状态逻辑语言迭代程序的可计算性*

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　状态逻辑型程序设计语言是一种有坚实理论基础,既可描述递归又可描述迭代的高效、实用、新型的纯逻辑式程序设计语言.递归无疑地确保了其计算能力.本文证明了仅用状态逻辑的迭代程序就可计算部分递归函数.这就等于证明了,任何图灵可计算的对象都可用纯逻辑迭代地定义和高效地计算.

关键词　状态逻辑,迭代计算,部分递归函数,可计算性.

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.这种情况对基于 Horn 子句的逻辑型程序设计语言[2]已有讨论.对函数型程序设计语言也有类似情况.[3]近 20 年来各种陈述式语言被广泛研究[4~6],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.

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,文献[7]提出了基于迭代的状态逻辑型计算模型,给出了基于该模型的程序设计语言的语法及操作语义.该类语言是一种有良好数学基础,既可描述递归又可描述迭代的高效的、实用的、新型的纯逻辑式程序设计语言.任何一种程序设计语言,只有在其计算能力和图灵机等价时才可接受.[8]我们知道,递归函数的计算能力等价于图灵机已被证明,因此该类语言的计算能力是可保证的.但递归的时空效率很低,特别是空间效率.文献[7]提出的状态逻辑型程序设计语言是否仅用其迭代程序就可计算任何部分递归函数尚未证明.本文证明了仅用状态逻辑语言的迭代程序就可计算部分递归函数.这就等于证明了,任何图灵可计算的对象都可用有坚实理论基础的纯逻辑迭代地定义和高效地计算.

文中首先描述了文献[7]中提出的状态逻辑型计算模型的语法;区分了用状态逻辑型语言对可计算对象的递归和迭代定义;为和部分递归函数比较,给出了部分递归函数的定义;

---

* 本文研究得到国家 863 高科技项目基金资助.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 1995－09－25 收到修改稿

最后证明了仅用状态逻辑语言的迭代程序就可计算部分递归函数.

1 语　法

一阶逻辑中与逻辑蕴含有关的问题可以通过演示其子句形语句的不可满足性解决.子句形语句语法简单而又不失谓词演算的表达能力.状态逻辑语言是一种子句形语言.

定义. 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 $f(t_1,\cdots,t_n)$是一个初始函数项.若 f 是定义函数符号,则 $f(t_1,\cdots,t_n)$是一个定义函数项.

定义. 让 $t_1,\cdots,t_n$ 是项,如果 p 是 n 元定义谓词符号,则 $p(t_1,\cdots,t_n)$是一个原子谓词,简称原子.如果 p 是 n 元初始关系符号,则 $p(t_1,\cdots,t_n)$是一个约束谓词,简称约束原子.

初始关系符号如=,<,>等.一个约束原子可以中缀表示,如(x=y),(x<y).不含变量的项或原子被称为基础项或原子.

定义. 形式如下的子句是断言子句.

1)$\rightarrow A$

是事实子句,被看作一个事实的断言.其中 A 是基础原子,又称输入谓词.

2)$A,C_1,\cdots,C_n\rightarrow B \qquad n\geqslant 0$

是规则子句,被看作规则断言,A 称前置断言,B 称后置断言.让 $Var(A)$,$Var(C_i)$和 $Var(B)$分别是 A,C_i 和 B 中的变量集,则子句应满足下面条件:

(1)B 是原子,A 是仅以变量作为项的原子.

(2)C_i 是由约束原子和符号"$\rightarrow$"构成的约束文字,"$\rightarrow$"表示"非".

(3)$Var(A)=Var(C_1)\cup\cdots\cdots\cup Var(C_n)\cup Var(B)$

若 $x_1,\cdots,x_n$ 是出现在子句中的变量,则该子句的意义是"对所有的 $x_1,\cdots,x_n$,如果 A 是真,并且 $C_1,\cdots,C_n$ 都是真,则 B 是真".

3)$E\rightarrow$

是目标断言,又称目标子句.若 $x_1,\cdots,x_n$ 是出现在 E 中的变量,则该子句的意义是"对所有的 $x_1,\cdots,x_n$,E 不是真".其中 E 是仅以变量作为项的原子,又称输出谓词.

4)□

是空子句.它被解释为矛盾.

定义. 规则子句和目标子句称为程序子句.

定义. 一个状态逻辑程序是程序子句的一个有限集合.

