算子式语言到过程语言的变换语义*

阎志欣

(北京航空航天大学计算机科学与工程系 北京 100083)

黄冬泉

(南京大学计算机软件研究所 南京 210093)

摘要 算子式程序设计语言是一种有坚实理论基础、高效的、实际有用的、高级的新型程序设计语言.如何编绎该类语言是一个应该研究的重要问题.过程式语言到机器代码的编绎问题已被解决,因此编绎的关键是由算子程序向过程的转换.本文用模式匹配法,给出了算子式语言到while程序的变换语义.给出了基本变换规则集,证明了该变换规则集的完备性和一致性,为该类语言到过程语言的翻译提供了理论和技术基础.

关键词 算子式语言,变换语义,变换规则,完备性,一致性.

程序设计语言的目的是对待求解问题的算法实现人—机通讯.用传统的过程式语言虽然可描述迭代,使得程序有高的时空效率,但程序缺乏数学特性,从而导致了不易理解、分析和证明等许多问题.[1]陈述式语言把算法的逻辑和控制分离,程序仅描述算法的输入输出关系,使得程序有良好的数学特性,易于理解、分析和证明.但由于其程序仅包含输入输出变量,难以描述迭代,从而难以构造出高效的执行系统.[2]近20年来各种陈述式语言被广泛研究[3~5],但由于程序的控制结构仍然是递归,因而时空效率问题一直是要研究的重要问题.

基于迭代的算子式程序设计语言是一种新型的、有良好数学基础、既可描述递归又可描述迭代的高效、实际有用的程序设计语言.文献[6]给出了该类语言的语法,并用解释实现给出了操作语义.该类语言仅用其迭代算子就可计算任何部分递归函数已被证明.[7]这就显示了,任何图灵可计算的对象,都可用有坚实数学基础的迭代算子描述它,并且可以构造高效的执行系统计算它.执行系统的解释实现对定义语言的操作语义是重要的,对计算机辅助开发程序是有用的,但其程序执行效率低.因此如何编绎该类语言也是应该研究的一个重要问题.算子式程序不是一个强制性指令序列,而是一种描述性程序.由于基于while程序的过程式语言的编绎已经解决,因此算子式程序编绎的关键是从描述到过程的转换.

把语言 A_1 的成分变换为语言 A_2 的成分可用2种方法:(1)算法方法,用一个过程把 A_1

* 本文研究得到南京大学计算机软件研究所基金和航空科学技术基金资助.作者阎志欣,1936年生,教授,主要研究领域为计算、推理理论和语言.黄冬泉,1964年生,讲师,北京航空航天大学硕士生,主要研究领域为语言.

本文通讯联系人:阎志欣,北京100083,北京航空航天大学计算机科学与工程系

本文1995—09—25收到修改稿

的成分作为输入参数,经一系列加工得到 A_2 的成分;(2)模式匹配方法,也以 A_1 的成分作为输入参数,但加工很简单:把输入参数和预制的模板匹配,若匹配成功,则取与此模板相对应的 A_2 的成分作为输出,并在输出时作相应的参数代真. 模式匹配法又称变换方法. 这种方法可用于由 n 个语言构成的广谱语言 $A_1,A_2,\cdots,A_n$. 其中,对所有的 i,$1\leqslant i<n$,A_{i+1}的语义决定了 A_i 的语义,并且对每个 i 有一组从 A_i 到 A_{i+1}的变换规则. 语言 A_1 的实现由各组变换规则决定. 因而这种实现又称变换语义,是操作语义的一个变种.[8]本文给出了算子式语言到 while 程序的变换语义. 文中给出了基本变换规则集,证明了该变换规则集的完备性和一致性. 从而为该类语言到过程语言的编绎提供了理论和技术基础. 我们已用变换方法实现了多类型算子式语言到 PASCAL 的变换系统. 限于篇幅,本文仅涉及算子式程序的算法描述部分,即表达式,到 PASCAL 语句序列的变换.

