



编译原理

第七章 语义分析和中间代码产生

第七章 语义分析和中间代码产生

- 中间语言
- 赋值语句的翻译
- 布尔表达式的翻译
- 控制语句的翻译
- 过程调用的处理

第七章 语义分析和中间代码产生

- 中间语言
- 赋值语句的翻译
- 布尔表达式的翻译
- 控制语句的翻译
- 过程调用的处理

7.3 赋值语句的翻译

7.3.1 简单算术表达式及赋值语句

■ $id:=E$

- 对表达式 E 求值并置于变量 T 中值
- $id.place:=T$

从赋值语句生成三地址代码的 S- 属性文法

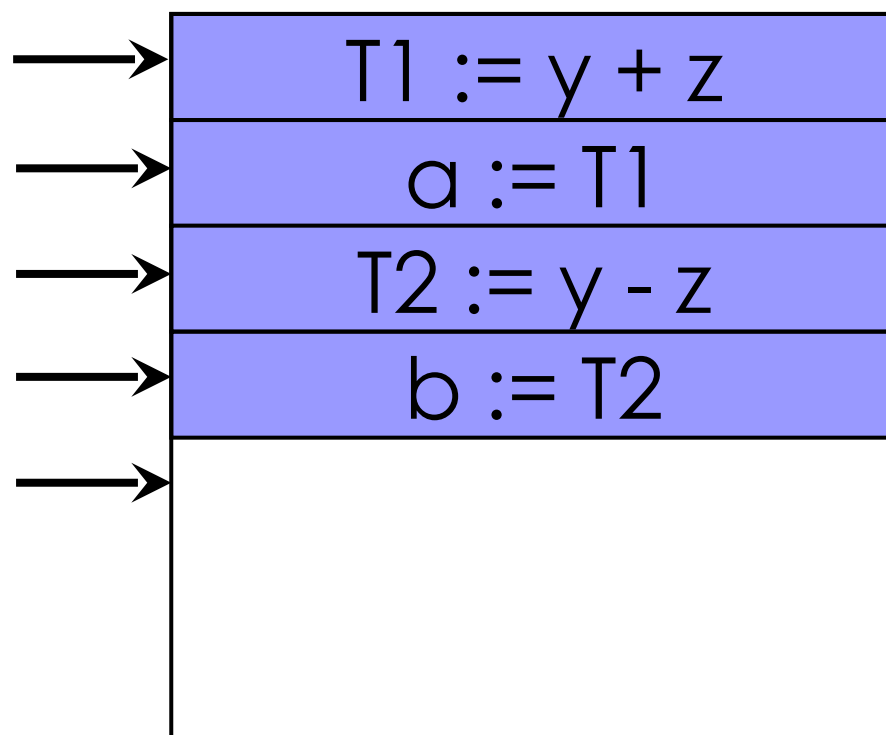
- 非终结符号 S 有综合属性 `S.code`，它代表赋值语句 S 的三地址代码
- 非终结符号 E 有如下两个属性
 - `E.place` 表示存放 E 值的名字
 - `E.code` 表示对 E 求值的三地址代码
 - 函数 `newtemp` 的功能是返回一个不同的临时变量名
- 计算思维的典型方法 -- 递归
 - 问题的解决又依赖于类似问题的解决，只不过后者的复杂程度或规模较原来的问题更小
 - 一旦将问题的复杂程度和规模化简到足够小时，问题的解法其实非常简单

为赋值语句生成三地址代码的 S- 属性文法定义

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

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



$S \rightarrow id := E$ $S.code := E.code \parallel \text{gen}(id.place \text{ ':=' } E.place)$
 $E \rightarrow E_1 + E_2$ $E.place := \text{newtemp};$
 $E.code := E_1.code \parallel E_2.code \parallel \text{gen}(E.place \text{ ':=' } E_1.place \text{ '+' } E_2.place)$
 $E \rightarrow E_1 * E_2$ $E.place := \text{newtemp};$
 ~~$E.code := E_1.code \parallel E_2.code \parallel \text{gen}(E.place \text{ ':=' } E_1.place \text{ '*' } E_2.place)$~~

产生赋值语句三地址代码的翻译模式

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

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

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

$E \rightarrow -E_1$	$E.place := newtemp;$ $E.code := E_1.code \parallel gen(E.place := 'uminus' E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place;$ $E.code := E_1.code$
$E \rightarrow id$	$E.place := id.place;$ $E.code := ' '$

