中国科学 E 辑：信息科学 2008 年 第 38 卷 第 6 期：829 ~ 842
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE IN CHINA PRESS

量子程序设计语言初探

徐家福*，宋方敏

南京大学 计算机软件新技术国家重点实验室，南京 210093
* E-mail: xjf@nju.edu.cn

收稿日期: 2007-11-30; 接受日期: 2008-01-21
国家自然科学基金资助项目(批准号: 60721002)

摘要 结合简述几种有代表性的量子程序语言，着重阐明量子计算、语言风范、程序结构、输入输出、异常机制，以及南京大学量子计算研究组的新近研究成果，即函数式量子程序设计语言 NDQFP. 附录中给出了 NDQFP 之所有原始函数及组合型.

关键词
量子程序设计语言
语言风范
命令式程序设计语言
申述式程序设计语言
词法分析程序
语法分析程序
汇编程序
解释程序

1 量子计算

1.1 理解[1~4]

量子计算指的是基于量子力学基本原理的计算.

1.1.1 数据

量子态(叠加态)可表示成可分 Hilbert 空间元素

$$|\psi>=\alpha_1|\psi_1>+\ldots+\alpha_n|\psi_n>,$$

其中诸 $|\psi_i>(i=1,\ldots,n)$ 为基向量，诸 $\alpha_i(i=1,\ldots,n)$ 为复系数，且 $|\alpha_1|^2+\cdots+|\alpha_n|^2=1$.

特点是：未测量时，量子态为一叠加态，即其多个基态(基向量)的线性组合；测量后，量子态可随机蹦缩至其中任何一个基态. 这是量子计算优于经典计算的原因之一.

1.1.2 运算

量子计算过程一般可归结为初化、演化、终化三阶段. 初化即初态制备，对应于经典计算

的置初值; 演化即对量子态进行各种酉运算; 完毕后, 即进入终化, 以便观察结果. 此时, 量子态蹋缩为基态, 化为经典态. 因此, 原则上包含两类运算, 一类是酉运算, 为可逆运算, 施于各种量子态; 另一类是测量运算, 为不可逆运算, 对应于由量子态蹋缩为基态. 值得注意的是, 由于酉运算的线性特征而导致量子并行性, 这是量子计算优于经典计算的另一重要原因.

1.2 特点

• 兼有两类运算, 即可逆运算与不可逆运算;

• 演化过程不可见, 由于演化是在封闭的量子设备上进行, 不为人见;

• 终化之破坏性与随机性. 终化将量子态蹋缩至基态, 破坏了量子态, 而且这种蹋缩是随机性的, 不可预见的;

• 无清除运算.

1.3 语言

语言用于描述算法, 算法刻画计算过程. 计算过程包括计算与控制. 一般计算(指量子计算)在量子设备上进行, 控制(指计算流程控制)在经典机上进行. 所设想的量子计算机由量子设备与经典机两部分组成, 为混成式机器. 同时, 量子算法中亦可包含经典计算. 因而, 量子程序设计语言亦是混成式的, 既包含用于描述量子计算过程的量子成分, 又包含用于描述经典计算过程的经典成分.

2 语言风范

2.1 理解

语言风范指的是计算过程之表现风格与规范. 申言之, 它可刻画为以下三点;

一是反映语言设计之机器背景; 二是反映计算过程之刻画重点; 三是影响语言程序之结构.

2.2 种类

大体上说, 语言风范可分为命令式风范与申述式风范两类.

2.2.1 命令式风范语言[5]

以 von Neumann 结构机器(机器指令逐条执行, 表现为状态改动, 程序内存)为设计背景; 计算过程之刻画重点在“如何做”, 强行规定计算过程中各个计算步之执行次序, 计算步之执行表现为计算状态之改动.

语言程序之结构为运算列, 反映计算以运算为中心, 虽然运算要起作用, 必须作用于数据, 但考虑之重点在运算一方.

2.2.2 申述式风范语言[6]

以非 von Neumann 结构机器(如数据流机考虑计算过程中数据之流程, 无计算状态)为设计背景.

计算过程之刻画重点在“做什么”. 虽然“做什么”要落实到“如何做”, 但考虑之重点在“做

什么”. 如 FP 语言中重点考虑函数之功能, 函数如何实现其功能用户无须考虑. 无计算状态.

语言程序之结构为作用列. 如FP语言中, 即为函数列. 函数作用于对象(数据)表示函数之执行. 在一定程度上, 反映以数据为中心.

