中国科学: 信息科学　2017 年　第 47 卷　第 10 期: 1300–1315
SCIENTIA SINICA Informationis　中国科学院学部　科学与技术前沿论坛
《中国科学》杂志社 SCIENCE CHINA PRESS
量子计算专题 · 论文

$Q|SI\rangle$: 一个量子程序设计环境

刘树森[1,4*], 周立[2], 官极[1], 贺旸[1], 段润尧[1], 应明生[1,3,2*]

1. Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney 2007, Australia
2. 清华大学计算机科学与技术系, 北京 100084, 中国
3. 中国科学院软件研究所计算机科学国家重点实验室, 北京 100190, 中国
4. 中山大学数据科学与计算机学院, 广州 516000, 中国
* 通信作者. E-mail: shusen88.liu@gmail.com, Mingsheng.Ying@uts.edu.au

收稿日期: 2017–06–04; 接受日期: 2017–06–20; 网络出版日期: 2017–10–13

摘要　本文介绍了作者设计并实现的一个量子程序设计环境: $Q|SI\rangle$. $Q|SI\rangle$ 是基于.net 语言开发的支持量子 **while** 扩展语言的平台环境, 它包括了一个支持量子 **while** 语言的编译器和一套支持量子计算模拟、量子电路优化、量子程序分析和量子程序验证的工具. 阐述了如何在经典计算机上通过平台的各个组件和概率算法来模拟量子行为. 平台可扩展的架构允许用户在平台上自定义新的功能. 同时, 编译器是 $Q|SI\rangle$ 的核心, 它内置的分解算法扩展了现阶段的量子硬件功能, 使通用量子计算成为了可能, 成为连接量子硬件和量子软件的桥梁.

关键词　量子程序, 量子编译, 量子模拟, 量子程序分析, 量子程序验证

1　引言

众所周知, 量子计算机相比经典计算机可以更有效地解决一大类计算问题, 著名的例子包括 Shor 大数分解算法 [1]、Grover 搜索算法 [2] 和最近由 Harrow, Hassidim 和 Lloyd 提出的线性方程组求解算法 [3] 等. 近些年, 各国政府和工业界在量子计算机研制中竞争十分激烈. 量子计算机发展如同经典计算机的情形, 在商业化的进程中, 程序员自然需要一个可以表达量子算法和实现量子算法的平台, 这个平台应该可以让程序员摒弃量子程序所对应量子电路实现中的细枝末节. 实际上, 相比于经典计算机的程序设计, 量子计算机程序设计更需要这样的平台. 使用这样的平台, 程序员不必考虑太多量子算法的物理实现, 因此可以 (部分) 避免程序员对于量子系统的违反直觉的一些特征的误解及带来的负面影响.

过去的二十年里, 人们已经发展了一些量子编程平台. 1998 年, Ömer 在 C++ 平台上实现了第一个量子程序设计语言 QCL [4,5]. 与此类似, Bettelli 等 [6] 于 2003 年在 C++ 基础上实现了一个量子

引用格式: 刘树森, 周立, 官极, 等. $Q|SI\rangle$: 一个量子程序设计环境. 中国科学: 信息科学, 2017, 47: 1300–1315, doi: 10.1360/N112017-00095
Liu S S, Zhou L, Guan J, et al. $Q|SI\rangle$: a quantum programming environment (in Chinese). Sci Sin Inform, 2017, 47: 1300–1315, doi: 10.1360/N112017-00095

© 2017《中国科学》杂志社

语言的类库, 命名为 Q 语言. Sanders 和 Zuliani [7] 在 2000 年左右引入了 qGCL 语言, 它是 Dijkstra 的 GCL (guarded-command language) 和 pGCL (probabilistic extension of GCL) 的量子扩充版本. 近几年, 一些可扩展的、更具鲁棒性的量子语言逐渐出现. 比如在 2013 年, Green 等 [8] 提出了为量子计算服务的可扩展的函数式内嵌编程语言 Quipper, 它使用 Haskell 作为宿主语言; 2014 年, 来自微软量子系统结构及计算研究组 QuArc (microsoft research quantum architecture and computation team) 的 Wecker 和 Svore [9] 设计并发布了一套现代化的嵌入在 F# 语言上的量子软件开发工具 LIQU$i|\rangle$; 同年, JavadiAbhari 等 [10,11] 开发了量子编译软件 ScaffCC. 2016 年, Intel 的 Smelyanskiy 等 [12] 实现了一个并行量子计算模拟器 qHiPSTER, 它在超级计算机上以极高的性能模拟了高达 40 个量子 Bit 的计算.

