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。

7.2.2 数组元素的引用

7.2.2 数组元素的引用

- 数组在C语言中的定义

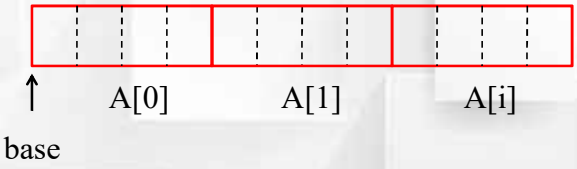
  数组包含给定类型的一些对象，并将这些对象依次存储在连续的内存空间中。每个独立的对象被称为数组的元素（element）。

  数组元素存储在一个连续的存储块中，根据数组元素的下标数字可以快速地查找每个元素。

7.2.2 数组元素的引用

- 一维数组元素的地址计算

数组A的基址用base表示，则base就是A[0]的地址，即数组A的首地址。

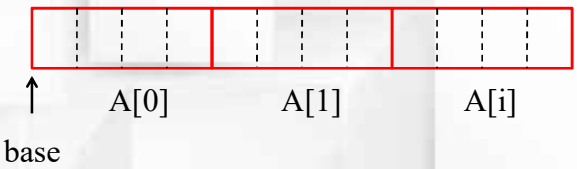


每个数组元素的宽度为3个字符

7

7.2.2 数组元素的引用

- 一维数组元素的地址计算



假如数组A中每个元素的宽度为w，则数组A中第i个元素的地址为：

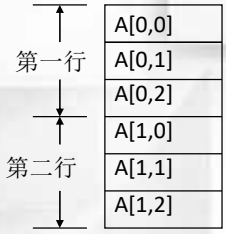
$$base + i \times w$$

如果每个数组元素的宽度为w=3，则A[1]的地址为base + 1 ×3

8

7.2.2 数组元素的引用

- 二维数组元素的地址计算



假如每维的长度已知为w_1，每个元素宽度为w_2

则A[i_1, i_2] (A[2,1]) 的地址为：

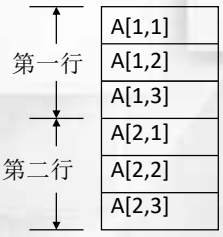
base + $i_1 \times w_1 + i_2 \times w_2$

假如该数组有k维(k行)，每行的长度w_1和每个元素的宽度w_2的通用表达形式为w_j, 1<=j<=k

则地址为：base + $i_1 \times w_1 + i_2 \times w_2 + \ldots + i_k \times w_k$

7.2.2 数组元素的引用

- 二维数组元素的地址计算



假如每维的元素个数有low和high标记下界和上界，则相对地址计算：

上界–下界+1

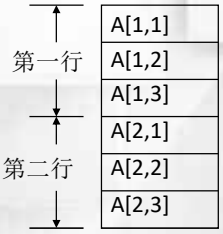
例如，第二维的元素的个数为

$n_2 = high_2 - low_2 + 1$

假如每个元素的长度为w，则数组元素A[i_1,i_2] (A[2,1]) 的相对地址：　　base + ((i_1-low_1)× n_2+i_2–low_2) × w

7.2.2 数组元素的引用

- 二维数组元素的地址计算



假如每个元素的长度为w，则数组元素的相对地址：

base + ((i_1-low_1)× n_2+i_2 –low_2) × w

把i_1-low_1用i_1表示， i_2-low_2用i_2表示

假如该数组有k维，则地址为：

base +((… ((i_1 × n_2 + i_2)× n_3 + i_3)…) × n_k + i_k)× w

7.2.2 数组元素的引用

- 二维数组元素的地址计算

通俗化一些，数组元素不是以数字0开始
我们用low, low+1, low+2….high表示
base就是A[low] 的地址
则A[i]：base + (i – low) × w

