带迭代算子的函数式程序设计*

阎志欣

（北京航空航天大学计算机科学与工程系　北京　100083）

摘要　本文提出了程序设计语言的一种新的计算模型，带迭代算子的函数式模型. 文中给出了基于该计算模型的程序设计语言的语法、计算规则集及确定性证明. 该类语言以表达式的复合描述顺序，以迭代表达式描述循环，以条件表达式描述分支，使得程序以数学语义为基础，易于理解、证明及构造高效的执行系统. 相信该类语言是一种有坚实理论基础、高效的、实际有用的，高级确定性程序设计语言.

关键词　迭代算子，函数式语言，归约，计算，确定性.

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. 用传统的指令式语言，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. 1978 年 John Backus 在他的图灵奖演讲中对该类语言存在的严重问题有全面的分析.[1]

近 20 年来陈述式语言被广泛研究. John Darlington，Yi—keguo，H. C. R. lock 和 Helen Pull 在文献[2～4]中对各种陈述式语言的语法、语义、操作模型等有详细讨论. 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，缺乏对机器时空资源的有效控制手段，使得难以构造出高效的执行系统.[5]

20 年来，陈述式程序设计语言的效率问题一直是研究的重要问题. 有 3 条途径：第 1 条，并行计算提高执行速度，这仅是以硬件资源换取速度，并没有解决效率问题本身. 第 2 条，在程序中引入强制性指令，如 Prolog 中的 cut 和 fail 及 Lisp 中的 Prog 函数. 这条途径可控制程序执行顺序和进行迭代计算，然而破坏了程序的数学特性. 显然这与陈述式语言的宗旨相违背. 第 3 条，尾递归优化，存在着可迭代计算，但非尾递归的函数不能优化.

为使程序设计语言既有良好的数学特性，又易于构造高效的执行系统，使程序有高的执行效率，我们在文献[6]中提出了状态逻辑型程序设计语言的计算模型. 本文提出和构造了带迭代算子的函数式计算模型. 因迭代算子可由已知函数生成新函数，因此称基于该模型的语言为带迭代算子的函数式程序设计语言. 由于该类语言以表达式的复合描述顺序，以迭代表达式描述循环，以条件表达式描述分支，使得程序不仅有良好的数学特性，易于理解、分析

* 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，推理理论和程序设计语言.

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 1995－06－19 收到修改稿

和证明,而且易于构造高效的执行系统,使程序有高的执行效率.我们相信该类语言是一种有坚实理论基础、高效的、实际有用的、高级确定性程序设计语言.

任何图灵可计算的函数都是带迭代算子的函数式计算模型可计算的.限于篇幅,本文仅涉及计算模型及它作为程序设计语言的主要方面,给出了该模型的语法、基于归约的操作语义及操作模型的确定性证明.该类语言的可计算性、指称语义及其到指令式语言的变换语义有另文发表.

1 语 法

这里,我们撇开程序设计语言的语法细节,给出该语言的抽象语法.

让 m 是任何自然数,对所有的 j,$1\leqslant j\leqslant m$,笛卡耳积 $D^m=D_1\times\cdots\times D_m$ 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,又称 0 元函数或常函数.

(3)如果 f 是一个 n 元初始或者定义函数符号,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 $(\neg A)$ 是一个谓词.

(3)如果 A,B 是谓词,则 $(A)\wedge(B)$ 是一个谓词.

其中,$\wedge$ 和 $\neg$ 是逻辑联词,$\wedge$ 表示"与",$\neg$ 表示"非".显然,一个谓词是一个布尔表达式.

定义 4. 让 $h:D^m\rightarrow D$ 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;$w:N^2\rightarrow\{k\mid 1\leqslant k\leqslant m\}$ 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:

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

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,$i=e$ 表示 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.