显然,一个状态逻辑程序像传统逻辑程序一样,可以包含多个子程序.不同之处是状态逻辑程序中的每个子程序定义一个函数.

上面给出的状态逻辑语言的语法是该类语言的一个模式.如果给出一个非逻辑符号集,包括:常量符号集,初始和定义函数符号集,构成约束原子的初始关系符号集和谓词符号集,

就给出了一个特定的状态逻辑语言.

状态逻辑程序子句集的过程解释是:1)形如 $A,C_1,\cdots,C_n\rightarrow B$ 的规则子句被解释为过程声明,简称过程.其中 A 被解释为过程名,A 中的变量被解释为过程的形式参数;B 被解释为过程体,过程是一个可包含函数调用的过程调用;C_i 被解释为可包含函数调用的约束条件,当多个同名过程与一个调用(函数调用或过程调用)匹配时,所有约束条件为真的过程的过程体被执行.一个过程体或约束条件中的定义函数项或子项被解释为函数调用.这里,匹配的过程解释是在一个函数或过程调用下搜索选择一个过程,并且用该调用的实在参数替换被选过程的形式参数的过程.2)形如 $E\rightarrow$ 的目标子句被解释为输出语句,可被看做过程体为空的特殊过程.当它被调用时,以其中的函数名作为变量返回一个函数调用的计算结果.

在传统的逻辑语言中,程序的解释执行过程是由顶向下的推理过程.因而,目标子句作为启动程序的附加子句,不作为程序的子句,而事实作为程序子句.在状态逻辑语言中,程序的执行过程是由底向上的推理过程.因而,事实作为启动程序的附加子句,不作为程序的子句,而目标子句作为程序子句.

在传统的逻辑程序中,变量、定义谓词和定义函数符号是 3 个不相交的子集.而在状态逻辑程序中,程序所定义的函数符号 f 在逻辑上既可作为一个函数符号,又可作为一个谓词符号,也可作为一个变量.符号 f 的不同用法,在操作上被过程解释统一起来.因为一个规则子句被过程地解释为一过程,其中后置断言被解释为过程体,前置断言被解释为过程名.这样,当 f 出现在一个规则子句的后置断言中用于构成形如 $f(t_1,\cdots,t_n)$ 的项(或子项)时,则 $f(t_1,\cdots,t_n)$ 被过程地解释为函数调用.当 f 出现在一个规则子句的前置断言中构成一个形如 $f(x_1,\cdots,x_n)$ 的原子时,则 $f(x_1,\cdots,x_n)$ 被过程地解释为过程名.显然,像传统语言一样,一个调用可与一个过程名匹配.另外,当 f 出现在一个目标子句中用作项时,则它被解释为变量,当目标子句和某一调用匹配时,用于返回一个函数调用的值.状态逻辑程序中匹配的语义基础就是上述的过程解释.状态逻辑程序的指称语义在模型论语义下是程序的最小模型.因解释域是由常量和初始函数符号构造的海尔布朗全域,不包含由定义函数符号 f 构成的项,使得符号 f 的不同用法在逻辑上被指称语义统一起来.限于篇幅,对其指称语义的讨论有另文发表.

状态逻辑语言的操作语义是带函数调用的约束归结,在文献[7]中已给出了在按值调用的函数调用规则下的推理规则集.对递归和迭代程序,推理规则集是相同的.限于篇幅,这里不再一一列出各个推理规则.让 S 是一个状态逻辑程序;A_1 是起始断言,即事实;R 是一个函数调用规则.则 $S\cup\{A_1\}$ 的一个经过 R 的计算,是一个由断言或断言序列组成的序列 $A_1,\cdots,A_n$ 使得 A_{i+1} 是由断言或断言序列 A_i 在 R 下通过推理规则产生的新断言或断言序列;$S\cup\{A_1\}$ 的一个经过 R 的反驳是 $S\cup\{A_1\}$ 的经过 R 的一个有限的计算,即 n 有确定的值,并且 A_n 是空子句.为了后面的理解,这里仅以迭代计算阶乘函数 $fact$ 为例说明解释过程.