更通俗化一点，用i × w + c 表示，则

base + (i – low) × w = i × w + c

子表达式c = base – low × w

7.2.2 数组元素的引用

- 符号表中的数组元素地址计算

  $i \times w + c$，w是数组中元素的宽度，c是子表达式

  $c = base - low \times w$

  当w的大小一定时，该子表达式可以在处理数组说明时**提前**算出，则数组元素A[i]的相对地址也能计算，在c的计算结果加上$i \times w$。

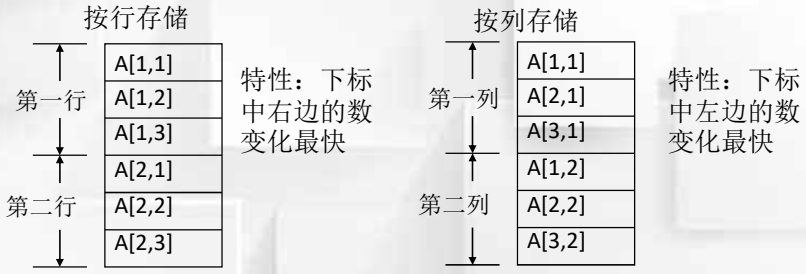
有一种特例不能提前计算，当数组元素的宽度是动态变化时

7.2.2 数组元素的引用

- 符号表中的数组元素的存储方式

  上述的地址计算都是基于**按行存储**，例如Pascal和C语言。

  有的语言是**按列存储**，例如FORTRAN语言。



7.2.2 数组元素的引用

- 数组元素的引用翻译

假如该数组有k维(k行)，则相对地址为：

base + $i_1 \times w_1 + i_2 \times w_2 + \ldots + i_k \times w_k$

或

base +((… (($i_1 \times n_2 + i_2$)× $n_3 + i_3$)…) × $n_k + i_k$)× w

数组元素引用代码生成的**主要问题**：把该式与数组引用的文法联系起来。

7.2.2 数组元素的引用

- 数组元素的引用翻译(书上版本)
- id出现的地方也允许下面产生式中的L出现

L → id [Elist] | id

Elist→Elist,E | E　多个维度的下标

为了便于语义处理，使得对Elist翻译时能随时知道数组名id的全部信息。

文法改写为

L→Elist] | id

Elist→Elist,E | id [E

数组名和最左边的下标表达式结合在一起，使得符号表中数组名作为Elist的综合属性array传递。

7.2.2 数组元素的引用

- 数组元素的引用翻译(书上版本)
- 引入下列语义变量或语义过程:
  - ✓ Elist.ndim： Elist的维数（下标表达式的个数）
  - ✓ Elist.place ： 表示临时变量，用来临时存放由Elist中的下标表达式计算出来的值
  - ✓ limit(array，j) ： 函数过程，它给出数组array的第j维的长度
  - ✓ Invariant(array)取得数组的地址计算的不变部分，即:
  - ✓ C=Base-((…((low$_1$n$_2$+low$_2$)n$_3$+low$_3$)…)n$_k$+low$_k$)*w
  - ✓ E.array为E的综合属性，表示数组名。

17

7.2.2 数组元素的引用

- 数组元素的引用翻译(书上版本)
- 非终结符L有两项语义值
  - –L.place：
    - 若L为简单变量i, 指变量i的符号表入口
    - 若L为下标变量，指存放CONSPART的临时变量的整数码
  - –L.offset ：
    - 若L为简单变量，null，
    - 若L为下标变量，指存放VARPART的临时变量的整数码

18

7.2.2 数组元素的引用

- 数组元素的引用翻译(书上版本)

(1) S→L:=E
(2) E→E+E
(3) E→(E)
(4) E→L
(5) L→Elist]
(6) L→id
(7) Elist→ Elist, E
(8) Elist→id [E

7.2.2 数组元素的引用

- 三地址代码的生成 (书上版本)

(1) S→L:=E
{ if L.offset=null then /*L是简单变量*/
emit(L.place ':=' E.place)
else emit(L.place ' [' L.offset ']' ':=' E.place)}

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

7.2.2 数组元素的引用

- 三地址代码的生成 (书上版本)

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

(4) E→L
{ if L.offset=null then
E.place:=L.place
else begin
E.place:=newtemp;
emit(E.place ':=' L.place '[' L.offset ']')
end
}

7.2.2 数组元素的引用

- 三地址代码的生成 (书上版本)

(5) L→Elist]
{ L.place:=newtemp;
emit(L.place ':=' Elist.array '－' C);
L.offset:=newtemp;
emit(L.offset ':=' w '*' Elist.place) }