产生赋值语句三地址代码的翻译模式

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

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

$a := uminus b$

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

7.3.2 数组元素的引用

- 数组元素地址的计算

- $X := A[i_1, i_2, \dots, i_k] + Y$

- $A[i_1, i_2, \dots, i_k] := X + Y$

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 $A[i_1, i_2, \dots, i_k]$ 相对地址公式

可变部分

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

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

不变部分

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

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

$$\text{base} - ((\cdots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \cdots) n_k + \text{low}_k) \times w$$

- id 出现的地方也允许下面产生式中的 L 出现

$$L \rightarrow \text{id} [\text{Elist}] \mid \text{id}$$

$$\text{Elist} \rightarrow \text{Elist}, E \mid E$$

为了便于处理，文法改写为

$$L \rightarrow \text{Elist}] \mid \text{id}$$

$$\text{Elist} \rightarrow \text{Elist}, E \mid \text{id} [E$$

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

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

- 引入下列语义变量或语义过程
 - Elist.ndim: 下标个数计数器
 - Elist.place: 表示临时变量，用来临时存放已形成的 Elist 中的下标表达式计算出来的值
 - Elist.array: 记录数组名
 - limit(array, j) : 函数过程，它给出数组 array 的第 j 维的长度

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

$$\text{base} - ((\cdots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \cdots) n_k + \text{low}_k) \times w$$


■ 每个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整数码

□ 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整数码



(1) $S \rightarrow L := E$

(2) $E \rightarrow E + E$

(3) $E \rightarrow (E)$

(4) $E \rightarrow L$

(5) $L \rightarrow \text{Elist }]$

(6) $L \rightarrow \text{id}$

(7) $\text{Elist} \rightarrow \text{Elist}, E$

(8) $\text{Elist} \rightarrow \text{id } [E$

带数组元素引用的赋值语句翻译模式

(1) $S \rightarrow L := E$

{ if $L.offset = \text{null}$ then /*L 是简单变量 */

emit($L.place := E.place$)

else emit($L.place$ '[' $L.offset$ ']' $:= E.place$)}

(2) $E \rightarrow E_1 + E_2$

{ $E.place := \text{newtemp}$;

emit($E.place := E_1.place + E_2.place$)}

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

(4) $E \rightarrow L$

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

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

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

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(8) $\text{Elist} \rightarrow \text{id} [E$

$\{ \text{Elist.place} := E.\text{place};$

$\text{Elist.ndim} := 1;$

$\text{Elist.array} := \text{id.place} \}$

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

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

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(7) $\text{Elist} \rightarrow \text{Elist}_1, E$

```
{  t:=newtemp;
   m:=Elist1.ndim+1;
   emit(t ':=' Elist1.place '*' limit(Elist1.array,m) );
   emit(t ':=' t '+' E.place);
   Elist.place:=t;
   Elist.ndim:=m
   Elist.array:= Elist1.array;
```

}

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

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

$\text{base} - ((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w$

(5) $L \rightarrow \text{Elist}$]

{ L.place := newtemp;

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

L.offset := newtemp;

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

(6) $L \rightarrow \text{id}$ { L.place := id.place;
L.offset := null }

类型转换

- 用 $E.type$ 表示非终结符 E 的类型属性
- 对应产生式 $E \rightarrow E_1 \text{ op } E_2$ 的语义动作中关于 $E.type$ 的语义规则可定义为：
 { if $E_1.type = \text{integer}$ and $E_2.type = \text{integer}$
 $E.type := \text{integer}$
 else $E.type := \text{real}$ }
- 算符区分为整型算符 int op 和实型算符 real
 op ,


■ $x := y + i*j$

其中 x 、 y 为实型； i 、 j 为整型。这个赋值句产生的三地址代码为：

$$T_1 := i \text{ int* } j$$
$$T_3 := \text{inttoreal } T_1$$
$$T_2 := y \text{ real+ } T_3$$
$$x := T_2$$

关于产生式 $E \rightarrow E_1 + E_2$ 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
```



```
else if E1.type=integer and E2.type=real then begin
    u:=newtemp;
    emit (u ':=' 'inttoreal' E1.place);
    emit (E.place ':=' u 'real+' E2.palce);
    E.type:=real
end
else if E1.type=real and E2.type=integer then begin
    u:=newtemp;
    emit (u ':=' 'inttoreal' E2.place);
    emit (E.place ':=' E1.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

作业

- P218-4 , 5
- 注: P218-5 , 设:
 - A 、 B : 10×20
 - C 、 D : 20
 - 宽度 $w = 4$
 - 下标从 1 开始