2.3 命令式量子程序设计语言

2.3.1 量子伪码[7]

一种用于书写量子算法之量子伪码, 其中给出了量子寄存器与经典寄存器之区别约定, 量子寄存器纠缠范围约定, 以及量子寄存器之初化、使用、测量等约定. 严格说来, 量子伪码尚难称作一种量子程序设计语言.

2.3.2 qGCL[8]

一种在 pGCL(概率卫氏命令语言)基础上增添不确定性(描述某些量子算法所需)与概率(描述“观察”量子系统所需)之命令式量子程序设计语言, 用以刻画量子计算之初化、演化、终化三个基本阶段.

2.3.3 QCL[9]

一种结构化命令式量子程序设计语言, 程序是语句与定义列. 定义分常量定义与变量定义. 语句有简单语句、流程控制语句、交互命令三类. 类型有标量、张量、量子、布尔、串等, 其中量子类型又分通用、常量、目标、清除四种类型.

2.3.4 NDQJava

一种在 Java 之基础上增添量子成分之命令式量子程序设计语言, 所增添之主要量子成分有量子类型、量子变量、量子表达式、量子语句、量子分程序等, 已在经典机上模拟实现.

2.4 申述式量子程序设计语言

2.4.1 Qgol[10]

一种描述量子算法之图示语言(图之结点表示数据, 连线表示操作), 作者阐明了对函数式语言的看法, 指出函数式语言与量子系统间的某些对应, 尚难看出 Qgol 本身是一种函数式语言(暂行列入此类). 已开发出一个图形编辑程序.

2.4.2 QML[11]

一种有限类型之函数式量子程序设计语言. 它与“经典控制、量子计算”不同, 提出了“量子数据、量子控制”, 利用一阶严格线性逻辑使弱化得以显式表达, 程序是一列项定义. 以范畴论为基础、给出了 QML 的指称语义与操作语义.

2.4.3 NDQFP

一种函数式量子程序设计语言, 其主要特点: 1), 它是一种模块式语言, 程序是一个或多个模块; 2), 它是一种有类型的语言. 除整型、实型、复型等经典类型外, 定义了一种量子类型; 3), 定义了 in 及 out 输入、输出成分; 4), 定义了异常机制(详见 NDQFP 概述一节).

2.5 命令式语言与申述式语言之比较

2.5.1 从语言设计之机器背景看: 命令式语言之设计背景为 von Neumann 结构机器、申述式语言则相反. 由于量子算法具申述性, 故而作为主要用以描述量子算法之量子程序设计语言,

采用申述式风范较为合宜, 但又因当前为程序设计语言可用之量子计算机尚未问世, 只能在经典计算机上模拟实现量子程序设计语言. 因此, 从长远看, 申述式量子程序设计语言较宜. 从目前看, 何者利大弊小, 尚待实践检验.

2.5.2 从反映计算过程之刻画重点看: 命令式语言重点在刻画“如何做”, 申述式语言重点在刻画“做什么”, 前者粒度小, 后者粒度大. 因此, 从理论上说, 粒度大者, 易读性好. 然而联系到程序阅读者之数学素养, 实际情形又可能是命令式语言程序易于阅读.

2.5.3 从影响语言程序之结构看: 命令式语言程序一般为运算列, 申述式语言程序一般为作用列. 当然, 一般作用列之级别高于运算列. 从程序之易读性以及有利于程序正确性的角度看, 申述式语言程序为宜, 考虑到当前之模拟实现平台一般仍为经典计算机, 未能体现量子计算之真正优越性, 而且联系到其设计之机器背景, 命令式语言又较合适. 二者之利弊得失, 仍待实践检验.

3 程序结构

3.1 理解

程序结构指的是程序之一级组成成员之集合与各组成成员之间的联系.

3.2 几种不同的程序结构

3.2.1 模块式结构[12]

基于“分而治之”原则, 问题分解成一列可分解之子问题. 程序由一个或多个模块组成, 换言之, 程序为模块列. 模块用以封装数据与过程或其他成分. 模块彼此之间的联系通过所属过程以及移入、移出设施进行. 优点是: 语言的层次性强, 导致程序的层次性强, 从而导致程序的易读性强. 缺点是: 在某些情形, 类似人们穿上一件不必要的外衣, 酿成浪费. 模块的定义方式有多种: 如移入、移出表方法, 模块式与模块体分离法, 亦有此二者相结合的方法等等, 各有利弊.