C是预先计算好的值,
Elist.array是base的地址

(6) L→id { L.place:=id.place; L.offset:=null }

7.2.2 数组元素的引用

- 三地址代码的生成 (书上版本)

(7) Elist→ $Elist_1$, E
{ t:=newtemp;
m:=$Elist_1$.ndim+1;
emit(t ':=' $Elist_1$.place '*' limit($Elist_1$.array, m));
emit(t ':=' t '+' E.place);
Elist.array:= $Elist_1$.array;
Elist.place:=t;
Elist.ndim:=m
}

✓ Elist.ndim ：Elist的维数（下标表达式的个数）

$low_1 n_2$

$low_1 n_2 + low_2$

✓ Elist.place ：表示临时变量，用来临时存放由Elist中的下标表达式计算出来的值

23

7.2.2 数组元素的引用

- 三地址代码的生成 (书上版本)

(8) Elist→id [E
{ Elist.place:=E.place;
Elist.ndim:=1;
Elist.array:=id.place }

24

7.2.2 数组元素的引用

- 例：A是10 × 20的数组，即n_1=10，n_2=20，取w=4，数组的第一个元素是A[1, 1]，则有：
  - C := (($low_1 \times n_2$)+low_2) × w = (1 × 20+1) × 4 = 84
  - 对赋值语句x:=A[y , z]的三地址语句序列是：
  - (其中每个变量，用它们的名字来代替id.place)

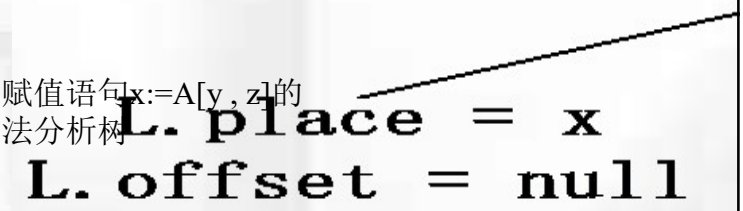
n是某个维度中元素的个数

T_1 := y*20　　$i_1n_2 + i_2$
T_1 :=T_1+z
T_2 := A–84　　base – C
T_3 := 4*T_1　　$(i_1n_2 + i_2)*w$
T_4 := T_2[T_3]
x := T_4

25

(1) S→L:=E
(2) E→E+E
(3) E→(E)
(4) E→L
(5) L→Elist]
(6) L→id
(7) Elist→ Elist, E
(8) Elist→id [E

对赋值语句x:=A[y , z]的语法分析树



T_1 := y*20
T_1 :=T_1+z
T_2 := A-84
T_3 := 4*T_1
T_4 := T_2[T_3]
x := T_4

26

7.2.2 数组元素的引用

- 数组元素的引用翻译(简化版本)

$L \rightarrow L[E] \mid id[E]$

假如该数组有k维(k行)，则相对地址为：
$base + i_1 \times w_1 + i_2 \times w_2 + \ldots + i_k \times w_k$

在C语言和Java中最小的下标是0。假如使用上述的公式计算，我们可以把下列文法翻译过来。

7.2.2 数组元素的引用

产生式
$S \rightarrow id := E \mid L = E$
$E \rightarrow E_1 + E_2 \mid id \mid L$
$L \rightarrow id[E] \mid L_1[E]$

L的地址用两个属性L.place和L.offset。
L.place是数组的基址
L.offset是数组元素相对于基址的偏移量，公式中$i_j \times w_j$的和

产生式	语义规则
$S \rightarrow id := E$	{gen(id.place ':=' E.place);}
$\mid L = E$	{gen(L.place '[' L.offset ']' ':=' E.place);}

非数组变量赋值

数组变量赋值，把E指向的值**赋值**给L指向的符号表中地址。

7.2.2 数组元素的引用

产生式
S → id := E | L = E
E → E_1 + E_2 | id | L
L → id [E] | L_1 [E]

L的地址用两个属性L.place和L.offset。
L.place是数组的基址
L.offset是数组元素相对于基址的偏移量，公式中$i_j \times w_j$的和