1 语　法

下面,我们撇开一些细节,给出了算子式程序设计语言的语法. 让 m 是任何自然数,对所有的 j,$1\leqslant j\leqslant m$,笛卡尔积 $D^m=D_1\times\cdots\times D_m$ 表示 m 元函数的定义域,D_j 被称为子域,D 表示函数的值域. 后面称 D_j 和 D 为数据域,$d_j\in D_j$ 和 $d\in D$ 是任何对象,被称为常量或数据项.

定义 1. 项被归纳地定义如下:

(1)一个变量是一个项.

(2)一个常量是一个项.

(3)如果 f 是一个 n 元初始或定义函数符号,并且 $t_1,\cdots,t_n$ 是项,则 $f(t_1,\cdots,t_n)$是一个项.

若 f 是初始函数符号,则 $f(t_1,\cdots,t_n)$称初始函数项,若 f 是定义函数符号,则 $f(t_1,\cdots,t_n)$称定义函数项.

定义 2. 让 t_1,t_2 是项,r 是 2 元初始关系符号,则$(t_1\ r\ t_2)$是一个初始原子. 初始关系符号如:$=$,$>$,$<$,$\geqslant$,$\leqslant$等.

定义 3. 谓词被归纳地定义如下:

(1)一个初始原子是一个谓词.

(2)如果 A 是一个谓词,则(not A)是一个谓词 .

(3)如果 A,B 是谓词,则(A)and(B)是一个谓词.

其中 not 和 and 是逻辑联词,and 表示"与",not 表示"非". 显然,一个谓词是一个布尔表达式.

定义 4. 让 $h:D^m\to D$ 是 m 元初始或定义函数,又称主迭代函数,$(t_1,\cdots,t_m)$是项表;$w:N^2\to\{k\mid 1\leqslant k\leqslant m\}$是自然数集上的 2 元初始函数,又称位置函数;$P$ 是一个谓词;e,n 是自然数;则一个迭代表达式是一个形式如下的表达式:

$$P$$
$$\backslash w(i,n)\backslash h(t_1,\cdots,t_m)$$
$$i=e$$

其中 i 是 P 和 w 的一个变量，又称迭代变量，并且允许出现在 $t_1,\cdots,t_m$ 中，$i=e$ 指示 i 的初值是 e，每步计算后 i 的值自动加 1；若以 I^d 表示该迭代表达式执行了从 $i=e$ 至 $i=d$ 若干次迭代后的阶段性计算结果，则该表达式应满足：

(1)若在 $i=e$ 下有 $w(e,n)=a$，则迭代表达式有初始结果： $I^e=t_a$

(2)若在 $i=d$ 下 P 为假，并且 $w(d,n)=a$，$w(d+1,n)=b$，则迭代表达式在 $i=d+1$ 时有结果： $$I^{d+1}=h(t_1,\cdots,t_{a-1},I^d,t_{a+1},\cdots,t_m)[i\leftarrow d+1]$$

并且 I^{d+1} 被存储在项表 $(t_1,\cdots,t_m)$ 的位置 b 处.

(3)如果存在一个 c 使得 P 在 $i=c$ 下为真，则对所有 j，$j\geqslant c$，迭代表达式有终止结果：

$$I^j=I^c$$

(4)若不存在任何 c 使得 P 在 $i=c$ 下为真，则迭代表达式无定义，即计算不终止.

其中，$X[y\leftarrow z]$ 表示若 y 在 X 中出现则 X 中的 y 用 z 置换所得到的结果，若 y 不在 X 中出现，则置换为空，X 不变. 迭代表达式中的所有置换是同时的.

位置函数 w 是使得 $w(x,y)=z$，并且 $1\leqslant z\leqslant m$ 成立的任何初始函数，用于指示迭代表达式中间计算结果的存放位置. 特别有用的是自然数集上的常函数和 x 除以 y 的余数加 1 函数. 上述 4 项条件由算子式语言的操作语义体现.[6]

定义 5. 一个表达式被归纳地定义如下：

(1)一个项是一个表达式.