3.2.2 函数式结构[13]

函数是程序的核心, 程序为函数列. 函数作用于对象(数据)反映程序的执行. 函数与函数之间的联系直接, 前者的结果值为后者的变元值. 优点是: 理论严谨、结构简明, 有利于保证程序之正确性, 而且程序紧凑, 信息量少. 缺点是: 层次性差、不易阅读, 对数学素养平平者, 更是如此.

3.2.3 面向对象结构[14]

在纯面向对象语言中, 程序为类列. 类为相同结构对象的集合或抽象表示. 类之实例为对象, 对象是运行实体. 程序设计人员从考察现实系统中之对象出发, 上升为类, 再由类上升为程序. 程序处理系统则反之. 由于量子系统不一定能分解成一组可分之子系统, 且其各个组成部分之间的消相干为一重要而棘手的待解问题, 因此, 面向对象结构和面向对象语言可能不宜用作量子程序结构和量子程序设计语言.

3.2.4 其他结构

将程序定义为说明与语句列之程序结构不少. 于此不拟赘述.

4 输入输出

输入输出设施为经典语言的重要组成部分. 量子程序设计语言亦然. 个别语言(如 FP)中并未设有输入输出设施, 而将此工作委交语言处理系统统一解决, 这样的处理方式似应看成语言本身之缺陷. FP 语言自 1977 年推出后, 迄今未见实用, 可能输入输出设施欠缺亦其原因之一.

问题在于如何定义输入输出设施. 一般情形有两种: 一种是将外设与文件分开, 即分别定义两类设施(语句或过程), 由外设输入与输出到外设上的设施名为 in 与 out. 由文件输入与输出到文件上的设施名为 get 与 put. 分开的好处是: 程序易读性好, 处理简便; 缺点是: 语言成分增加, 增添了处理程序的额外工作. 另一种方法则是不区分外设与文件, 而统一成一类设施. 如只定义输入设施(语句或过程)in 与输出设施(语句或过程)out. 二者既可用于由外设输入与输出到外设上, 又可用于由文件输入与输出到文件上. 其优缺点和第一种方法相反.

在量子程序设计语言中, 输入对应于初态制备, 输出对应于观察结果. 在函数式量子程序设计语言中, 初化借助函数 in, 但由于输入函数之值和时间有关. 在时刻 t_1 外设上的当前值一般不等于在时刻 t_2 外设上的当前值. 如不引进时间参数, 则输入函数将成为多值函数, 不符合函数式语言中对函数的单值要求. 为了解决这一问题, 有的语言如 QML 添加一“历史”参数, 但此参数对程序人员透明, 由处理系统进行处理.

5 异常机制[15]

异常乃正常之反, 有正常就必有异常, 无异常亦无正常之可言. 在早期出现的程序设计语言中, 并未考虑设有异常机制. 在计算过程中, 一旦出现异常现象, 则由系统统一处理, 或借助人工干预, 从而影响到处理之自动化程度, 用户有感不便.

随着程序设计语言之发展, 语言中逐渐设有异常机制. 要解决的问题有三: 一是要明确异常之定义; 二是要刻画异常之出现; 三是当异常出现时, 如何进行处理. 欲解决第 1 问题, 语言中要明确何谓异常(如计算过程中出现死循环, 程序运行永不终止, 或者运算作用于不合适的数据, 而迫使程序运行终止). 可定义两类异常, 一类是语言定义的, 另一类是由用户自行定义的. 为了解决第 2 问题, 语言中引进刻画异常出现的设施. 如引进异常引发语句(或函数), 程序执行到此类语句(或函数)时, 迫使程序挂起, 自动转至相应的异常处理程序处理之. 解决第 3 问题的方法是, 定义一组合适的异常处理程序, 用以处理各类异常.

语言不同, 异常处理程序执行完毕后的处置, 又可区分两种情形, 一种是自动停机, 停机后如何做, 由程序人员决定; 另一种则是自动转回到异常的发生点处, 再继续往后运行. 两种方案, 难分优劣, 应根据不同语言、不同的处理问题类而定, 不好一概而论.