让谓词 $fact(x)$ 表示 $x\geqslant 0$,断言"阶乘"程序的定义域;$factp(x,i,z)$ 表示 $i\leqslant x\wedge z=i!$,断言"阶乘"迭代程序的中间状态.$factout(x,z)$ 表示 $z=x!$,断言"阶乘"程序的输出和输入关系.假定已有计算"乘"函数 ml 的子句集 S_{ml},则下面是计算 $fact$ 的迭代程序:

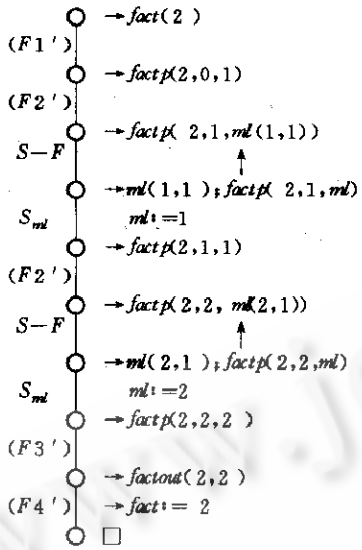
$(F1')$ $fact(x)\rightarrow factp(x,0,1)$;

$(F2')$ $factp(x,i,z),\neg(i=x)\rightarrow factp(x,s(i),ml(z,s(i)))$;

$(F3')$ $factp(x,i,z),(i=x)\rightarrow factout(x,z)$;

$(F4')$ $factout(x,fact)\rightarrow$;

为了计算 2 的阶乘,我们以事实$\rightarrow fact(2)$作为附加子句,则解释过程如下:



其中左边的$(F1')\sim(F4')$表示归结时的被选子句,$S-F$ 表示在此步计算中使用了分裂函数调用的推理规则,以便将函数调用从一个谓词中分离出来首先计算它,符号"↑"标示了被分离的函数调用. S_{ml}表示由计算函数 ml 的子句集确定的子计算,并且计算结果以函数名 ml 作为变量返回. 最后,阶乘函数的计算结果是 2 并且存储在变量 $fact$ 中. 在上面的计算过程表示中,我们忽略了判定约束条件的过程.

2　递归与迭代

传统的纯陈述式语言基于递归,难以描述迭代,这使得难以构造出高效的执行系统,因而其程序难以有高的执行效率. 状态逻辑既可描述递归又可描述迭代.

例 1:递归计算

让谓词 $fact(x)$表示 $x\geqslant 0$,$ml(x,y)$表示 $x,y\geqslant 0$,$add(x,y)$表示 $x,y\geqslant 0$,分别断言"阶乘"、"乘"和"加"程序的定义域;$factout(x,z)$表示 $z=x!$,$mlout(x,y,z)$表示 $z=x*y$,$addout(x,y,z)$表示 $z=x+y$,分别断言"阶乘"、"乘"和"加"程序的输出和输入的关系. 下面是计算这 3 个函数的递归程序:

$(F1)$ $fact(x),(x=0)\rightarrow factout(x,1)$;

$(F2)$ $fact(x),\neg(x=0)\rightarrow factout(x,ml(x,fact(pd(x))))$;

$(F3)$ $factout(x,fact)\rightarrow$;

$(M1)$ $ml(x,y),(x=0)\rightarrow mlout(x,y,0)$;

$(M2)$ $ml(x,y),\neg(x=0)\rightarrow mlout(x,y,add(y,ml(y,pd(x))))$;

$(M3)$ $mlout(x,y,ml)\rightarrow$;

$(A1)$ $add(x,y),(x=0)\rightarrow addout(x,y,y)$;

$(A2)$ $add(x,y),\neg(x=0)\rightarrow addout(x,y,s(add(y,pd(x))))$;

$(A3)$ $addout(x,y,add)\rightarrow$;

其中$(F1)\sim(F3)$,$(M1)\sim(M3)$和$(A1)\sim(A3)$是程序子句集的3个子集,它们分别定义了函数 $fact$,ml 和 add. s 是后继函数,pd 是先驱函数,它们是初始函数. 递归表现在,程序中存在这样的子句,其过程体中函数调用的函数符号与其过程名的谓词符号相同,因而构成递归调用. 如在过程$(F2)$中,其过程体中的函数调用 $fact(pd(x))$与其过程名 $fact(x)$构成递归调用. 同样,$ml(y,pd(x))$和 $add(y,pd(x))$也分别与 $ml(x,y)$和 $add(x,y)$构成递归调用. 显然递归需要大的堆栈以存储被展开的数据,而且堆栈大小随输入数据的增大而增大. 由于计算机的存储空间总是有限的,这使得对某些问题虽然可写出其递归程序,但在输入大的数据时会因堆栈溢出而计算不下去. 解决此类问题的根本途径是用迭代.

