各种语言成分的语法及其翻译方案(示例)

1. 普通声明语句的翻译

下面是声明语句的文法：

$P \rightarrow$ **prog id** (**input**, **output**) D ; S

$D \rightarrow D$; D | $List$: T | **proc id** D ; S

$List \rightarrow List_1$, **id** | **id**

$T \rightarrow$ **integer** | **real** | **array** C of T_1 | $\uparrow T_1$ | **record** D

$C \rightarrow$ [**num**] C | ε

声明语句的翻译模式：

$P \rightarrow$ **prog id** (**input**, **output**){*offset* := 0}D ; S

$D \rightarrow D$; D

$D \rightarrow$ **id**: T{*enter* (**id**.*name*, T.*type*, *offset*); *offset*:= *offset* + T.*width*}

$T \rightarrow$ **integer**{T.*type* := *integer*; T.*width* := 4}

$T \rightarrow$ **real**{T.*type* :=*real*; T.*width* :=8}

$T \rightarrow$ **array** [**num**] of T_1{T.*type* := *array*(**num**.*val*, T_1.*type*);T.*width* := **num**.*val*×T_1.*width*}

$T \rightarrow \uparrow T_1${$T$.*type* := *pointer*($T_1$.*type*); T.*width* := 4}

2. 嵌套过程中声明语句的翻译

嵌套过程声明语句的产生式。

$P \rightarrow$ **prog id** (**input**, **output**) D ; S

$D \rightarrow D$; D | **id :** T | **proc id** ; D ; S (7.1)

嵌套过程声明语句的翻译模式：

$P \rightarrow$ **prog id** (**input**, **output**) MD;S{*addwidth*(*top*(*tblptr*),*top*(*offset*));
pop(*tblptr*);*pop*(*offset*)}

$M \rightarrow \varepsilon${$t$:= *mktable*(*nil*);*push*(t, *tblptr*); *push*(0, *offset*)}

$D \rightarrow D_1$;D_2

$D \rightarrow$ **proc id**; N D_1 ; S{t:= *top*(*tblptr*);*addwidth*(t, *top*(*offset*));*pop*(*tblptr*);
pop(*offset*); *enterproc*(*top*(*tblptr*), **id**.*name*,t)}

$D \rightarrow$ **id** : T{*enter*(*top*(*tblptr*), **id**.*name*, T.*type*, *top*(*offset*));
top(*offset*) := *top*(*offset*) + T.*width*}

$N \rightarrow \varepsilon${$t$:= *mktable*(*top*(*tblptr*)); *push*(t, *tblptr*); *push*(0, *offset*)}

3. 记录的翻译

下面是生成记录类型的产生式：

$T \rightarrow$ **record** D **end**

生成记录类型的翻译模式：

$T \rightarrow$ **record** L D **end** {T.*type* := *record*(*top*(*tblptr*));
T.*width* := *top*(*offset*);
pop(*tblptr*); *pop*(*offset*)}

$L \rightarrow \varepsilon${$t$:= *mktable*(*nil*); *push*(t, *tblptr*); *push*(0, *offset*)}

4. 赋值语句的翻译

下面是典型的赋值语句文法：

$S \rightarrow Left := E$

$E \rightarrow E_1 + E_2 \mid E_1 * E_2 \mid - E_1 \mid (E_1) \mid Left$

$Left \rightarrow Elist$] | **id**

Elist →*Elist*, *E* | **id** [*E* (7.2)

赋值语句的翻译模式：

⑴ *S*→*Left*:=*E*{**if** *Left.offset*=**null then** /**Left* 是简单变量 **id***/
gencode(*Left.addr* ':=' *E.addr*);
else
gencode(*Left.addr* '[' *Left.offset* '] ' ':=' *E.addr*)} /**Left* 是数组元素*/

⑵ $E \rightarrow E_1+E_2$ {*E.addr*:=*newtemp*;*gencode*(*E.addr* ':='E_1.*addr*'+'E_2.*addr*)}

