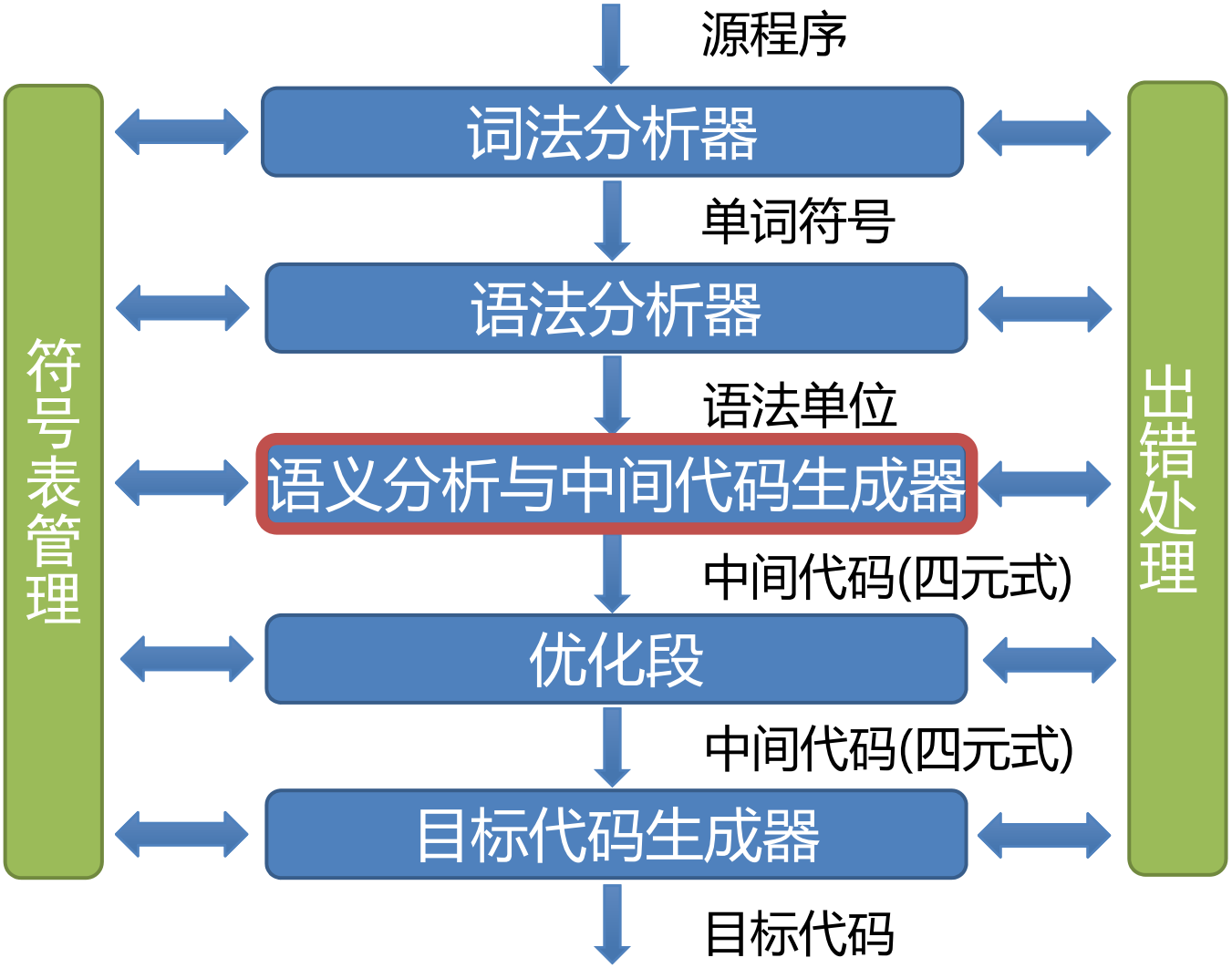
编译原理

赋值语句的翻译

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

编译原理

赋值语句的属性文法

简单算术表达式及赋值语句

- 赋值语句的形式
  - id:=E
- 赋值语句的意义(功能)
  - 对表达式E求值并置于变量T中
  - id.place:=T

赋值语句生成三地址代码的S-属性文法

- 非终结符号S有综合属性S.code，它代表赋值语句S的三地址代码
- 非终结符号E有两个属性
  - E.place表示存放E值的单元的名字(地址)
  - E.code表示对E求值的三地址语句序列
- 函数newtemp的功能是，每次调用它时，将返回一个不同的临时变量名字,如T_1,T_2,…