例 2:迭代计算

让谓词 $factp(x,i,z)$表示 $i\leqslant x\wedge z=i!$,$mlp(x,y,i,z)$表示 $i\leqslant x\wedge z=i*y$,$addp(x,y,i,z)$表示 $i\leqslant x\wedge z=i+y$,分别断言"阶乘"、"乘"和"加"的迭代程序的中间状态. 其它谓词与前面相同. 下面是3个函数的迭代程序:

$(F1')$ $fact(x)\rightarrow factp(x,0,1)$;

$(F2')$ $factp(x,i,z),\neg(i=x)\rightarrow factp(x,s(i),ml(z,s(i)))$;

$(F3')$ $factp(x,i,z),(i=x)\rightarrow factout(x,z)$;

$(F4')$ $factout(x,fact)\rightarrow$;

$(M1')$ $ml(x,y)\rightarrow mlp(x,y,0,0)$;

$(M2')$ $mlp(x,y,i,z),\neg(i=x)\rightarrow mlp(x,y,s(i),add(z,y))$;

$(M3')$ $mlp(x,y,i,z),(i=x)\rightarrow mlout(x,y,z)$;

$(M4')$ $mlout(x,y,ml)\rightarrow$;

$(A1')$ $add(x,y)\rightarrow addp(x,y,0,y)$;

$(A2')$ $addp(x,y,i,z),\neg(i=x)\rightarrow addp(x,y,s(i),s(z))$;

$(A3')$ $addp(x,y,i,z),(i=x)\rightarrow addout(x,y,z)$;

$(A4')$ $addout(x,y,add)\rightarrow$;

迭代表现在,程序中不存在这样的子句,其过程体中函数调用的函数符号与其过程名的谓词符号相同. 根据各原子谓词的上述意义,不难验证 $(F1)\sim(F2)$和$(F1')\sim(F3')$中每个子句的正确性. 它们分情况地描述了阶乘程序的前置状态和后置状态的逻辑关系. $(F3)$和$(F4')$的意义是对任何 x 和 $fact$,$fact=x!$ 不成立. 同样方法可验证 $(M1)\sim(M2)$,$(M1')\sim(M3')$,$(A1)\sim(A2)$和$(A1')\sim(A3')$的正确性,并且对$(M3)$,$(M4')$,$(A3)$和$(A4')$有类似于$(F3)$和$(F4')$的解释. 为了计算3的阶乘,我们附加一个子句:

$(F0')\rightarrow fact(3)$;

它断言 $3\geqslant 0$ 这一事实,作为输入谓词. $(F0')$和$(F1')\sim(F3')$逻辑上蕴含 $fact(3)$的值是6,这与$(F4')$矛盾. 证明过程从事实$(F0')$出发找这个逻辑矛盾,并且当它被找到时则产生空子句□,而且以$(F4')$中的函数名 $fact$ 作为输出变量返回计算结果6. 程序计算 $fact(3)$的过程中,通过$(F2')$中项 $ml(z,y)$的实例构成形式如$\rightarrow ml(a,b)$的附加子句以调用由$(M1')\sim(M4')$构成的子程序. 同样,程序计算 $ml(a,b)$的过程中,通过$(M2')$中项 $add(z,y)$的实例构成形式如$\rightarrow add(c,d)$的附加子句以调用由$(A1')\sim(A4')$构成的子程序. 我们也可以用

形如→$ml(a,b)$和→$add(a,b)$的附加子句计算在输入为a和b时"乘"和"加"函数的值.对递归程序也有类似的计算.

3 部分递归函数

为了研究状态逻辑语言迭代程序的计算能力,下面我们仅在自然数集上给出部分递归函数的定义.

定义. 下面自然数集N上的函数,是部分递归函数的初始函数:

(1)常函数$c_k=k$,其中$k\in N$;

(2)后继函数$s(x)=x+1$,其中$x\in N$;

(3)投影函数$u_i^n(x_1,\cdots,x_n)=x_i$,其中对所有$i$和$n$,$1\leqslant i\leqslant n$,$x_i\in N$.