(2)一个迭代表达式是一个表达式.

(3)若 C 是谓词，E_1 是项或迭代表达式，E_2 是一个表达式，则 $C\rightarrow E_1;E_2$ 是一个表达式，又称条件表达式. 其中符号“→”表示蕴含.

这里，$C\rightarrow E_1;E_2$ 等价于 if C then E_1 else E_2. 显然，一个表达式是由项、迭代表达式和条件函数 if then else 构造的复合结构. 让 $E=C\rightarrow E_1;E_2$，其意义是：若 C 是真，则 $E=E_1$，否则 $E=E_2$.

定义 6. 一个函数定义其形式如： $f(x_1,\cdots,x_n)=exp$

其中，f 是定义函数符号，exp 是表达式，对所有 j，$1\leqslant j\leqslant n$，x_j 是函数 f 的变量. 让 i 表示 exp 中的迭代变量，$Var(f)$，$Var(exp)$ 分别表示函数 f 和 exp 中的变量集合，则一个函数定义应满足条件： $$Var(f)\cup\{i\}=Var(exp)$$

显然，$x_1,\cdots,x_n$ 在函数定义中是约束变量，i 可看作自由变量. 后面称仅含约束变量的项为约束项.

定义 7. 一个算子式程序是一个函数定义的集合，其中的定义函数符号是各不相同的.

一个函数定义的指称是一个函数. 一个算子式程序的指称是一个函数集. 上面给出的算子式程序设计语言的语法是该类语言的一个模式. 如果给出常量符号集，初始函数符号集和构成谓词的初始关系符号集，就给出了一个特定的语言.

2 while 程序设计语言

在变换语义下，我们用一个小的然而其计算能力与图灵机等价的 while 程序设计语言的语义确定 OPLAN 的语义. 下面给出 while 程序设计语言的语法. 让 C 是布尔表达式，t 是

项,S 是语句.它们可带下标.则下面是 while 程序的语句:

(1)赋值语句　$x:=t$

(2)复合语句　begin $S_1;S_2$ end

(3)条件语句　if C then S_1 else S_2

　　if C then S

(4)循环语句　while C do S

这里的项 t 和布尔表达式的定义与上面给出的算子式语言相同.显然,上面给出的 while 程序设计语言是 PASCAL 的一个子集.

3 变换规则

用语言 A_2 的语义确定语言 A_1 的语义,关键是给出从 A_1 到 A_2 的变换规则集.由于在变换中需要引入非约束变量并且置换约束项,下面首先给出有关定义.

定义 8. 让 δ 是一个形式如 $\{mrs\_1/t_1,\cdots,mrs\_n/t_n\}$ 的有限集合,其中对每个 j,$1\leqslant j\leqslant n$,$mrs\_1,\cdots,mrs\_n$ 是各不相同的非约束变量,每个 t_j 是一个不同于 mrs_j 的约束项,则称 δ 是一个约束项置换.

定义 9. 让 $h(t_1,\cdots,t_m)$ 是一个项,若其中存在某些 j,$1\leqslant j\leqslant m$,t_j 是约束项,$\delta=\{mrs\_j/t_j\mid 1\leqslant j\leqslant m$ 并且 t_j 是约束项$\}$ 是一个约束项置换,则称 $h(t_1,\cdots,t_m)\delta$ 为 $h(t_1,\cdots,t_m)$ 的一个约束项置换实例.

例:让 $h(g_1(x_1,\cdots,x_n),g_2(x_1,\cdots,x_n),g_3(x_1,\cdots,x_n,i))$ 是一个项,若在一个函数定义中 $x_1,\cdots,x_n$ 是约束变量,那么 $g_1(x_1,\cdots,x_n)$ 和 $g_2(x_1,\cdots,x_n)$ 是约束项,则

$$\delta=\{mrs\_1/g_1(x_1,\cdots,x_n),mrs\_2/g_2(x_1,\cdots,x_n)\}$$

是一个约束项置换,并且

$$h(g_1(x_1,\cdots,x_n),g_2(x_1,\cdots,x_n),g_3(x_1,\cdots,x_n,i))\delta=h(mrs\_1,mrs\_2,g_3(x_1,\cdots,x_n,i))$$

是一个约束项置换实例.