赋值语句生成三地址代码的S-属性文法

产生式	语义规则
$S \rightarrow id:=E$	S.code:=E.code || gen(id.place ':=' E.place)
$E \rightarrow E_1+E_2$	E.place:=newtemp; $E.code:=E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '+' E_2.place)
$E \rightarrow E_1*E_2$	E.place:=newtemp; $E.code:=E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '*' E_2.place)
$E \rightarrow -E_1$	E.place:=newtemp; $E.code:=E_1.code \,\|\|$ gen(E.place ':=' 'uminus' E_1.place)
$E \rightarrow (E_1)$	$E.place:=E_1.place$; $E.code:=E_1.code$
$E \rightarrow id$	E.place:=id.place; E.code=' '

编译原理

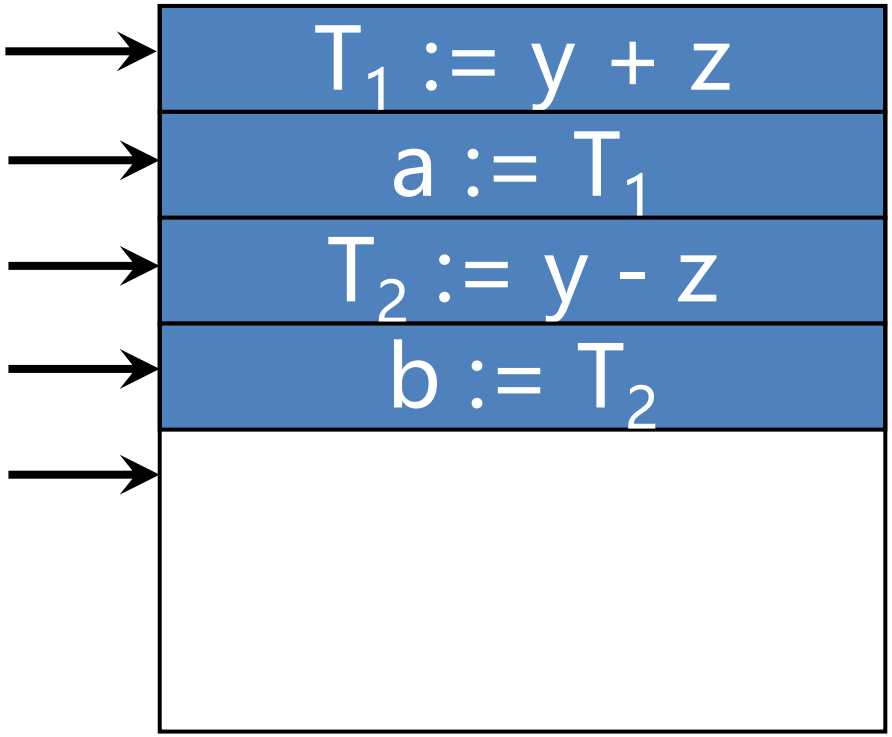
赋值语句的翻译模式

赋值语句生成三地址代码的S-属性文法

产生式	语义规则
$S \rightarrow id:=E$	S.code:=E.code || gen(id.place ':=' E.place)
$E \rightarrow E_1+E_2$	E.place:=newtemp; $E.code:=E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '+' E_2.place)
$E \rightarrow E_1*E_2$	E.place:=newtemp; $E.code:=E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '*' E_2.place)
$E \rightarrow -E_1$	E.place:=newtemp; $E.code:=E_1.code \,\|\|$ gen(E.place ':=' 'uminus' E_1.place)
$E \rightarrow (E_1)$	$E.place:=E_1.place$; $E.code:=E_1.code$
$E \rightarrow id$	E.place:=id.place; E.code=' '