定义. 让$h,g_1,\cdots,g_k$是部分递归函数,其中$h:N^k\to N$是k元函数,对每个j,$1\leqslant j\leqslant k$,$g_j:N^n\to N$是n元函数.则函数f:

$$f(x_1,\cdots,x_n)=h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$$

是由h和$g_1,\cdots,g_k$经过复合运算生成的部分递归函数.

定义. 让h,g是部分递归函数,其中$h:N^{n+1}\to N$是$n+1$元函数,$g:N^{n-1}\to N$是$n-1$元函数,则函数f:

$$f(0,x_2,\cdots,x_n)=g(x_2,\cdots,x_n)$$

$$f(n+1,x_2,\cdots,x_n)=h(f(n,x_2,\cdots,x_n),n,x_2,\cdots,x_n)$$

是由h和g经过原始递归运算生成的部分递归函数.

定义. 让$q:N^{n+1}\to N$是任何$n+1$元全函数,则函数f:

$$f(x_1,\cdots,x_n)=\mu y[q(y,x_1,\cdots,x_n)=1]$$

是由q通过最小化运算μ生成的部分递归函数.如果使得$q(y,x_1,\cdots,x_n)=1$为真,y存在,则f有定义,并且$f(x_1,\cdots,x_n)$的值就是最小的y,否则f无定义.

定义. 一个函数是部分递归的,当且仅当它可由初始递归函数通过有限次地应用复合运算、原始递归运算和对任何全函数应用最小化运算生成.

4 可计算函数

状态逻辑程序计算的是任意域上的函数.为了研究其迭代程序的计算能力,下面我们仅考虑自然数集上的数论函数;首先给出3个初始函数和一个初始谓词;然后证明仅用状态逻辑语言的迭代程序可计算由复合、原始递归和最小化运算构成的部分递归函数.

定义. 下面自然数集N上的函数是状态逻辑的初始函数:

(1)常函数$c_k=k$,其中$k\in N$;

(2)后继函数$s(x)=x+1$,其中$x\in N$;

(3)投影函数$u_i^n(x_1,\cdots,x_n)=x_i$,其中对所有$i$和$n$,$1\leqslant i\leqslant n$,$x_i\in N$.

定义. 让符号"="表示比较2个自然数的相等关系,则$(x=y)$是初始谓词,其中$x,y\in N$.

为证明部分递归函数可由状态逻辑的迭代程序计算,后面用到了结构归纳法.

下面定理给出了结构归纳的一般方法.[9]

定理(Structural induction, Burstall). 让$(S,<)$是一个良序集,φ是S上的全谓词:

如果$(\forall a\in S)\{[(\forall b\in S$ 使得 $b<a)\varphi(b)]\supset\varphi(a)\}$,则$(\forall c\in S)\varphi(c)$.

自然数集就通常的$<$(小于)关系是良序集.用结构归纳法证明$(\forall c\in S)\varphi(c)$的步骤是:首先对自然数集上的最小数0,证明$\varphi(0)$为真;然后假定对任意自然数$x$,$\varphi(x)$为真,证明$\varphi(x+1)$为真.本节给出鉴别状态逻辑的迭代程序计算能力的一个重要结果如下:

定理. 一个函数是部分递归的,则它是状态逻辑的迭代程序可计算的.

证明:如果$f:N^n\to N$是部分递归的,则我们要证明:f可由状态逻辑的迭代程序计算.为此,我们根据部分递归函数f的定义,归纳证明于函数的结构:

(1)所有初始递归函数,即常函数$c_k=k$,后继函数$Succ(x)=x+1$和投影函数$u_i^n(x_1,\cdots,x_n)=x_i$,都是状态逻辑的初始函数,显然它们是状态逻辑程序可计算的.

(2)让$h,g_1,\cdots,g_k$是部分递归函数,其中$h:N^k\to N$是k元函数,对所有的i,$1\leqslant i\leqslant k$,$g_i:N^n\to N$是n元函数,通过复合运算生成的部分递归函数f是:

$$f(x_1,\cdots,x_n)=h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$$

假设函数$h,g_1,\cdots,g_k$可用状态逻辑的迭代程序计算,我们要证明下面由$h,g_1,\cdots,g_k$构成的迭代程序所计算的函数f'就是函数f:

$(F1)$ $f'(x_1,\cdots,x_n)\to fout(x_1,\cdots,x_n,h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n)))$;

$(F2)$ $fout(x_1,\cdots,x_n,f')\to$;

