语句带回填技术的翻译

产生式： $S \rightarrow$ while M_1 (B) do M_2 S_1

不带回填的语义规则{S.begin := newlabel();

B.true := newlabel(); B.false := S.next; S_1.next := S.begin

S.code := gen(S.begin ':') || B.code || gen(B.true ':')

|| S_1.code || gen('goto' S.begin);}

带回填的语义规则：{backpatch(S_1.nextlist, M_1.quad);

backpatch(B.truelist, M_2.quad);

S.nextlist := B.falselist;

emit('j , -, -,' M_1.quad);}

7.4 控制语句的翻译

- 控制流语句带回填技术的翻译

产生式：$L \rightarrow L_1\ M\ S$

带回填的语义规则：{backpatch(L_1.nextlist, M.quad);
L.nextlist := S.nextlist;}

产生式：$L \rightarrow S$

带回填的语义规则：{L.nextlist := S.nextlist;}

产生式：$M \rightarrow \varepsilon$

带回填的语义规则：{M.quad := nextquad;}

产生式：$N \rightarrow \varepsilon$

带回填的语义规则：{Nextlist := makelist(nextquat);
emit('j, -, -,' _);}

7.4 控制语句的翻译

- 控制流语句带回填技术的翻译

翻译语句while (a<b) 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

7.5 过程调用的处理

7.5 过程调用的处理

- 过程调用Procedures

  传递参数（传递实参）

  转子，把程序控制转移到子程序(过程段)

7.5 过程调用的处理

- 过程调用中实参的两种形式

  ① 变量或数组元素，如int a or int a[]，直接传递地址

  ② 表达式，如A+B，先计算值并存放再某个临时单元T中，然后传送T的地址

  传递实参的目的：把实参的地址逐一放在转子的指令前。

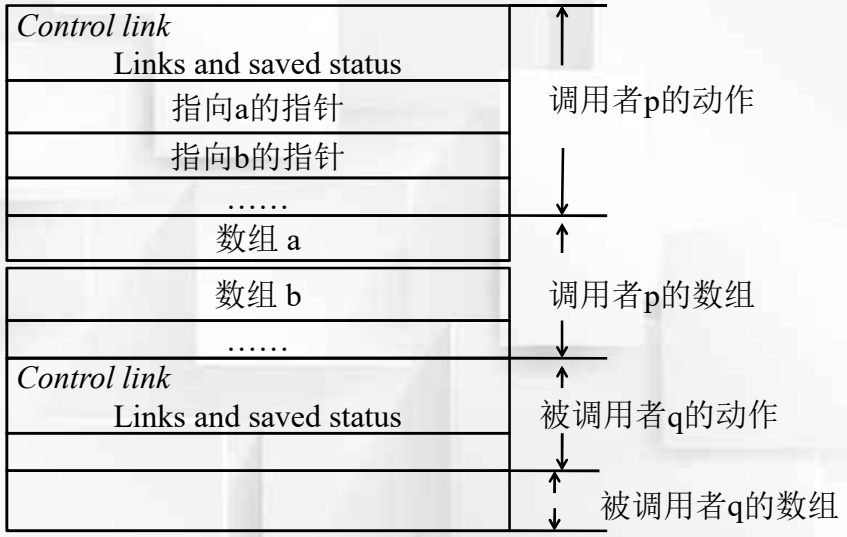
7.5 过程调用的处理

- 过程调用中实参的传递过程

  过程：在被调用的子程序过程中，相应每个形式参数都有一个单元(形式单元)用来存放相应的实在参数的地址。这个结构是**队列**。

  子程序对实参的调用：子程序中对形式参数的任何引用都当作是对形式单元的间接访问。子程序段的第一步工作就是把实在参数的地址取到对应的形式单元中，然后，再开始执行本段中的语句。

7.5 过程调用的处理



7.5 过程调用的处理

- 过程调用文法

(1) S → call id (Elist)
(2) Elist → Elist, E
(3) Elist → E

队列中的每一项生成一条param语句

Call S(A+B,Z)将被翻译成：

- T:=A+B //计算A+B置于T中的代码
- K-3: Param T //第一个实参地址
- K-2: Param Z //第二个实参地址
- K-1: Call S //转子指令
- K: Result //return 结果

7.5 过程调用的处理

- 过程调用文法

3. Elist→E
   { 初始化queue仅包含E.place }
2. Elist→Elist, E
   { 将E.place加入到queue的队尾 }
1. S→call id (Elist)
   { for 队列queue中的每一项p do
   emit(‘param’ p);
   emit(‘call’ id.place) }

第七章 小结

第七章 小结

- 7.4 控制语句的翻译
- 7.5 过程调用的处理

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。

29

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。

30

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