定义 5. 让 $g_1,\cdots,g_k$ 是出现在 $P,t_1,\cdots,t_m$ 中的定义函数符号,$x_1,\cdots,x_n$ 是出现在 $t_1,\cdots,t_m$ 和 P 中的变量,T 表示迭代算子,则 T 被定义为:

$$T:T(h,w,g_1,\cdots,g_k,e)(x_1,\cdots,x_n)=\backslash w(i,n)\backslash h(t_1,\cdots,t_m)\quad (i=e \text{ 为 } P)$$

迭代算子 T 把函数 $h,w,g_1,\cdots,g_k,e$ 变换为一个函数(即项),其中 e 是自然数集上的某个常函数. 若 $h,w,g_1,\cdots,g_k,e$ 是给定函数,则应用迭代算子定义的 n 元函数 f 被表示为:

$$f(x_1,\cdots,x_n)=\backslash w(i,n)\backslash h(t_1,\cdots,t_m)\quad (i=e \text{ 为 } P)$$

迭代算子可由已知函数产生新的函数,在这种意义下,我们说本文讨论的是带迭代算子的函数式程序设计.

定义 6. 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.

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 $E=E_2$.

不包含变量的项、谓词和表达式,被称为基础项、谓词和表达式.

定义 7. 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:　　$Var(f)\cup\{i\}=Var(exp)$

定义 8. 一个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. 一个程序的指称是一个函数集. 上面给出的带迭代算子的函数式程序设计语言的语法是该类语言的一个模型. 如果给出常量符号集、初始和定义

函数符号集及构成谓词的初始关系符号集，就给出了一个特定的语言.

2 例

(1)迭代计算阶乘函数

让 $w(i,n)=1$，是自然数集上的常函数，则

$$fact(x)=\backslash 1\backslash mul(1,i)_{i=0}^{(i=x)}$$

构成了计算阶乘函数的迭代程序，其中 *mul* 是乘函数，假定它是初始函数. 给定一个函数调用，则此程序将返回该调用的计算结果. 如我们有函数调用 *fcat*(3)，则程序返回计算结果 6.

(2)递归计算 ackermann 函数

ackermann 函数的递归程序是：

$$ack(x,y)=(x=0)\rightarrow s(y);(y=0)\rightarrow ack(pd(x),s(0));ack(pd(x),ack(x,pd(y)))$$

其中 s 是自然数集上的后继函数，即 $s(x)=x+1$，pd 是自然数集上的先驱函数，即 $pd(x)=x-1$，并且 $pd(0)=0$，假定它们是初始函数. 给定一个函数调用，如 $ack(1,1)$，则程序返回计算结果 3.

(3)递归＋迭代计算 ackermann 函数

ackermann 函数是一个非原始递归，但完全可计算的双递归函数. 它的时空消耗随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. 让 $w(i,n)=2$，则下面是单递归＋迭代的程序：

$$ack(x,y)=(x=0)\rightarrow s(y);\backslash 2\backslash ack(pd(x),ack(pd(x),s(0)));_{i=0}^{(i=y)}$$

同样，给定一个函数调用，如 $ack(1,1)$，则程序将返回计算结果 3.

(4)迭代计算 fibonacci 函数

该函数的值在 $x=0,1,2,3,4,5,6,\cdots\cdots$ 时，分别是 0,1,1,2,3,5,8,……. 其递归定义方程组是：$fib(0)=0$；

$fib(1)=1$；

$fib(x)=fib(x-1)+fib(x-2)$；

用递归计算的时间是 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迭代. 传统的纯陈述式语言只能用递归计算. 让 $w(i,n)=g(i,2)$，g 是 i 除以 2 的余数加 1 函数，$add(x,y)=x+y$，假定它们是初始函数，则下面是迭代程序：

$$fib(x)=\backslash g(i,2)\backslash add(0,1);_{i=0}^{(i=x)}$$

同样，给定一个函数调用，如 $fib(6)$，则程序将返回计算结果 8. 用该程序计算的时间是 x 的

线性函数,需要的空间是一个常量.

(5)迭代计算表的 *reverse* 和 *i_tail* 函数

让 x 是一个表,则 *reverse* 是表 x 的反序函数,*i_tail* 是表 x 从第 i 个元素开始的尾表. 让 *rev* 函数中 $w(i,n)=2$,*i_tail* 函数中 $w(i,n)=1$,则程序是:

$$rev(x)=\underset{i=0}{\backslash 2}\backslash cons(head(i\_tail(x,i)),nil);\qquad i\_tail(x,y)=\underset{j=1}{\backslash 1}\backslash tail(x);\quad (i=Ln(x))\quad (j=y)$$

其中函数 $head(x)$表示求表 x 的头元素,$tail(x)$表示求表 x 去掉头元素后的表,$cons(mem,x)$表示将元素 *mem* 联接到表 x 作为头元素形成的新表,*nil* 表示空表,$Ln(x)$表示求表 x 的长度,假定它们都是初始函数. 让 x 是一个整数表,则给定一个函数调用,如 *rev*([12 5 34]),程序将返回结果[34 5 12]. 同样,给定一个函数调用,如 *i_tail*([12 5 34],2),则程序将返回结果[5 34].