6 NDQFP 概述

6.1 研究动因[16,17]

鉴于量子计算对某些问题而言, 其优越性远非经典计算所可比拟, 虽然当前可用之量子

计算机尚未问世, 并且在研制中尚存在相当困难, 但是, 进展喜人. 有人预测, 2030 年左右, 即可出现实用之量子计算机; 另一方面, 南京大学研究软件语言多年, 积累了一定经验, 因此, 2004 年组织了量子计算研究组, 确定以量子程序设计语言为其主攻方向, 在学习前人工作之基础上, 于 2006 年 6 月完成了一种基于 Java 的命令式量子程序设计语言 NDQJava 的设计与实现. 由于量子算法本质上具有申述性, 看来, 采用申述式量子程序设计语言书写量子算法, 较采用命令式语言在理论上似更合宜. 故而, 2006 年 9 月起, 我们循申述式途径(函数式是申述式的一种)设计并实现了一种函数式量子程序设计语言 NDQFP. 设计中吸取了 FP, FL, Haskell 以及 NDQJava 等语言设计的有益经验, 在语言总体及局部上均有较大改动.

6.2 研制准则[18]

6.2.1 实用性

设计语言之目的在于用, 在于用以书写算法. 特别是, 设计量子程序设计语言之目的主要是用以书写量子算法. 为此, 一是必须能写; 二是必须好写. 能写之义自明; 好写者, 意指程序人员稍经学习训练, 即不难试用, 由试用到使用, 由使用到乐用. 更为要者, 程序人员写出之程序, 必须能正确运行, 起到实际作用.

6.2.2 简练性

简练者, 简明练达之谓也. 简明乃烦琐之反; 意即, 内容简要、形式明了. 内容简要, 意指抓住主要矛盾. 举凡语言结构之拟定, 成分之取舍, 处理系统之设计途径、系统结构, 以及实现方法等均能提纲挈领, 言简意骇. 形式明了意指, 语言及其处理系统在用户面前之展现形式较为自然, 用户不致望而生畏, 就之却步. 因此, 用户乐于使用. 练达者, 精练达意也, 语言及其处理系统纲举目张, 层次分明, 有助于书写中避免错误, 实现中易于查错、排错.

6.2.3 功效性

语言也好, 处理系统也好, 为使用户乐于使用, 必须有较高功效. 功效之表现不外空间与时间. 空间意指处理系统产生之目标程序以及处理系统本身所占之存储空间; 时间指的是: 目标程序之运行速度或程序之解释执行速度, 以及处理系统之处理速度. 于此, 必须注意以下 3 点: 一为功效之义相对; 二为应侧重目标程序或解释程序之功效; 三为宜在保证正确性之前提下注意功效.

6.3 语言概貌

6.3.1 模块式结构

程序由一个或多个模块组成, 其中有一个为主模块, 主模块中有一个主函数, 主函数乃程序之入口.

模块用以封装一组定义, 其中包括类型定义、函数定义, 以及异常定义. 但主模块之模块式中尚含如下构造, 即: 主函数: 对象. 它表示将主函数作用于对象所得之结果对象, 记作 MC.

6.3.2 模块定义方式

每一模块由模块式与模块体组成.

模块式: 刻画“做什么”(和外界接口信息),

可能有移出表、移入表.

模块体: 刻画“如何做”(对用户可隐蔽之实现细节).

模块呈下形:

module specification ML;

[**export** $t_\varepsilon,\ldots,t_q$; $f_q,\ldots,f_w$; $e_m,\ldots,e_p$; $(\varepsilon,\ldots,\eta\in(1,\ldots,n),\ q,\ldots,w\in(1,\ldots,g),m,\ldots,p\in(1,\ldots,s)]$

[**import** $t_r,\ldots,t_s;f_e,\ldots,f_y;e_f,\ldots,e_g;$]

[MC]

$t_1;t_2;\ldots;t_n;$

$f_1;f_2;\ldots;f_g;$

$e_1;e_2;\ldots;e_s;$

end ML;

module body ML;

tdef t_1=类型;

⋮

tdef t_n=类型;

fdef f_1=函数表达式;

⋮

fdef f_g=函数表达式;

edef e_1=串;

⋮

edef e_s=串;

end ML;

移出表 **export** $t_\varepsilon,\ldots,e_p$; 中所列元素表示在本模块中定义, 在其他模块中可以使用之成分;

移入表 **import** $t_r,\ldots,e_m$; 中所列元素表示在其他模块中定义, 在本模块中可以使用之成分.

6.3.3 函数

定义了两类函数, 一类是语言定义的, 即一组原始函数[17,18](详见附录A); 另一类则是用户借助函数定义来定义的函数.

函数定义呈下形:

fdef f =函数表达式.