产生赋值语句三地址代码的翻译模式

- 过程emit将三地址代码送到输出文件中



产生赋值语句三地址代码的翻译模式

$S \rightarrow id := E$ { p:=lookup(id.name);
if p≠nil then emit(p ':=' E.place)
else error }

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

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

产生式	语义规则
$S \rightarrow id := E$	S.code:=E.code || gen(id.place ':=' E.place)
$E \rightarrow E_1 + E_2$	E.place:=newtemp; E.code:=E_1.code || E_2.code || gen(E.place ':=' E_1.place '+' E_2.place)
$E \rightarrow E_1 * E_2$	E.place:=newtemp; E.code:=E_1.code || E_2.code || gen(E.place ':=' E_1.place '*' E_2.place)

产生赋值语句三地址代码的翻译模式

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

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

$E \rightarrow$ id { p:=lookup(id.name);
if p≠nil then E.place:=p
else error }

产生式	语义规则
$E \rightarrow -E_1$	E.place:=newtemp; E.code:=E_1.code || gen(E.place ':=' 'uminus' E_1.place)
$E \rightarrow (E_1)$	E.place:=E_1.place; E.code:=E_1.code
$E \rightarrow$ id	E.place:=id.place; E.code=' '

编译原理

数组元素引用的翻译

数组元素的引用

- 数组元素地址的计算
  - X := A[i_1,i_2,...,i_k] + Y
  - A[i_1,i_2,...,i_k] := X + Y

数组元素地址计算

- 设A为n维数组，按行存放，每个元素宽度为w
  - low_i 为第i维的下界
  - up_i 为第i维的上界
  - n_i 为第i维可取值的个数($n_i = up_i - low_i + 1$)
  - base为A的第一个元素相对地址
- 元素$A[i_1,i_2,\ldots,i_k]$相对地址公式

$$((\ldots i_1\ n_2+i_2)n_3+i_3)\ldots)n_k+i_k)\times w\ +$$

$$base-((\ldots((low_1\ n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$$

- $C = ((\ldots((low_1\ n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$

可变部分

不变部分

数组元素地址计算

- 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

为了便于处理，文法改写为

  L → Elist] | id

  Elist → Elist, E | id [E

$$((...i_1\ n_2+i_2)n_3+i_3)...)n_k+i_k)\times w +$$

$$base-((...((low_1\ n_2+low_2)n_3+low_3)...)n_k+low_k)\times w$$

数组元素地址计算

- 引入下列语义变量或语义过程
  - Elist.ndim: 下标个数计数器
  - Elist.place: 保存临时变量的名字，这些临时变量存放已形成的Elist中的下标表达式计算出来的值
  - Elist.array: 保存数组名
  - limit(array，j) ： 函数过程，它给出数组array的第j维的长度

$$((...i_1\ n_2+i_2)n_3+i_3)...)n_k+i_k)\times w\ +$$
$$base-((...((low_1\ n_2+low_2)n_3+low_3)...)n_k+low_k)\times w$$

数组元素地址计算

- 代表变量的非终结符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，指存放不变部分的临时变量的名字
  - L.offset
    - 若L为简单变量，null
    - 若L为下标变量，指存放可变部分的临时变量的名字

$$((...i_1\ n_2+i_2)n_3+i_3)...)n_k+i_k)\times w\ +$$
$$base-((...((low_1\ n_2+low_2)n_3+low_3)...)n_k+low_k)\times w$$

带数组元素引用的赋值语句

(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

带数组元素引用的赋值语句翻译模式

(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)}

带数组元素引用的赋值语句翻译模式

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

(4) E→L

```
{  if L.offset=null then /*L是简单变量*/
        E.place:=L.place
   else begin
        E.place:=newtemp;
        emit(E.place ':=' L.place '[' L.offset ']' )
    end
}
```