3 归　约

带迭代算子的函数式语言和传统的基于递归的函数型语言,计算对象都是表达式,其操作模型都是归约. 归约需要考虑在表达式中选择子表达式的选择规则.

定义 9. 一个选择规则是一个从表达式到子表达式的映射函数,函数的值是那个表达式中的被选子表达式,被选子表达式称该选择规则的可归约表达式.

传统的基于递归的表达式,归约时有 6 种选择其子表达式的规则.[7]与其对应,基于迭代的表达式也有 6 种选择规则. 限于篇幅,我们仅考虑其最左最内规则. 下面以 *LI* 表示最左最内归则.

定义 10. 让 E 是任意的表达式,$RX(E)$表示在 *LI* 下的可归约表达式集. 则有:

(1)$f(d_1,\cdots,d_n)\in RX(E)$当且仅当 $f(d_1,\cdots,d_n)$是初始或定义函数项,并且 f 不是迭代表达式的主迭代函数符号,并且 $f(d_1,\cdots,d_n)$是 E 中最左出现的子表达式.

(2)$\underset{i=e}{\overset{P}{\backslash w(i,n)}}\backslash h(t_1,\cdots,t_m)\in RX(E)$当且仅当 h 是初始或定义函数符号,并且对所有 j,$1\leqslant j\leqslant m$,t_j 是常量或非基础项,并且$\underset{i=e}{\overset{P}{\backslash w(i,n)}}\backslash h(t_1,\cdots,t_m)$是 E 中最左出现的子表达式.

(3)$L(d_1,d_2)\wedge C'\rightarrow E_1;E_2\in RX(E)$当且仅当 $L(d_1,d_2)\equiv(d_1\ r\ d_2)$或$(\neg(d_1\ r\ d_2))$,$r$ 是初始关系符号,C'是谓词,E_1 是项或迭代表达式,E_2 是表达式,并且 $L(d_1,d_2)\wedge C'\rightarrow E_1;E_2$ 是 E 中最左出现的子表达式.

显然对任何 E,$RX(E)$是一个可判定集合. 我们以下划线"——"标示可归约表达式,让 h,g 表示初始函数符号,F 表示任意的定义函数符号. 下面是在 *LI* 规则下,一个表达式中的可归约表达式的例子:

例 1:可归约表达式是项:

LI:$h(F(0,\underline{F(1)}),F(F(2),h(3,4)))$　　*LI*:$h(\underline{h(2,3)},F(5))$

$LI:(2=F\underset{P}{(F(7,5))})\rightarrow h(F(0,F(1)),F(F(2),F(3)));F(6,7)$

$LI:F(7,\backslash 1\backslash F(F(0,\underset{i=3}{\underset{P}{F(6,3)}}),F(7,8))$

$LI:F(7,\backslash w(i,n)\backslash F(F(0,\underset{i=3}{\underset{P}{F(6,3)}}),F(7,8))$

例 2:可归约表达式是条件表达式

$LI:\backslash 3\backslash F(5,\underset{P}{(2=2)}\rightarrow h(F(2),F(8));\underset{i=7}{(3=h(9))}\rightarrow h(F(7,6);F(h(6)),F(4))$

例 3:可归约表达式是迭代表达式

$$h(\underset{i=5}{\underset{P}{F(0,\backslash w(i,n)\backslash F(8,5))}}) \qquad h(\underset{i=5}{\underset{P}{F(0,\backslash 2\backslash F(8,g(i)))}})$$

若以 E 和 B 表示表达式,一个计算规则被表示为:$\frac{E}{B}$

其中,E 表示计算前的表达式,又称前表达式;B 表示计算后的表达式,又称后表达式.