其中的函数表达式由函数名、对象名、借助组合名组合而成, 组合名为组合型[17,18](详见附录A)之名, 组合型之作用是, 组合已有函数或对象, 以构成新函数.

函数名亦可受限, 形如

M・f;

其中的 f 为函数名, M 为模块名.

6.3.4 对象

对象起数据作用.

对象: 基元, 序列, 底元(bottom), 异常名.

基元: 字符, 数, 真假值, 量子态.

序列:

$\langle O_1,\ldots,O_m\rangle$, O_i (i=1, …, m)为对象.

底元: 导致程序运行不终止, 迂此, 迫使程序挂起.

异常名: 语言定义, 用户定义(借助异常定义)

异常定义呈下形:

edef e = 串;

其中 e 为标识符, 表示所定义之异常名.

字符: 字母, 数字, 特定字符.

数: 整数, 实数, 复数.

真假值: **true**, **false.**

量子态: 形如

$$\alpha_1 |\varepsilon_{11}\ldots\varepsilon_{1m}> +\alpha_2 |\varepsilon_{21}\ldots\varepsilon_{2m}> +\ldots+\alpha_n |\varepsilon_{n1}\ldots\varepsilon_{nm}>,$$

其中诸 $\varepsilon_x(x=11,\ldots,1m,21,\ldots,2m,\ldots,n1,\ldots,nm)$ 为 0 或 1, 诸 $\alpha_i(i=1,\ldots,n)$ 为复数, 且 $|\alpha_1|^2+\cdots+|\alpha_n|^2=1$.

6.3.5 作用

函数: 对象, 它表示函数作用于对象所得之结果对象.

6.3.6 值

值有正常值与异常值之分. 正常值为基元或项为正常值之对象构成之序列; 异常值为 bottom 及异常名.

6.3.7 类型

类型有语言定义之类型与用户定义之类型之分, 前者有字符型(**char**)、布尔型(**bool**)、整型(**int**)、实型(**real**)、复型(**complex**)、序列型(**seq**)、向量型(**vector**)、矩阵型(**matrix**)和量子型(**quantum**); 后者则借助下形之类型定义来定义.

tdef 类型名=类型.

6.3.8 输入输出

语言定义了如下的输入函数 in 与输出函数 out:

in(串), 其中参数“串”表示外设名或文件名, 单值问题由系统统一处理;

out(串, v), 其中参数“串”之含义同前, v 表示其中存放待输出之值的寄存器名.

6.3.9 异常

有两类异常, 即语言定义之异常与用户定义之异常, 于此不拟赘述.

6.4 示例

随机整数生成.

问题描述: 输入一个自然数 n, 利用 Toss 算法生成一个 0 与 n 之间的随机整数.

算法流程:

根据 n 生成 m, 使得 $2^{m-1} \leqslant n \leqslant 2^m$;

生成一个长度为 m 的序列: $\langle 1, 1, \ldots, 1\rangle$;

将所生成之序列的每个元素用 Toss 算法替换成 0 或 1 的随机数;

将产生的二进制序列转换为整数 n';

若 $n'<n$ 则输出 n', 否则, 重复 2~4 步.

程序:

```
module specification Main:
export main;
        main, random, tyRandom, mkSeqByLen, ceilingOfLog2, ceiling,
binSeq2Int, isBinary ,is0, is1, s1, s2, H, toss
main: 任一正常值
end Main:
module body Main;
      fdef main=out ∘ ["scr",id] ∘ random ∘ ~n;
      fdef random = s1 ∘(while (>=∘[s1,s2]) ([tryRandom, id] ∘s2)) • [~,id]
      fdef tryRandom=binSeq2Int ∘ ( all toss ) ∘ mkSeqByLen∘
ceilingOfLog2;
      fdef mkSeqByLen= s1 ∘ (while (<∘[s1∘length, s2])[append1∘[~1,s1],s2])
∘[~<>,id];
      fdef ceilingOfLog2=ceiling∘/∘[In,In∘~2];
      fdef ceiling= neg∘floor∘neg;
      fdef binSeq2Int=seqOf isBinary→~0^(+∘[s1, * ∘[2,s2]]) ∘reverse;
      fdef inBinary=or ∘ [is0, is1];
      fdef is0=curry= =0;
      fdef is1=curry= =1;
      fdef s1=curry= =select 1;
      fdef s2=curry= =select 2;
      fdef toss=Measure ∘H∘[~0,~|0⟩]
end Main;
```

7 结语

量子计算为一门被坚认的学科, 量子程序设计语言亦然;

迄今所知, QCL 和 QML 为两种颇具代表性之量子程序设计语言;

本文所论语言风范、程序结构、输入/输出, 以及异常设施等均为语言设计者所应考虑与

解决的问题. 笔者新近设计与实现之量子程序设计语言 NDQFP 只是我们在该领域中进行的第 2 次试验, 无疑会有缺陷, 诚恳希望读者不吝赐教, 笔者深知谢忱.