若以 $\frac{M_1}{M_2}(B)$ 表示一个变换规则,其中 M_1 是输入模式,M_2 是输出模式,B 表示变换规则的应用条件,则下面是算子式程序到 while 程序的变换规则集:

$T1$——项表达式变换规则

$$\frac{f(x_1,\cdots,x_n)=X(A)}{f(x_1,\cdots,x_n)=X(A)[A\leftarrow f:=t]}$$　(若成分 A 的模式是项 t)

$T2$——单值迭代变换规则

$$\frac{f(x_1,\cdots,x_n)=X(A)}{f(x_1,\cdots,x_n)=X(A)[A\leftarrow WS1(m,k)]}$$　(若成分 A 的模式是 $wexp1(m,k)$)

P

其中模式 $wexp1(m,k)\equiv\backslash k\backslash h(t_1,\cdots,t_m)$

$i:=e$

模式 $WS1(m,k)\equiv$begin

　　$i_1:=e_1$;

　　$mrs_1:=t_k$;

　　while $not(P)$ do

```
        begin
          i:=i+1;
          mrs:=h(t1,…,tk-1,mrs,tk+1,…,tm);
        end;
      f:=mrs;
    end;
```

这里 k 是自然数常量，$1\leqslant k\leqslant m$，变量 mrs 用于存储 $WS1$ 的中间计算结果，变量 f 用于存放最后计算结果，i 是必在 P 中出现的迭代控制变量.

$T3$——串值迭代变换规则

$$\frac{f(x_1,\cdots,x_n)=X(A)}{f(x_1,\cdots,x_n)=X(A)[A\leftarrow WS2(m,k)]}_{P}\qquad(\text{若成分 } A \text{ 的模式是 } wexp2(m,k))$$

其中模式 $wexp2(m,k)\equiv\backslash w(i,k)\backslash h(t_1,\cdots,t_m)_{i=e}$

模式 $WS2(m,k)\equiv$

```
begin i:=e;
  mrs_1:=t1;
  mrs_2:=t2;
  ……
  mrs_m:=tm;
  while not(P) do
    begin
      i:=i+1;
      if w(i,k)=1 then mrs_1:=h(t1,…,tm)δ;
      if w(i,k)=2 then mrs_2:=h(t1,…,tm)δ;
      ……
      if w(i,k)=m then mrs_m:=h(t1,…,tm)δ;
      end;
    if w(i,k)=1 then f:=mrs_1;
    if w(i,k)=2 then f:=mrs_2;
    ……
    if w(i,k)=m then f:=mrs_m;
  end;
```

这里 k 是自然数常量，$w(i,k)\in\{1,2,\cdots,k\mid k\leqslant m\}$，变量 $mrs\_1,\cdots,mrs\_m$ 用于存储 $WS2$ 的中间计算结果，变量 f 用于存放最后计算结果，i 是必在 P 中出现的迭代控制变量，$h(t_1,\cdots,t_m)\delta$ 是 $h(t_1,\cdots,t_m)$ 的一个约束项置换实例.

$T4$——条件表达式变换规则

$$\frac{f(x_1,\cdots,x_n)=X(A)}{f(x_1,\cdots,x_n)=X(A)[A\leftarrow\text{if } C \text{ then } Y \text{ else } Z]}\qquad(\text{若成分 } A \text{ 的模式是 } C\rightarrow Y;Z)$$

4 变 换

算子式程序到 while 程序的变换，是以 while 程序为最终对象，重复地使用变换规则，由旧的变换对象推出新的变换对象. 更精确地我们有下面定义.