让 $\theta=\{[x_1/t_1],\cdots,[x_n/t_n]\}$ 表示一组项对一组变量的替换,其中对所有的 i,$1\leqslant i\leqslant n$,x_i 表示被替换的变量,并且 $x_1,\cdots,x_n$ 是各不相同的,t_i 表示用于替换的基础项,在 LI 下 $t_i\in D_i$;$A[A_1\leftarrow A_2]$表示以 A_2 置换 A 中 A_1 的出现所得结果,若 A_1 不出现在 A 中,则置换为空,A 不变. 带迭代算子的函数式程序设计语言的解释器可以有如下的计算规则定义:

(1)M 规则——匹配规则

若 $A\equiv f'(d_1,\cdots,d_n)$ 是 E 中在 LI 下的一个可归约表达式,f' 是定义函数符号,并且存在一个 θ 和一个函数定义: $f(x_1,\cdots,x_n)=exp$

使得 $\theta f(x_1,\ldots,x_n)\equiv A$,则规则 M 可用,并且有 $\frac{E}{E[A\leftarrow\theta exp]}$.

显然,若 A 在 LI 下是 E 中唯一的被选子表达式,那么由于定义函数符号 f 是唯一的,则对任何 E,M 应用于 A 得到的后表达式是唯一的.

(2)$C-R$ 规则——条件表达式化简规则

若 $A\equiv L(d_1,d_2)\wedge C'\rightarrow E_1;E_2$ 是 E 中在 LI 下的可归约表达式,则 $C-R$ 可用,并且有 $\frac{E}{E[A\leftarrow E_2]}$;当 $L(d_1,d_2)$ 为假 $\frac{E}{E[A\leftarrow E_1]}$;当 $L(d_1,d_2)$ 为真,并且 C' 为空 $\frac{E}{E[A\leftarrow A']}$;当 $L(d_1,d_2)$ 为真,并且 C' 不空,其中 $A'\equiv C'\rightarrow E_1;E_2$.

同样,若 A 在 LI 下是 E 中唯一的被选子表达式,那么由于 $L(d_1,d_2)$ 的真假值是唯一的,则对任何 E,$C-R$ 应用于 A 得到的后表达式是唯一的.

(3)$I-E$ 规则——迭代展开规则

若 $A\equiv\underset{i=e}{\underset{P}{\backslash w(i,n)\backslash h(t_1,\cdots,t_m)}}$ 是 E 中在 LI 下的可归约表达式,则 $I-E$ 可用,并且若

$w(e,n)=a, w(e+1,n)=b$,则有:$\frac{E}{E[A\leftarrow A']}$

P

其中 $A'\equiv P[i\leftarrow e]\rightarrow t_a;\backslash w(i,n)\backslash h(t_1,\cdots,t_a,\cdots,t_m)[t_b\leftarrow h(t_1,\cdots,t_a,\cdots,t_m)[i\leftarrow e+1]]$

$i=e[e\leftarrow e+1]$

上面 $X[y\leftarrow z]$表示一个置换,并且后表达式中的所有置换是同时的.此规则把一个可归约的迭代表达式展开成一个条件表达式.

同样,若 A 在 LI 下是 E 中唯一的被选子表达式,则对任何 E,$I-E$ 应用于 A 得到的后表达式是唯一的.

(4)R 规则——初始函数项置换规则

若 $f(d_1,\cdots,d_n)$是 E 中在 LI 下的可归约表达式,f 是初始函数符号,则 R 可用,并且若 d 是 $f(d_1,\cdots,d_n)$的值,则有 $\frac{E}{E[f(d_1,\cdots,d_n)\leftarrow d]}$

同样,若 A 在 LI 下是 E 中唯一的被选子表达式,那么由于初始函数符合 f 是唯一的,则对任何 E,R 应用于 $f(d_1,\cdots,d_n)$得到的后表达式是唯一的.

4 计 算

对带迭代算子的函数式程序,计算的概念是以数据项作为最终目标,重复使用其计算规则,由旧的表达式推出新的表达式.更精确地我们有下面的定义.

定义 11. 让 S 是一个程序,E_1 是一个形如 $f(d_1,\cdots,d_n)$的表达式,其中 f 是定义函数符号,R 是一个选择规则,$Rule$ 是一个计算规则集,$S\cup\{E_1\}\cup\{Rule\}$的一个经过 R 的计算是一个由表达式组成的序列 $E_1,\cdots,E_n$,使得表达式 E_{i+1}是由表达式 E_i 在 R 下通过一个计算规则产生的新表达式.