致谢 参加本项工作的还有徐明君、焦阳、吴楠、董青、朱晓瑞、颜仙乐、朱振兴等人. 北京大学信息科学与技术学院, 国防科学技术大学计算机学院, 中创软件工程公司, 苏州大学计算机学院, 南京大学计算机软件新技术国家重点实验室及计算机科学与技术系均对本项目给予支持, 在此一并深致谢忱.

参考文献

1 Feynman R P. Simulating physics with computers. Int J theor Phys, 1982, 21: 467—488 [DOI]
2 Deutsch D. Quantum theory, the Church-Turing principle and the universal quantum computer. P Roy Soc Lond A Mat, 1985, 400: 97—117
3 von Neumann J. Mathematical Foundations of Quantum Mechanics. Princeton: Princeton University Press. 1955
4 Nielson M A, Chuang I L. Quantum Computation and Quantum Information. Cambridge: Cambridge University Press. 2000
5 张效祥. 计算机科学技术百科全书. 第 2 版. 北京: 清华大学出版社. 2005
6 Xu J F, Chong C H, Yang F Q, et al. On the design, implementation and use of the systems programming language XCY. Information Processing 80, 1980, 10: 305—308
7 Knill E. Conventions for Quantum Pseudocode. LANL Report. LA-UR-96-2724. Los Alamos National Laboratory. June 1996
8 Zuliani P. Quantum Programming. Oxford: St Cross College University of Oxford. Trinity Term 2001
9 Ömer B. Structured quantum programming. Ph D Thesis. Institute for Theoretical Physics. Vienna: Technical University of Vienna, 2003.
10 Baker G D. QgoL: a system for simulating quantum computation: theory, implementation and insights. Honours Thesis. Macquarie: Department of Computing, Macquarie University, 1996, 10
11 Gratlage J J. QML: a functional quantum programming language. Ph D Thesis. Nottingham: University of Nottingham, 2006
12 徐家福, 杨芙清, 仲萃豪. 模块: 一种结构化的程序设计工具. 电子学报, 1982, 6: 6—10
13 徐家福. 试论程序设计语言的设计原则. 计算机技术, 1986, 1: 1—3
14 徐家福, 王志坚, 翟成祥. 对象式程序设计语言. 南京: 南京大学出版社, 1992
15 徐家福. 系统程序设计语言. 北京: 科学出版社, 1983. 6
16 Backus J. Can programming be liberated from the von Neumann style, a functional style and its algebra of programs? ACM, 1978, 21(8): 613—641
17 Backus J, Williams J H, Winners E L, et al. FL Language Manual. Parts 1 and 2. IBM Almaton Research Center, 1989. 10
18 徐家福, 宋方敏, 钱士钧, 等. 量子程序设计语言 NDQJava. 软件学报, 2008, 1: 1—8

附录 A

1. 原始函数

原始函数乃本语言之基本成分, 在定义原始函数集时, 考虑到满足书写经典程序与量子程序之需

求, 同时亦适当考虑便于程序人员使用.

1) 谓词类

ⅰ) 测整 Isint **anytype→bool** 其中的 **anytype** 表示本语言中所定义之任何类型.

ⅱ) 测实 Isreal **anytype→bool**

ⅲ) 测复 Iscomplex **anytype→bool**

ⅳ) 测序列 Isseq **anytype→bool**

ⅴ) 测字符串 Isstring **anytype→bool**

ⅵ) 测布尔 Isbool **anytype→bool**

ⅶ) 测字符 Ischar **anytype→bool**

ⅷ) 测矩阵 Ismatrix **anytype→bool**

2) 关系类

ⅰ) 等 = = (**anytype**, **anytype**)→**bool**, 其中 **anytype** 表示任何经典类型, 但这里的 2 个 **anytype** 需表示同一类型.

ⅱ) 不等 ! = (**anytype**, **anytype**)→**bool**, 其中 **anytype** 之义同前, 但这里的 2 个 **anytype** 需表示同一类型.