⑶ $E \rightarrow (E_1)$ {*E.addr*:= E_1.*addr*}

⑷ *E*→*Left*{**if** *Left.offset*=**null then** /**Left* 是简单 **id***/
E.addr:= *Left.addr*
else begin /**Left* 是数组元素*/
E.addr:=*newtemp*;
gencode(*E.addr* ':=' *Left.addr* ' [' *Left.offset* ']')
end}

⑸ *Left*→*Elist*]{ *Left.addr*:=*newtemp*; /**Left* 是数组元素，因此存放基址和位移*/
Left.offset:=*newtemp*;
gencode(*Left.addr* ':=' *c*(*Elist.array*));
gencode(*Left.offset* ':=' *Elist.addr* '*' *width*(*Elist.array*))}

⑹ *Left*→**id**{*Left.addr*:=**id**.*addr*; *Left.offset*:=**null**}

⑺ $Elist \rightarrow Elist_1$, *E*{*t*:=*newtemp*;*m*:= $Elist_1$.*ndim*+1;
gencode(*t* ':=' $Elist_1$.*addr* '*' *limit*($Elist_1$.*array*, *m*)); /*计算 $e_{m-1} \times n_m$ */
gencode(*t* ':=' *t* '+' *E.addr*); /* 计算+ i_m */
Elist.array:= $Elist_1$.*array*;
Elist.addr:=*t*;
Elist.ndim:=*m*}

⑻ *Elist*→**id**[*E* {*Elist.array*:=**id**.*addr*; *Elist.addr*:= *E.addr*; *Elist.ndim*:=1}

5.各种控制结构的翻译

5.1 布尔表达式的翻译

布尔表达式的文法为：

⑴ $B \rightarrow B_1$ **or** $M\ B_2$

⑵ $B \rightarrow B_1$ **and** $M\ B_2$

⑶ $B \rightarrow$ **not** B_1

⑷ $B \rightarrow (B_1)$

⑸ $B \rightarrow E_1$ **relop** E_2

⑹ $B \rightarrow$ **true**

⑺ $B \rightarrow$ **false**

⑻ $M \rightarrow \varepsilon$

布尔表达式的翻译模式如下所示：

⑴ $B \rightarrow B_1$ **or** $M\ B_2$ { *backpatch*(B_1.*falselist*, *M.quad*);
B.truelist := *merge*(B_1.*truelist*, B_2.*truelist*);
B.falselist := B_2.*falselist*}

⑵ $B \rightarrow B_1$ **and** $M\ B_2$ {*backpatch*(B_1.*truelist*, *M.quad*);

B.truelist := B_2.*truelist*;
B.falselist := *merge*(B_1.*falselist*, B_2.*falselist*)}

⑶ B→**not** B_1 {*B.truelist* := B_1.*falselist*; *B.falselist* := B_1.*truelist*}

⑷ B→(B_1) {*B.truelist* := B_1.*truelist*; *B.falselist* := B_1.*falselist*}

⑸ B→E_1 **relop** E_2 {*B.truelist* :=*makelist*(*nextquad*);
B.falselist := *makelist*(*nextquad*+1);
gencode('if' E_1.*addr* **relop**.*op*E_1.*addr* 'goto –');
gencode('goto –')}

⑹ B→**true** {*B.truelist* := *makelist*(*nextquad*); *gencode*('goto –')}

⑺ B→**false** {*B.falselist* := *makelist*(*nextquad*); *gencode*('goto –')}

⑻ M→ε {*M.quad* := *nextquad*}

5.2 常用控制流语句的翻译

控制流语句 **if-then**，**if-then-else** 和 **while-do** 的文法为：

⑴ S→**if** B **then** S_1

⑵ S→**if** B **then** S_1 **else** S_2

⑶ S→**while** B **do** S_1

⑷ S→**begin** L **end**

⑸ S→A

⑹ L→L_1;S

⑺ L→S (7.9)

if-then，**if-then-else** 和 **while-do** 语句的翻译模式：

⑴ S→**if** B **then** M_1 S_1 N **else** M_2 S_2 {*backpatch*(*B.truelist*, M_1.*quad*);
backpatch(*B.falselist*, M_2.*quad*);
S.nextlist := *merge*(S_1.*nextlist*, *merge*(*N.nextlist*, S_2.*nextlist*))}