定义 12. $S\cup\{E_1\}\cup\{Rule\}$的经过 R 的一个成功计算是一个终止的计算,并且该计算的最后表达式 A_n 是一个数据项.

一个计算可能终止或不终止.一个终止的计算可能是成功的或失败的.一个失败的计算其最后的表达式 E_n 由于没有 $S\cup\{E_n\}\cup\{Rule\}$在 R 下的计算而终止.一个失败的计算意味着程序不完全.下面我们仅显示在最左最内规则 LI 下的成功计算.

图 1 是第 2 节中单递归+迭代计算 ackermann 函数的程序执行过程.图 2 是迭代计算阶乘函数的过程.图 3 是迭代计算 fibonacci 函数的过程.限于篇幅,图 1、2 和 3 中省略了应用规则 R 的步骤,仅标以规则符号,R^* 表示重复应用 R.

5 确定性

带迭代算子的函数式程序设计语言描述的是函数,因而要求其程序所完成的计算应该是确定的.

定义 13. 让 S 是一个程序,R 是一个选择规则,E,E_1 和 E_2 是表达式 ,我们称计算规则对前表达式 E 是确定的,当且仅当在 S 和 R 下对于任何 E,E_1 和 E_2,若有:$\frac{E}{E_1}$和$\frac{E}{E_2}$,则有 $E_1=E_2$.