ⅲ) 小于 < (**real**, **real**)→**bool**

ⅳ) 不大于 <= (**real**, **real**)→**bool**

ⅴ) 大于 >(**real**, **real**)→**bool**

ⅵ) 不小于 >= (**real**, **real**)→**bool**

3) 逻辑类

ⅰ) 与 and (**bool**, **bool**) →**bool**

ⅱ) 或 or (**bool**, **bool**) →**bool**

ⅲ) 非 not (**bool**, **bool**) →**bool**

4) 序列类

ⅰ) 取首 head **seq** →**seq** head: $\langle x_1,\ldots, x_n\rangle=\langle x_1\rangle$, if $n\neq 0$

head: $\langle\rangle=\langle\rangle$

ⅱ) 取尾 tail **seq** →**seq** tail: $\langle x_1, \ldots, x_n\rangle=\langle x_2,\ldots, x_n\rangle$ if $n>1$,

tail: $\langle x_1\rangle=\langle\rangle$

tail: $\langle\rangle=\langle\rangle$

ⅲ) 左补 appendl (**anytype**, **seq**)→**seq**

appendl: $\langle x_1,\langle y_1, \ldots, y_n\rangle\rangle=\langle x_1, y_1, \ldots, y_n\rangle$

ⅳ) 右补 appendr (**seq**, **anytype**) →**seq**

appendr: $\langle\langle x_1, \ldots, x_n\rangle, y\rangle=\langle x_1, \ldots, x_n, y\rangle$

ⅴ) 左分配 distl (**anytype**, **seq**)→**seq**

distl: $\langle x_1,\langle y_1, \ldots, y_n\rangle\rangle=\langle\langle x, y_1\rangle, \ldots,\langle x, y_n\rangle\rangle$

ⅵ) 右分配 distr (**seq**, **anytype**)→**seq**

distr: $\langle\langle x_1,\ldots, x_n\rangle, y\rangle=\langle\langle x_1, y\rangle,\ldots,\langle x_n, y\rangle\rangle$

ⅶ) 并置 cat (**seq**, **seq**) →**seq**

cat: $\langle\langle x_1,\ldots, x_m\rangle, \langle y_1,\ldots, y_n\rangle\rangle=\langle x_1,\ldots, x_m, y_1,\ldots, y_n\rangle$

viii) 长度 len **seq→int**

len: $\langle x_1,\ldots, x_n \rangle=n$

ix) 逆转 reverse **seq →seq**

reverse: $\langle x_1,\ldots, x_n\rangle=\langle x_n,\ldots, x_1\rangle$

x) 转置 trans **seq →seq**

trans: $\langle\langle x_{11},\ldots, x_{1n} \rangle\langle x_{21},\ldots, x_{2n} \rangle,\ldots,\langle x_{m1},\ldots, x_{mn}\rangle\rangle$

$=\langle\langle x_{11}, x_{21},\ldots, x_{m1} \rangle,\ldots,\langle x_{1n}, x_{2n},\ldots,x_{mn}\rangle\rangle$

xi) 选项 select (**int**, **seq**)→相应序列项类型

select : $\langle i, \langle x_1,\ldots, x_n\rangle\rangle=x_i$

xii) 建递增整列 intsto **int→seq**

intsto: $n=\langle 1,\ldots,n\rangle$

5) 算术类

i) 加 + (**datatype**, **datatype**)→**datatype**

ii) 减 – (**datatype**, **datatype**)→**datatype**

iii) 乘 * (**datatype**, **datatype**)→**datatype**

其中的 **datatype** 可为整、实、复、向量, 矩阵等类型, 但"→"左右方之三种类型需一致, 其中"→"左方之二类型可为复型与复型, 矩阵型与矩阵型, 复型与矩阵型或矩阵型与复型, →右方分别为复型、矩阵型、矩阵型.

iv) 除/(**datatype**, **datatype**)→**datatype**

其中的 **datatype** 只能是整型、实型与复型.

v) 取下表 floor **real→int**

vi) 取上表 ceiling **real→int**

vii) 取模 abs **real→real** (对实型数据取绝对值)

complex→real (对复型数据取模)

viii) 取幂 pow(**complex**, **real**)→**complex**

ix) 指数 exp **real→real** e 为底之指数函数.

x) 自然对数 ln **real→real**

xi) 取实部 getreal **complex→real**

xii) 取虚部 getimg **complex→real**

xiii) 取共轭 conjug **complex→complex**

6) 输入输出类

i) 输入 in **string→anytype**

anytype 之含义同前.

ii) 输出 out **anytype→(string**, **anytype)**

iii) 其中 anytype 之含义同前, 但→左, 右两方需一致.