带数组元素引用的赋值语句翻译模式

(8) Elist→id [E

{ Elist.place:=E.place;

Elist.ndim:=1;

Elist.array:=id.place }

$A[i_1, i_2, \ldots, i_k]$

$((\ldots i_1 n_2+i_2)n_3+i_3)\ldots)n_k+i_k)\times w +$

$base-((\ldots((low_1 n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$

带数组元素引用的赋值语句翻译模式

(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.place:=t;

Elist.ndim:=m

Elist.array:= $Elist_1$.array;

}

A[i_1,i_2,...,$\mathbf{i_m}$,...,i_k]

((...(...i_1 n_2+i_2)n_3+i_3) ...)$\mathbf{n_m}$+$\mathbf{i_m}$) ...)n_k+i_k)×w +

base-((...((low_1 n_2+low_2)n_3+low_3)...)n_k+low_k)×w

带数组元素引用的赋值语句翻译模式

(5) L→Elist]

{ L.place:=newtemp;

emit(L.place ':=' Elist.array ' - ' C);

L.offset:=newtemp;

emit(L.offset ':=' w '*' Elist.place) }

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

$A[i_1,i_2,\ldots,i_k]$

$((\ldots i_1\ n_2+i_2)n_3+i_3)\ldots)n_k+i_k) \times w +$

$base-((\ldots((low_1\ n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$

带数组元素引用的赋值语句翻译模式

(5) L→Elist]

{ L.place:=newtemp;

emit(L.place ':=' Elist.array ' - ' C);

L.offset:=newtemp;

emit(L.offset ':=' w '*' Elist.place) }

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

$A[i_1,i_2,\ldots,i_k]$

$((\ldots i_1\ n_2+i_2)n_3+i_3)\ldots)n_k+i_k) \times w +$

$base-((\ldots((low_1\ n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$

带数组元素引用的赋值语句翻译模式

(5) L→Elist]

{ L.place:=newtemp;

emit(L.place ':=' Elist.array ' - ' C);

L.offset:=newtemp;

emit(L.offset ':=' w '*' Elist.place) }

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

$A[i_1,i_2,\ldots,i_k]$

$((\ldots i_1\ n_2+i_2)n_3+i_3)\ldots)n_k+i_k) \times w +$

$base-((\ldots((low_1\ n_2+low_2)n_3+low_3)\ldots)n_k+low_k)\times w$

编译原理

类型转换

类型转换

- x := y + i*j，其中x、y为实型；i、j为整型
- 该赋值句产生的三地址代码为：

  T_1 := i int* j

  T_3 := inttoreal T_1

  T_2 := y real+ T_3

  x := T_2

类型转换

- 用E.type表示非终结符E的类型属性
- 产生式$E \rightarrow E_1$ op E_2的语义动作中关于E.type的语义规则可定义为：

{ if E_1.type=integer and E_2.type=integer
E.type:=integer
else E.type:=real }

产生式$E \rightarrow E_1 + E_2$的语义动作

{ E.place:=newtemp;
  if E_1.type=integer and E_2.type=integer then begin
    emit (E.place ':=' E_1.place 'int+' E_2.place);
    E.type:=integer
  end
  else if E_1.type=real and E_2.type=real then begin
    emit (E.place ':=' E_1.place 'real+' E_2.place);
    E.type:=real
  end

产生式$E \rightarrow E_1 + E_2$的语义动作

else if E_1.type=integer and E_2.type=real then begin
    u:=newtemp;
    emit (u ':=' 'inttoreal' E_1.place);
    emit (E.place ':=' u 'real+' E_2.palce);
    E.type:=real
end
else if E_1.type=real and E_2.type=integer then begin
    u:=newtemp;
    emit (u ':=' 'inttoreal' E_2.place);
    emit (E.place ':=' E_1.place 'real+' u);
    E.type:=real
end
else E.type:=type_error}

小结

- 赋值语句的翻译
  - 简单算术表达式及赋值语句
  - 数组元素的引用
  - 产生有关类型转换的指令