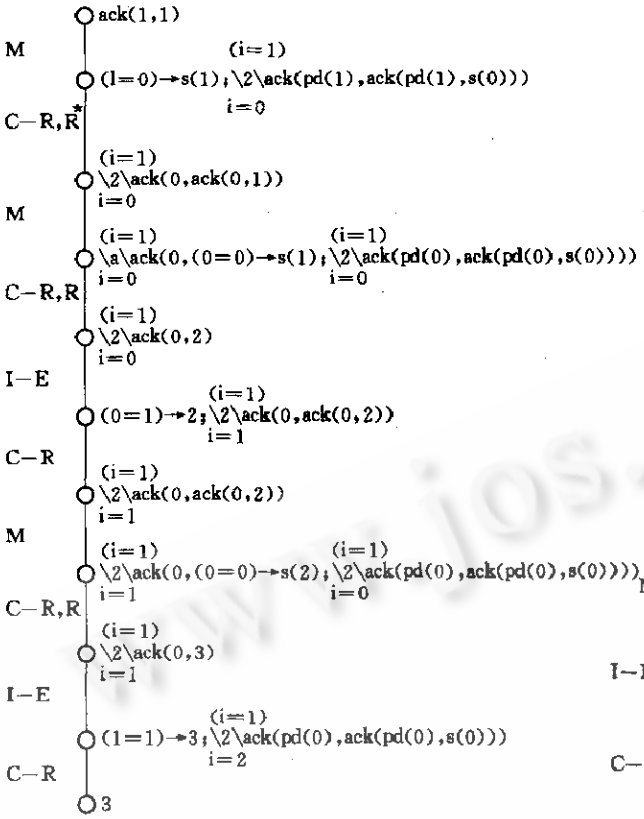

图1 单递归+迭代程序计算ackermann函数

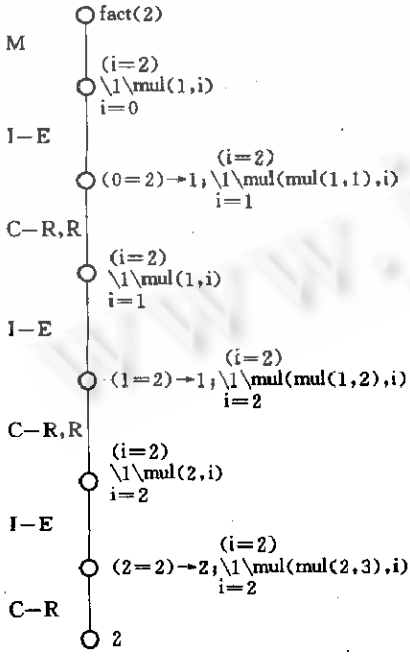

图2 迭代程序计算阶乘函数

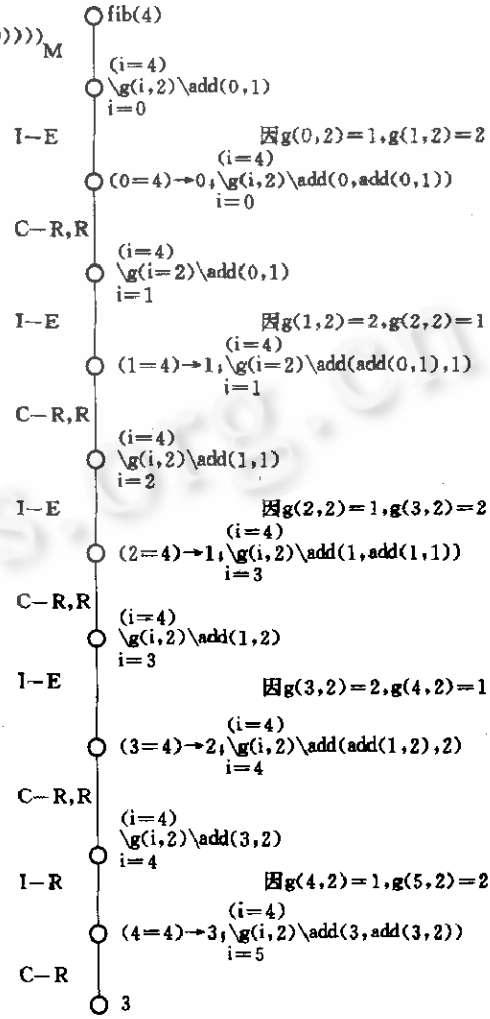

图3 迭代计算fibonacci函数

定理. 带算子的函数式程序设计语言的计算规则对前表达式在选择规则 LI 下是确定的.

证明:根据前面对每个计算规则的考察可知,对任何前表达式 E,若在 LI 下的被选子表达式是唯一的,则应用某个计算规则可得到唯一的后表达式.因此,仅需要证明:E 在 LI 下的被选子表达式是唯一的,并且对被选子表达式最多有一个计算规则可用.

任何表达式 E 是由项、条件表达式和迭代表达式构成的复合结构.下面归纳证明表达式 E 的结构.

(1)让 E 是一个任意表达式,A 是 E 在 LI 下的一个被选子表达式,它可能是一个是项,或一个条件表达式,或一个迭代表达式.根据定义 10,在选择规则 LI 下,若 A 是一个可归约表达式,$A\in RX(E)$,则 A 是 E 中最左出现的子表达式.而最左出现的子表达式总是唯一的,因此 A 在 LI 下是唯一的.

(2)当被选子表达式 A 是可归约的项时.分为定义函数项和初始函数项.若它是定义函数项,仅 M 规则可用.若它是初始函数项,仅 R 规则可用.

(3)当被选子表达式 A 是可归约的条件表达式时,仅 $C-R$ 规则可用.

(4)当被选子表达式是可归约的迭代表达式时,仅 $I-E$ 规则可用.

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, 1978,21(8):613～641.

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

3 Darlington John, Guo Yike, Pull Helen. A design space for iterating declarative languages. In: Declarative Programming, Sasbachwalden 1991, 1992.

4 Darlington John, Guo Yike, Pull Helen. Introducing constraint functional logic programming. In: Declarative programming, Sasbachwalden 1991, 1992.

5 Morris James H. Real programming in function languages. In: Function Programming And Its Applications, 1982.

6 阎志欣.状态逻辑型程序设计语言.软件学报,1994,5(10):24～32.

7 Manna Z. Mathematical theory of computation. New York: McGRAW—HILL Inc., 1974.

FUNCTION PROGRAMMING WITH ITERATION OPERATOR

Yan Zhixin

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

Abstract A new computation model of programming language, function model with iteration operator was presented. The syntax of language based on the model, the set of computation rules and a proof of determinism were given. In this language, sequence is expressed by composition of expressions, branch by condition expressions and iteration by iterative expressions, such that its programs are based on mathematical semantic founda-

tion, easy to understand, prove its correctness and constitute efficient execution system. The authors believe that this language is an efficient, useful and practical, high—level deterministic programming language with sound theoretical foundation.

Key words Iteration operator, function language, reduction, computation, determinism.