定义10. 让 P_1 是一个算子式程序，P_n 是一个 while 程序，一个变换是一个由变换对象组成的序列 $P_1,\cdots,P_n$ 使得对所有的 i，$1\leqslant i<n$，P_{i+1} 是由变换对象 P_i 通过一个变换规则产生的新变换对象.

让 oplan 表示算子式语言. 显然,对所有的 i,$1<i<n$,$P_i \in \text{oplan} \cup \text{while}$,即 P_i 是由算子式语言和 while 程序2种语言成分构成的变换对象.

让 P_i 是一个变换对象,$\frac{M_1}{M_2}(B)$是一个变换规则,若选定了一个变换规则,应用它由 P_i 得到 P_{i+1},此过程由3步组成:

(1)分析:在其输入模式 M_1 中的成分和 P_i 的相应成分之间建立起一一对应的关系.

(2)测试:测试 P_i 的成分 A 是否满足条件 B,若满足 B 则转步(3),否则另选变换规则,并转步(1).

(3)代真:把输出模式 M_2 中的相应成分代之以 P_i 的相应成分,并且必要时作约束项置换,则得 P_{i+1}.

下面以例说明算子程序到 while 程序的变换过程. 在各例中左边标示了一步变换所用的变规则,以下划线"——"标示在一步变换时,成分 X 中满足条件 B 的成分.

例1:计算 ackermann 函数的递归程序

算子式程序是:

$$ack(x,y)=(x=0)\rightarrow y+1;(y=0)\rightarrow ack(x-1,1);ack(x-1,ack(x,y-1));$$

应用上述变换规则可将其变换为等价的 while 程序(也是递归的). 变换过程如图1.

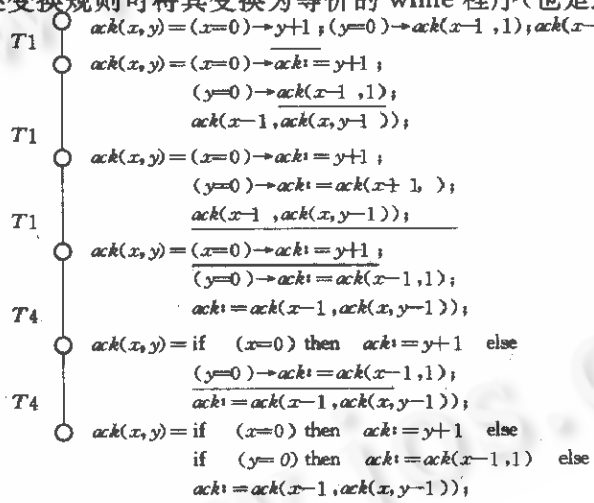


图1

例2:递归+迭代计算 ackermann's 函数

ackermann's 函数是一个非原始递归、但完全可计算的双递归函数. 它的时空消耗因输入值的增长速度极高. 提高多重递归时空效率的有效途径是用迭代减少递归重数. 这要求程序设计语言具有同时表达递归和迭代的能力. 这对传统的陈述式语言是非常困难的. 下面是单递归+迭代的算子式程序.

$$\begin{matrix}(i=y)\\ ack(x,y)=(x=0)\rightarrow y+1;\backslash 2\backslash ack(x-1,ack(x-1,1));\\ i=0\end{matrix}$$

应用上述变换规则可将其变换为等价的 *while* 程序(也是递归+迭代). 变换过程如图2.