因计算$h,g_1,\cdots,g_k$的子程序是迭代程序,则上面的程序和所有子程序构成一个迭代程序.其中谓词$f'(x_1,\cdots,x_n)$表示$x_1,\cdots,x_n\geqslant 0$,断言程序的定义域;$fout(x_1,\cdots,x_n,z)$表示$z=f'(x_1,\cdots,x_n)$,断言程序的输入输出关系.让$x_1,\cdots,x_n$是任意特定的自然数,则我们可把$\to f'(x_1,\cdots,x_n)$作为事实子句并由它出发有如下计算过程和结果:

$a)\to f'(x_1,\cdots,x_n)$;	起始事实
$b)\to fout(x_1,\cdots,x_n,h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n)))$;	由$a)$和$(F1)$归结
$c)\to\square$ $f':=h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$	由$b)$和$(F2)$归结

在计算步骤$c)$中f'被赋以值$h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$并得空子句而停机.停机时$f'=h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n))$,这意味着停机时有:

$$f'(x_1,\cdots,x_n)=h(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n)).$$

因而$f'(x_1,\cdots,x_n)=f(x_1,\cdots,x_n)$.

(3)由部分递归函数h,g通过原始递归运算生成的部分递归函数f是:

$$f(0,x_2,\cdots,x_n)=g(x_2,\cdots,x_n);$$

$$f(n+1,x_2,\cdots,x_n)=h(f(n,x_2,\cdots,x_n),n,x_2,\cdots,x_n)$$

其中$h:N^{n+1}\to N$是$n+1$元函数,$g:N^{n-1}\to N$是$n-1$元函数.假设函数h,g是状态逻辑的迭代程序可计算的.我们要证明,下面由h,g,s及初始谓词$=$构成的迭代程序所计算的函数f'就是函数f:

$(F1)$ $f'(x_1,\cdots,x_n)\to fp(x_1,\cdots,x_n,0,g(x_2,\cdots,x_n))$;

$(F2)$ $fp(x_1,\cdots,x_n,i,z),\neg(i=x_1)\to fp(x_1,\cdots,x_n,s(i),h(z,i,x_2,\cdots,x_n))$;

$(F3)$ $fp(x_1,\cdots,x_n,i,z),(i=x_1)\to fout(x_1,\cdots,x_n,z)$;

(F4) $fout(x_1,\cdots,x_n,f')\rightarrow$;

其中谓词 $f'(x_1,\cdots,x_n)$表示 $x_1,\cdots,x_n\geqslant 0$,断言程序的定义域;$fp(x_1,\cdots,x_n,i,z)$表示 $i\leqslant x_1\wedge z=f'(i,x_2,\cdots,x_n)$,断言程序的中间状态;$fout(x_1,\cdots,x_n,z)$表示 $z=f'(x_1,\cdots,x_n)$,断言程序的输入输出关系.

该程序的归纳变量和上面的原始递归函数一样,是 x_1. 其中 i 是 x_1 的逼近变量,当 $i=x_1$时得计算结果 z,并由(F4)中的函数名 f'作为变量返回结果 z 的值.

让谓词 $\varphi(n)$是$(\forall n\in N)[f(n,x_2,\cdots,x_n)=f'(n,x_2,\cdots,x_n)]$,这是一个全谓词,并且$(N,<)$是良序集[8],因此可用结构归纳法证明如下:

1)当 $n=0$ 时,让 $x_2,\cdots,x_n$ 是任意的自然数,我们要证明:

$f(0,x_2,\cdots,x_n)=f'(0,x_2,\cdots,x_n)$,即 $\varphi(0)$为真. 为了计算 $f'(0,x_2,\cdots,x_n)$的值,我们用$\rightarrow f'(0,x_2,\cdots,x_n)$作为附加子句并由它出发,则有下面计算过程和结果:

$a)\rightarrow f'(0,x_2,\cdots,x_n)$;　　附加子句

$b)\rightarrow fp(0,x_2,\cdots,x_n,0,g(x_2,\cdots,x_n))$;　　由 $a)$和(F1)归结

$c)\rightarrow fout(0,x_2,\cdots,x_n,g(x_2,\cdots,x_n))$;　　由 $b)$和(F3)归结

$d)\ \square\ f':=g(x_2,\cdots,x_n)$　　由 $c)$和(F4)归结

在计算步 $d)$中 f'被赋以值 $g(x_2,\cdots,x_n)$并得空子句而停机. 停机时 $f'=g_1(x_2,\cdots,x_n)$,这意味着有 $f'(0,x_2,\cdots,x_n)=g(x_2,\cdots,x_n)$. 而根据原始递归函数的定义 $f(0,x_2,\cdots,x_n)=g(x_2,\cdots,x_n)$,因而在 $n=0$ 时 $\varphi(0)$为真.