⑵ N→ε {*N.nextlist* := *makelist*(*nextquad*); *gencode*('goto –')}

⑶ M→ε {*M.quad* := *nextquad*}

⑷ S→**if** B **then** M S_1 {*backpatch*(*B.truelist*, *M.quad*);
S.nextlist := *merge*(*B.falselist*, S_1.*nextlist*)}

⑸ S→**while** M_1 B **do** M_2 S_1 {*backpatch*(S_1.*nextlist*, M_1.*quad*);
backpatch(*B.truelist*,M_2.*quad*);*S.nextlist*:=*B.falselist*; *gencode*('goto'M_1.*quad*)}

⑹ S→**begin** L **end** {*S.nextlist*:=*L.nextlist*}

⑺ S→A {*S.nextlist* := **nil**}

⑻ L→L_1;MS {*backpatch*(L_1.*nextlist*, *M.quad*); *L.nextlist* := *S.nextlist*}

⑼ L→S {*L.nextlist* := *S.nextlist*}

5.3 for 循环语句的翻译

for 循环语句的文法如下所示：

S → **for id** := E_1 **to** E_2 **step** E_3 **do** S_1

for 循环语句的翻译模式如下所示：

S → **for id** := E_1 **to** E_2 **step** E_3 **do** M S_1 {*backpatch*(S_1.*nextlist*, *M.again,*);
gencode('goto', -, -, *M.again*); *S.nextlist* := *M.again*;}

M→ε {*M.addr* := *entry*(**id**); *gencode*(':=', E_1.*addr*, -, *M.addr*); T_1:=*newtemp*;
gencode(':=', E_2.*addr*, -, T_1); T_2:=*newtemp*; *gencode*(':=', E_3.*addr*, -, T_2); *q*:=*nextquad*;

gencode(‘goto’, -, -, *q*+2); *M.again*:=*q*+1; *gencode*(‘+’, *M.addr*, T_2, *M.addr*);
M.nextlist:=*nextquad*; *gencode*(‘if’ *M.addr* ‘>’T_1‘goto –’);}

5.4 repeat 语句的翻译

repeat 语句的文法如下所示：

S→ **repeat** S_1 **until** *B*

Repeat 语句的翻译模式如下所示：

S→**repeat** *M* S_1**until** *N B*{*backpatch*(*B.falselist*,*M.quad*);
S.nextlist:=*B.truelist*}
M→ε{*M.quad* := *nextquad*}
N→ε{*backpatch*(S_1.*nextlist*, *nextquad*)}

6. switch 语句的语法制导翻译

switch 语句的文法为：

S → **switch** (*E*) *Clist*
Clist→ **case** *V* : *S Clist* | **default** : *S*

switch 语句的翻译模式如下所示：

⑴ *S*→**switch** (*E*){*i*:=0; S_i.*nextlist*:=0; push S_i.*nextlist*; push *E.addr*; push *i*; *q*:=0; push *q*}
Clist{pop *q*;pop *i*;pop *E.addr*;pop S_i.*nextlist*;*S.nextlist*:=*merge*(S_i.*nextlist*, *q*); push *S.nextlist*}

⑵ *Clist*→**case** *V* :{pop *q*; pop *i*; *i*:=*i*+1; pop *E.addr*;
if *nextquad* ≠0 **then** *backpatch*(*q*, *nextquad*);
q:=*nextquad*;
gencode(‘if’ *E.addr* ‘≠’ V_i ‘goto’ L_i);
push *E.addr*; push *i*;
push *q*}*S*{pop *q*; pop *i*; pop *E.addr*; pop S_{i-1}.*nextlist*;
p:=*nextquad*;
gencode(‘goto -’); *gencode*(L_i‘:’);
S_i.*nextlist*:=*merge*(S_i.*nextlist*, *p*);
S_i.*nextlist*:=*merge*(S_i.*nextlist*, S_{i-1}.*nextlist*);
push S_i.*nextlist*; push *E.addr*; push *i*; push *q*}*Clist*