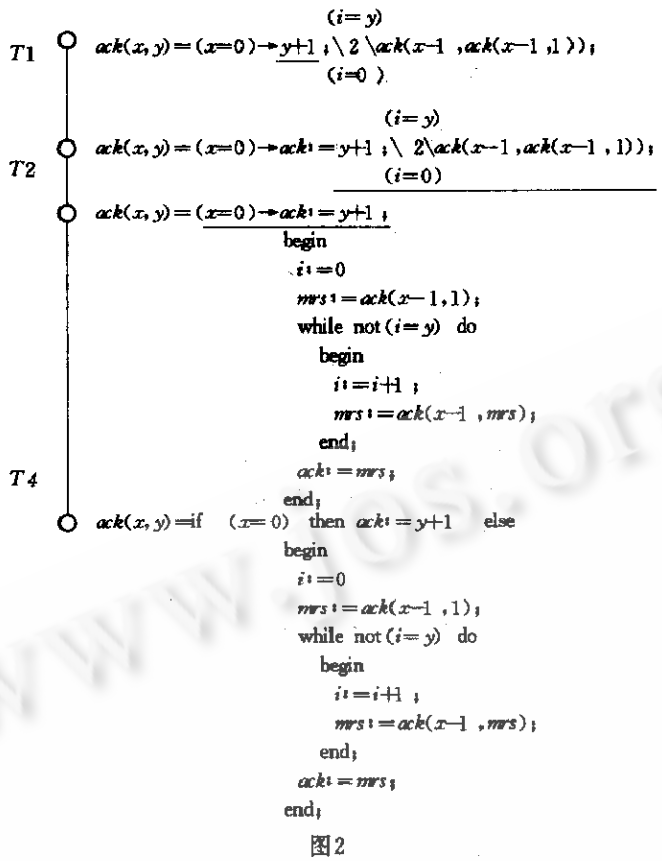


图2

例3：迭代计算 fibonacci 函数

该函数是一个串值依赖函数，其值在 $x=0,1,2,3,4,5,6,\cdots$ 时，分别是0，1，2，3，5，8，…. 用串值递归计算的时间是 x 的指数函数，需要的空间是 x 的线性函数. 提高时空效率的有效途径是用串值迭代. 传统的纯陈述式语言只能用串值递归计算. 让 s 是后继函数，$mod(i,2)$ 是 i 除以2的余数，则下面是算子式串值迭代程序：

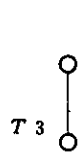
$$fib(x)=\underset{i=0}{\overset{(i=x)}{\backslash}}s(mod(i,2))\backslash add(0,1);$$

这里，$w(i,2)=s(mod(i,2))$. 应用上述变换规则可将其变换为等价的 *while* 程序（也是迭代）. 变换过程如图3.

5 完备性

由语言 A_1 到语言 A_2 的变换规则集可大可小，一个基本变换规则集是一个最小变换规则集. 基本变换规则集不是唯一的. 但它应满足类似于对公理集那样的要求，如完备性和一致性.[8]

定义11. 若由变换规则集 TS 可以导出变换规则 T，则 T 称为 TS 的一个导出规则. 如果 TS 是一个变换规则集，其中任何一个规则都不能由其它规则导出，则称 TS 是一个基本变换规则集.



$T3$

$(i=x)$

$fib(x)=\s(mod(i,2))\add(0,1);$

$i=0$

```
fib(x)= begin
          i:=0;
          mrs_1 :=0;
          mrs_2 :=1;
          while not (i=x) do
            begin
             i:=i+1
              if s(mod(i,2))=1   then mrs_1 :=add(mrs_1 ,mrs _2 )
              if s(mod(i,2))=2   then mrs_2 :=add(mrs_1 ,mrs _2 )
            end;
          if  s(mod(i,2))=1  then  fib:=mrs_1;
          if  s(mod(i,2))=2  then  fib:=mrs_2;
        end;
```

图3

上述变换规则集 $TS=\{T1,T2,T3,T4\}$ 中,任何一个规则都不能由其它规则导出,因而是一个基本规则集.

下面以 oplan 表示算子式语言,while 表示 while 程序语言.

定义12. 给定一个基本变换规则集 TS,又任给一个表达式 $P_1\in$oplan,若存在一个变换规则序列$\langle T_1,\cdots,T_{n-1}\rangle$,$T_i\in TS$,使得有变换 $P_1,P_2,\cdots,P_n$,其中对所有 i,$1\leqslant i<n$,规则 T_i 使得 $P_{i+1}=T_i(P_i)$,并且 $P_n\in$while,则称 TS 是完备的.