2)当 $n>0$ 时,假设 $f(n,x_2,\cdots,x_n)=f'(n,x_2,\cdots,x_n)$,即 $\varphi(n)$为真. 我们要证明:$f(n+1,x_2,\cdots,x_n)=f'(n+1,x_2,\cdots,x_n)$,即 $\varphi(n+1)$为真. 根据归纳假设 $f(n,x_2,\cdots,x_n)=f'(n,x_2,\cdots,x_n)$,则由 fp 的意义不难验证:$fp(n+1,x_2,\cdots,x_n,n,f(n,x_2,\cdots,x_n))$为真,因而有:

$a)\rightarrow fp(n+1,x_2,\cdots,x_n,n,f(n,x_2,\cdots,x_n))$;

用 $a)$与(F2)归结有:

$b)\rightarrow fp(n+1,x_2,\cdots,x_n,s(n),h(f(n,x_2,\cdots,x_n),n,,x_2,\cdots,x_n))$;

根据递归函数的定义有 $h(f(n,x_2,\cdots,x_n),n,,x_2,\cdots,x_n)=f(n+1,x_2,\cdots,x_n)$则有:

$c)\rightarrow fp(n+1,x_2,\cdots,x_n,s(n),f(n+1,x_2,\cdots,x_n))$;

因 $s(n)=n+1$,则可用 $c)$与(F3)归结得:

$d)\rightarrow fout(n+1,x_2,\cdots,x_n,f(n+1,x_2,\cdots,x_n))$;

再用 $d)$与(F4)归结得:

$e)\ \square\ f':=f(n+1,x_2,\cdots,x_n)$

在计算步 $e)$有 $f'=f(n+1,x_2,\cdots,x_n)$,这意味着 $f'(n+1,x_2,\cdots,x_n)=f(n+1,x_2,\cdots,x_n)$. 因而在 $n>0$ 时 $\varphi(n+1)$为真.

(4)由任何 $n+1$ 元全函数 $q:N^{n+1}\rightarrow N$ 通过最小化运算生成的部分递归函数 f 是:

$$f(x_1,\cdots,x_n)=\mu y[q(y,x_1,\cdots,x_n)=1]$$

假设函数 q 是可用状态逻辑的迭代程序计算的全函数. 我们要证明下面的由 q 和初始函数 s 构成的迭代程序所计算的函数 f'就是函数 f:

(F1) $f'(x_1,\cdots,x_n)\rightarrow fp(x_1,\cdots,x_n,0)$;

(F2) $fp(x_1,\cdots,x_n,i),\neg(q(i,x_1,\cdots,x_n)=1)\rightarrow fp(x_1,\cdots,x_n,s(i))$;

(F3) $fp(x_1,\cdots,x_n,i),(q(i,x_1,\cdots,x_n)=1)\rightarrow fout(x_1,\cdots,x_n,i)$;

(F4) $fout(x_1,\cdots,x_n,f')\rightarrow$;

其中谓词 $f'(x_1,\cdots,x_n)$表示 $x_1,\cdots,x_n\geqslant 0$,断言程序的定义域;$fp(x_1,\cdots,x_n,i)$表示 $i\leqslant f'(x_1,\cdots,x_n)$,断言程序的中间状态;$fout(x_1,\cdots,x_n,z)$表示 $z=f'(x_1,\cdots,x_n)$,断言程序的输入输出关系.