7) 其他经典函数

i) 等同 id **anytype→anytype**

其中的 **anytype** 表示任意经典类型, →左右两方之类型需一致.

ii) 引发 raise **exception→exception**

iii) 字符型转整型 chartoint

char→int

iv) 整型转字符型 inttochar

int→char

8) 量子类

i) 判是否量子类型 IsQuantum

anytype→bool

ii) 量子初化 QuInit

(**complex**, …, **complex**)→**quantum**

iii) 量子态张量积⊗

(**quantum**, **quantum**)→**quantum**

iv) 全测量 measure

quantum→vector

v) 部分测量 measure At

(**int**, **quantum**)→**vector**

vi) 等同门 I

(**int**, **quantum**)→**quantum**

vii) 非门 X

(**int**, **quantum**)→**quantum**

viii) Hadamard 门 H

(**int**, **quantum**)→**quantum**

ix) 受控非门 CNot

(**int**, **int**, **quantum**)→**quantum**

x) 旋转门 Rot

(**int**, **real**, **real**, **quantum**)→**quantum**

xi) 相位门 Phase

(**int**, **real**, **quantum**)→**quantum**

2. 组合型

组合型的作用是组合现有函数或对象以构成新函数.

本语言提供之组合型集如下:

1) 组合: $(f\circ g):x=f:(g:x)$

2) 构作: $[f_1, \ldots, f_n]:x=\langle f_1:x,\ldots, f_n:x\rangle$

3) 条件: $(p\rightarrow f;g):x=\begin{cases} f:x, 若\ p:x=\textbf{true};\\ g:x, 若\ p:x=\textbf{false};\\ \text{exception, 否则,}\end{cases}$

其中的 exception 意指 exception 集中某一元素或用户定义之异常名.

4) 常值

$$\sim x:y=\left\{\begin{array}{l} x, 若y 为任何正常值;\\ \text{exception, 否则.}\end{array}\right\}$$

其中的 exception 之含义同前.

5) 插入 $\wedge$f

$$\wedge \mathrm{f}:\langle x_1,\ldots,x_n\rangle=\mathrm{f}:\langle x_1,\wedge\mathrm{f}:\langle x_2,\ldots,x_n\rangle\rangle$$

6) 全用 **all** f

$$(\text{all f}):\langle x_1,\ldots,x_n\rangle=\langle \mathrm{f}:x_1,\ldots,\mathrm{f}:x_n\rangle$$

7) 当函数 (**while** *p* f)

$$(\textbf{while}\ p\mathrm{f}):x=\left\{\begin{array}{l}(\textbf{while}\ p\mathrm{f}):(\mathrm{f}:x),\text{若}p:x=\textbf{true};\\ x,\text{若}p{:}x{=}\textbf{false};\\ \text{exception, 否则 .}\end{array}\right\}$$

8) 打并对象 **curry**

$$(\textbf{curry}\,\mathrm{f}x):y=\mathrm{f}:\langle x,y\rangle$$

9) 谓词构作 **pcons**

$$\textbf{pcons}:\langle \mathrm{f}_1\cdots\mathrm{f}_n\rangle:x=\left\{\begin{array}{l}\textbf{true},\text{若}\,x=\langle x_1,\ldots,x_n\rangle\text{且}\\ \mathrm{f}_i:x_i=\textbf{true},(i=1,\ldots,n);\\ \textbf{false},\text{ 否则.}\end{array}\right\}$$

10) ……序列 seqof

$$\text{seqof}:p:x=\left\{\begin{array}{l}\textbf{true},\text{若 }p\text{为函数, 且}x=\langle x_1,\ldots,x_n\rangle,\\ p:x=\textbf{true}(i=1,\ldots,n);\\ \textbf{false},\text{ 否则.}\end{array}\right\}$$

11) 异常时

(body when Exc handler): *x*

先执行 body: *x*, 如产生异常, 则转至 handler: *x*.

12) 酉构作 (uCons umatrix): *x*

由酉矩阵构作酉操作.

13) 酉张量 $(\mathrm{f}\otimes\mathrm{g}):x$

由酉操作张量积构作新的酉操作.