定理1. 算子式语言 oplan 到 while 程序语言的基本变换规则集 $TS=\{T1,T2,T3,T4\}$ 是完备的.

证明:归纳证明于算子式语言表达式的结构,一般结构形式为:

$$P_1=C_1\rightarrow E_1;C_2\rightarrow E_2;\cdots;C_m\rightarrow E_m;E_{m+1}\quad m\geqslant 0$$

其中对所有的 i,$1\leqslant i\leqslant m+1$,$E_i$ 是项 t,或是迭代表达式 $wexp1(m,k)$,或是迭代表达式 $wexp2(m,k)$.

当 $m=0$时,若 $P_1=t$,则有规则 $T1\in\{T1\}\subseteq TS$ 使得 $T1(P_1)=f:=t\in$while.

若 $P_1=wexp1(m,k)$,则有规则 $T2\in\{T2\}\subseteq TS$ 使得 $T2(P_1)=WS1(m,k)\in$while.

若 $P_1=wexp2(m,k)$,则有规则 $T3\in\{T3\}\subseteq TS$ 使得 $T3(P_1)=WS2(m,k)\in$while.

当 $m>0$时,对所有 i,$1\leqslant i\leqslant m+1$,有规则 $Ti\in\{T1,T2,T3\}\subseteq TS$ 使得 $P_{i+1}=Ti(P_i)\in$oplan$\cup$while,从而可完成对 E_i 到 $S_i\in$while 的变换,因而有:

$$P_{m+2}=C_1\rightarrow S_1;C_2\rightarrow S_2;\cdots;C_m\rightarrow S_m;S_{m+1}\in\text{oplan}\cup\text{while}.$$

并且重复应用规则 $T4\in\{T4\}TS$,有子变换:

$$P_{m+3}=T4(P_{m+2}),P_{m+4}=T4(P_{m+3}),\cdots,P_n=T4(P_{n-1}),\text{使得}$$

$$P_n=\text{if } C_1 \text{ then } S_1 \text{ else if } C_2 \text{ then } S_2 \text{ else}\cdots\cdots\text{if } C_m \text{ then } S_m \text{ else } S_{m+1}\in\text{while}.$$

由上可知,对任何 m,即对算子式语言的任何表达式,都存在一个变换规则序列$\langle T_1,\cdots,T_{n-1}\rangle$,$T_i\in TS$,使得有变换 $P_1,P_2,\cdots,P_n$,其中对所有 i,$1\leqslant i<n$,$P_{i+1}=T_i(P_i)$,并且 $P_n\in$while,所以变换规则集 TS 是完备的.

由于在对任意给定的算子表达式 P_1的变换 $P_1,P_2,\cdots,P_n$ 中,对所有的 i,$1\leqslant i<n$,从 P_i

到 P_{i+1} 可用的变换规则不是唯一的，这就存在一致性问题.

定义13. 让 $P_1 \in oplan, P_n \in while$. 在从 P_1 到 P_n 的变换中，对所有的 i，$1 \leqslant i < n$，若有规则 $Ta, Tb \in TS$ 皆可应用于 P_i 时，必有规则 $Tc, Td \in TS$，使得 $Tc(Ta(P_i)) = Td(Tb(P_i))$，则称 TS 是一致的.

定理2. 算子式语言 *oplan* 到 *while* 程序语言的基本变换规则集 $TS = \{T1, T2, T3, T4\}$ 是一致的.

证明：归纳证明于算子式语言表达式的结构，一般结构形式为：

$$P_1 = C_1 \rightarrow E_1; C_2 \rightarrow E_2; \cdots; C_m \rightarrow E_m; E_{m+1} \qquad m \geqslant 0$$

其中对所有的 i，$1 \leqslant i \leqslant m+1$，$E_i$ 是项 t，或是迭代表达式 $wexp1(m,k)$，或是迭代表达式 $wexp2(m,k)$.