函数 f 的值是使得谓词$(q(y,x_1,\cdots,x_n)=1)$为真的最小自然数 y,若满足条件的 y 不存在时 f 无定义.而状态逻辑的迭代程序从最小的自然数 $i=0$ 开始测试谓词 q 的值是否为真,每测试 1 次,i 的值加 1,并且当第一个使得 q 为真的 i 被找到时则停机,此时的 i 就是最小的使得$(q(i,x_1,\cdots,x_n)=1)$为真的 i,若满足条件的 i 不存在则不停机,既 f' 无定义.证明分 2 种情况:

1)若不存在自然数 n 使得$(q(n,x_1,\cdots,x_n)=1)$为真,要证明 f 和 f' 都无定义.显然 f 无定义.现证明 f' 也无定义.让 $x_1,\cdots,x_n$ 是任意的自然数,并以$\rightarrow f'(x_1,\cdots,x_n)$作为事实子句,则有:

$a)\rightarrow f'(x_1,\cdots,x_n)$; 附加子句

$b)\rightarrow fp(x_1,\cdots,x_n,0)$; 由 $a)$和$(F1)$归结

因不存在 i 使得$(q(i,x_1,\cdots,x_n)=1)$为真,即对任何 i,$\neg(q(i,x_1,\cdots,x_n)=1)$总是真,则有下面不终止的计算过程:

$c)\rightarrow fp(x_1,\cdots,x_n,s(0))$; 由 $b)$和$(F2)$归结

$d)\rightarrow fp(x_1,\cdots,x_n,s(s(0)))$; 由 $c)$和$(F2)$归结

$e)\rightarrow fp(x_1,\cdots,x_n,s(s(s(0))))$; 由 $d)$和$(F2)$归结

……

这里,$s(0),s(s(0)),s(s(s(0))),\cdots$,分别是 $1,2,3,\cdots$.此过程将无限地继续下去,即计算 f' 的程序$(F1)\sim(F4)$不终止.因而 f' 无定义.

2)若存在自然数 n 使得$(q(n,x_1,\cdots,x_n)=1)$为真,要证明:

$$f(x_1,\cdots,x_n)=n\wedge f'(x_1,\cdots,x_n)=n$$

显然,当$(q(n,x_1,\cdots,x_n)=1)$为真时必有 $f(x_1,\cdots,x_n)=n$.下面证明 $f'(x_1,\cdots,x_n)=n$,让 $x_1,\cdots,x_n$ 是任意特定的自然数,并以$\rightarrow f'(x_1,\cdots,x_n)$作为事实子句,则有下面计算过程和结果:

$a)\rightarrow f'(x_1,\cdots,x_n)$; 起始事实

$b)\rightarrow fp(x_1,\cdots,x_n,0)$; 由 $a)$和$(F1)$归结

后面从 $b)$出发与$(F2)$归结得一个新的后置断言,然后再用新的后置断言与$(F2)$归结.此过程重复下去,直到找到一个 i 使得$(q(i,x_1,\cdots,x_n)=1)$为真为止.根据假设,当 i 是自然数 n 时有$(q(i,x_1,\cdots,x_n)=1)$为真,我们要证明 $f'(x_1,\cdots,x_n)=n$.当我们推得后置断言$\rightarrow fp(x_1,\cdots,x_n,n-1)$时,因$(q(n-1,x_1,\cdots,x_n)=1)$为假,$\neg(q(n-1,x_1,\cdots,x_n)=1)$为真,则$(F2)$可与 $fp(x_1,\cdots,x_n,n-1)$归结得:

$c)\rightarrow fp(x_1,\cdots,x_n,s(n-1))$;

因 $s(n-1)=n$,则有:

$d)\rightarrow fp(x_1,\cdots,x_n,n)$;

此时 $d)$与$(F3)$归结有:

$e)\rightarrow fout(x_1,\cdots,x_n,n)$;

$f)\ \square\ f'=n$　　由 $e)$和$(F4)$归结

在计算步 $f)$中 f'被赋以值 n,并得空子句而停机.因而 $f'(x_1,\cdots,x_n)=n$.

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 Kowalski R. Predicate logic as programming language. Proc. IFIP Cong. 1974, Amsterdam: North Holland Pub. Co., 1974. 569～574.

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

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

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

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

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

8 Kfoury A J, Moll Robert N, Arbib Michael A. A programming approach to computability. New York: Springer Verlag New York Inc., 1982.

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

THE COMPUTABILITY OF ITERATION PROGRAMS IN STATE LOGIC LANGUAGE

Yan Zhixin

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

Abstract　Programming language based on state logic is an efficient, useful and practical, new pure logical programming language with sound theoretical foundation. It can represent recursion and iteration. With recursion its computability can be provided. This paper proves that partial recursive functions can be computed only with iteration programs in state logic. It means that any computable object in Turing machine can be defined iteratively and computed efficiently with pure logic.

Key words　State logic, iteration computing, partial recurssion functions, computability.