⑶ *Clist*→**default** :{pop *q*; pop *i*; *i*:=*i*+1; pop *E.addr*;
if *nextquad* ≠0 **then** *backpatch*(*q*, *nextquad*);
q:=*nextquad*;
gencode(‘if’ *E.addr* ‘≠’ V_i ‘goto’ V_{i+1});
push *E.addr*; push *i*;
push *q*}*S*{pop *q*; pop *i*; pop *E.addr*; pop S_{i-1}.*nextlist*;
p:=*nextquad*;
gencode(‘goto -’); *gencode*(L_i‘:’);
S_i.*nextlist*:=*merge*(S_i.*nextlist*, *p*);
S_i.*nextlist*:=*merge*(S_i.*nextlist*, S_{i-1}.*nextlist*);
push S_i.*nextlist*; push *E.addr*; push *i*; push *q*}

7. 过程调用和返回语句的翻译

过程调用和返回语句的文法如下所示：

$S \rightarrow$ **call id**(*Elist*)

Elist →*Elist*, *E* | *E*

$S \rightarrow$ **return** *E*

过程调用语句的翻译模式如下所示：

⑴ *S*→**call id** (*Elist*) {*n* :=0;

repeat

n:=*n*+1;

从 *queue* 的队首取出一个实参地址 *p*;

gencode('param', -, -, *p*);

until *queue* 为空;

gencode('call', **id**.*addr*, *n*, -)}

⑵ *Elist*→*Elist*, *E*{将 *E.addr* 添加到 *queue* 的队尾}

⑶ *Elist*→*E*{初始化 *queue*，然后将 *E.addr* 加入到 *queue* 的队尾。}

过程返回语句的翻译模式为：

$S \rightarrow$ **return** *E*{**if** 需要返回结果 **then** *gencode*(':=', *E.addr*, -, *F*);

gencode('ret', -, -, -)}

其中，*F* 是存放结果的指定单元，四元式('ret', -, -, -)执行如下操作：

⑴ 恢复主调程序的寄存器内容；

⑵ 释放过程运行时所占用的数据区；

⑶ 按返回地址返回到主调程序。

8. 输入输出语句的翻译

带 I/O 参数的程序语句和输入输出语句的文法如下所示：

$P \rightarrow$ **prog id** (**input**, **output**) *D* ; *S*

$S \rightarrow$ **read** (*List*)

| **readln**(*List*)

$S \rightarrow$ **write** (*Elist*)

| **writeln**(*Elist*)

带 I/O 参数的程序语句和输入输出语句的翻译方案如下所示：

P→ **prog id** (*Parlist*) *M D* ; *S*

Parlist→ **input**(ε | , **output**)

S→ (**read | readln)** (*N List*); {*n*:=0;

repeat

move(Queue, i_n);

gencode('par', 'in', -, -);

n:=*n*+1;

until Queue 为空;

gencode('call', 'SYSIN', *n*-1, -);}

List→**id**, *L* (ε|*List*)

S→ (**write**| **writeln)** (*Elist*); { *n*:=0;

repeat

move(Queue, i_n);

gencode('par', 'out', -, -);

n:=*n*+1;

until Queue 为空;
gencode(‘call’, ‘SYSOUT’, *n*, ‘w’)}
/**n* 为输出参数个数，w 是输出操作类型*/

EList→*E*, *K* (*ε*|*EList*)
M→*ε* {*gencode*(‘prog’, **id**, *y*, -)} /**y* 的值表示 **input**，**output** 或两者皆有*/
N→*ε* {设置一个语义队列 Queue}
L→*ε* {*T*:=*entry*(**id**); *add*(Queue, *T*)}
K→*ε* {*T*:= *E.addr*; *add*(Queue, *T*)}