当 $m=0$ 时，若 $P_1 = t$，仅有规则 $T1 \in TS$ 可用.

若 $P_1 = wexp1(m,k)$，仅有规则 $T2 \in TS$ 可用.

若 $P_1 = wexp2(m,k)$，仅有规则 $T3 \in TS$ 可用.

因而当 $m=0$ 时，TS 是一致的.

当 $m>0$ 时，

对所有 i，$1 \leqslant i < n$，让 $P_i = P(C_j \rightarrow E_j; Z)$，$1 \leqslant j \leqslant m$，有规则 $Ta \in \{T1, T2, T3\} \subseteq TS$ 和 $T4 \in TS$ 都可应用于 P_i：

(1)若用 Ta，则 $P_{i+1} = Ta(P(C_j \rightarrow E_j; Z)) = P(C_j \rightarrow Ta(E_j); Z) = P(C_j \rightarrow S_j; Z)$，我们有 $T4 \in TS$ 使得 $P_{i+2} = T4(P_{i+1}) = T4(P(C_j \rightarrow S_j; Z)) = P(\text{if } C_j \text{ then } S_j \text{ else } Z)$

(2)若用 $T4$，则 $P_{i+1} = T4(P(C_j \rightarrow E_j; Z)) = P(T4(C_j \rightarrow E_j; Z)) = P(\text{if } C_j \text{ then } E_j \text{ else } Z)$，

我们有 $Ta \in \{T1, T2, T3\} \subseteq TS$，使得 $P_{i+2} = Ta(P_{i+1}) = Ta(P(\text{if } C_j \text{ then } E_j \text{ else } Z)) = P(\text{if } C_j \text{ then } T_a(E_j) \text{ else } Z) = P(\text{if } C_j \text{ then } S_j \text{ else } Z)$

因而对 $m>0$，有 $T4(Ta(P_i)) = Ta(T4(P_i))$，$TS$ 是一致的.

参考文献

1 Backus John. Can programming be liberated from the Von Neumann style? A function style and its algebra of programs. Communications of the ACM. August 1978, **21(8)**: 613～641.

2 Morris James H. Real programming in function languages. In: Darlington ed. Function Programming and Its Applications, 1982.

3 Guo Yike, Lok H C R. A classification scheme for declarative programming languages Syntax, semantics, and operational models. GMD—Studien, N8. 182, August 1990.

4 Darlington John, Guo Yike, Pull Helen. A design space for itegrating declarative languages. In: Darlington John, Dietrich Roland eds. Declarative Programming, 1992.

5 Darlington John, Guo Yike, Pull Helen. Introducing constraint functional logic programming. In: Darlington John, Dietrich Roland eds. Declarative Programming, 1992.

6 阎志欣. 带迭代算子的函数式程序设计. 软件学报, 1996, 7(增刊): 239～248.

7 阎志欣, 黄盛桦. 迭代函数及其可计算性. 软件学报, 1996, 7(增刊): 232～238.

8 陆汝钤. 计算机语言的形式语义. 北京: 科学出版社, 1992.

TRANSFORMATION SEMANTICS OF OPERATOR LANGUAGE TO PROCEDURE LANGUAGE

Yan Zhixin

(*Department of Computer Science and Technology Beijing University of Aeronautics and Astronautics Beijing* 100083)

Huang Dongquan

(*Institute of Computer Software Nanjing University Nanjing* 210093)

Abstract Operator programming language is an efficient, useful and practical, high-level, new programming language with sound theoretical foundation. How to compile this language is an important problem which should be made researches on. The translation of programs from procedure language into machine language has been solved, hence the linchpin upon which the translation depends is the transformation of operator programs to procedures. In this paper, the transformation semantics of operator language to while programs was given with model match method. A basic set of transformation rules was given and it was proved to be complete and consistent, so that the theoretic and technologic foundation to translate operator language to procedure language was provided.

Key words Operator language, transformation semantics, transformation rule, completeness, consistency.