产生式	语义规则
E→E_1+ E_2	{E.place:= newtemp; gen(E.place ':=' E_1.place '+' E_2.place);}
| id	{gen(E.place ':=' id.place);}
| L	{E.place ':=' newtemp; gen(L.place '[' L.offset ']' ':=' E.place);}

E.Place是新的临时存储地址，保存L.place[L.offset]指向的值

29

7.2.2 数组元素的引用

产生式
S → id := E | L = E
E → E_1 + E_2 | id | L
L → id [E] | L_1 [E]

L.type是L产生的子集类型，对于类型t，宽度为t.width。这里的类型被当成属性值，t.elem是数组元素的类型。

产生式	语义规则
L → id [E]	{L.place:= id.place; L.type := L.place.type.elem; L.offset := newtemp; gen(L.offset ':=' E.place '*' L.type.width);}

数组元素的类型

数组元素的大小

30

7.2.2 数组元素的引用

产生式

$S \rightarrow id := E \mid L = E$

$E \rightarrow E_1 + E_2 \mid id \mid L$

$L \rightarrow id\ [\ E\ ] \mid L_1\ [\ E\ ]$

L.type是L产生的子集类型，对于类型t，宽度为t.width。这里的类型被当成属性值，t.elem是数组元素的类型。

产生式	语义规则

$L \rightarrow L_1\ [\ E\ ]$	{L.place ':=' L_1.place;
	L.type := L_1.place.type.elem;
	t := newtemp;
	L.offset := newtemp;
	gen(t ':=' E.place '*' L.type.width);
	gen(L.offset ':=' L_1.offset '+' t);}

相对偏移值，计算高维地址

31

7.2.2 数组元素的引用

- 例题：有一个2 ×3的整型数组，c, i, 和j都是整数。我们有a的类型是array(2, array(3, integer))。整数长度为4，则a的元素宽度w=24(2 ×3 ×4)；数组a[i]的类型是array(3, integer), a[i]元素宽度为w_1=12。数组a[i][j]的类型是整型。求表达式c + a[i][j]的语法分析树和三地址代码。

数组示分析: 数组a = {a[1], a[2]}； a[1] = {a_{11}, a_{12}, a_{13}}，a[2] = {a_{21}, a_{22}, a_{23}}； a[i][j] = {整型数}。

32

7.2.2 数组元素的引用

- 例题：表达式c + a[i][j]

的语法分析树

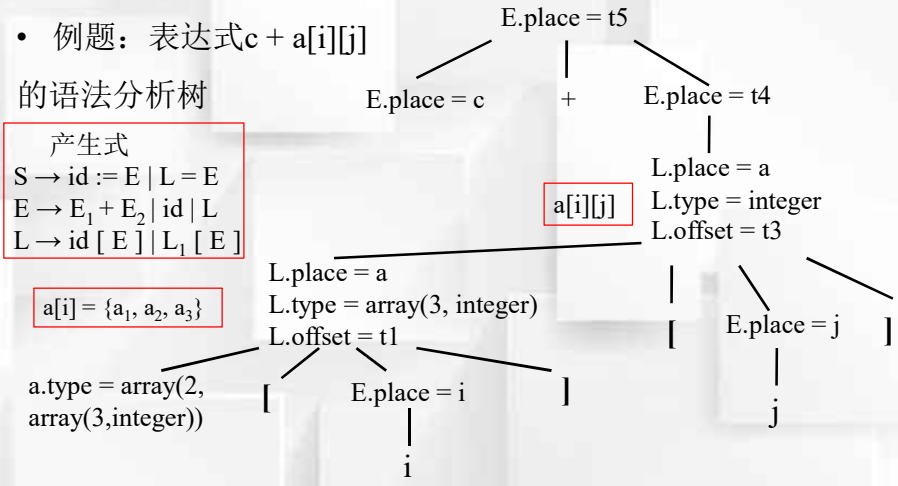
产生式

$S \rightarrow id := E \mid L = E$

$E \rightarrow E_1 + E_2 \mid id \mid L$

$L \rightarrow id [E] \mid L_1 [E]$

$a[i] = \{a_1, a_2, a_3\}$



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

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

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:

41

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

42

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}

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）

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') }

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)}

第七章 小结

第七章 小结

- 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