本文设计并编写了一个新的量子程序设计环境 $Q|SI\rangle$. $Q|SI\rangle$ 的命名是寓意将量子力学 (Q, quantum mechanics) 应用于软件 (S, software) 和信息技术 (I, information), 同时, 它还是部分作者所在实验中心的名称[1)]. $Q|SI\rangle$ 的核心是一套量子编程语言和它的编译器. 这个语言是经典的 **while** 程序语言在量子层面上的深度扩展. 它的语义和语法最早在文献 [13] (或 [14], 第 3 章) 中进行了研究. 特别地, 它包含了一个基于测量的选择语句和一个基于测量的循环语句. 这两种程序结构对描述一大类量子程序 (例如基于量子随机游走算法的程序) 来说是很方便的工具.

为了与量子硬件结合, 本文基于 QASM (quantum assembly language) [15] 定义了一个新的汇编语言 f-QASM (quantum assembly language with feedback) 作为底层交互语言. 在这个语言里引入了一个包含反馈的指令集合, 使之更有效地描绘基于测量的条件语句和循环语句. 平台的编译器会把量子程序编译到一系列的 f-QASM 指令集上, 依据指令集进一步生成等价的可执行的量子电路 (一系列可执行的量子门). $Q|SI\rangle$ 包含了优化量子程序的模块, 同时包含了一个模拟器模块用来在经典计算机上模拟检测量子计算机计算的结果. 进一步地, $Q|SI\rangle$ 包含了两个其他的编程环境所没有的新功能.

(1) 量子程序分析器. 本文的作者之一在文献 [16,17] 中给出了量子 **while** 程序的终止性分析算法和估计量子 **while** 程序平均运行时间的算法. 除此之外, 他还利用半正定规划 (semi-definite programming) 来生成量子 **while** 程序的不变量 [18]. 这些算法在 $Q|SI\rangle$ 中用来做量子程序的静态分析, 其分析的结果可以帮助编译器进一步优化量子程序.

(2) 量子程序验证器. 本文作者之一在文献 [13] (或 [14], 第 4 章) 中建立了量子程序的 Floyd-Hoare 型逻辑用来验证量子程序的正确性. 最近, Liu 等 [19] 在 Isabelle/HOL 之上实现了基于量子 Floyd-Hoare 型逻辑的证明器. 我们希望将来能够将文献 [19] 等所述的定理证明器连接到 $Q|SI\rangle$ 平台上用于量子程序验证.

2 量子力学 4 个基本假设

考虑到本文的大部分读者来自计算机方向, 他们可能不熟悉量子力学. 为了帮助读者更好地理解本文的内容, 我们首先简要回顾量子力学的基本假设. 在量子力学中有 4 个基本假设, 这些假设定义了量子行为 [20], 并导出了量子其他性质的刻画. 模拟量子行为本质上来说就是模拟这 4 个基本假设, 详细的设计将在第 6 节给出.

- **基本假设 1.** 与任何孤立的物理系统相关联的是具有内积的复向量空间 (Hilbert 空间), 这个空间称为系统的状态空间. 该系统被其状态向量完全描述, 而此向量是系统状态空间中的单位向量.
- **基本假设 2.** 封闭量子系统的演化被酉变换完全描述. 也就是说, 系统在 t_1 时间的状态 $|\psi\rangle$ 与系

1) http://www.qsi.uts.edu.au.

统在 t_2 时间的状态 $|\psi'\rangle$ 之间的联系由一个只依赖于时间 t_1 和 t_2 的酉算子 U 描述.

• **基本假设 3.** 一个量子测量被一组测量算子 $\{M_m\}$ 所描述. 这些算子作用在被测量的系统上. 索引 m 表示了试验中的测量所产生的可能的结果. 如果系统状态在被测量前为 $|\psi\rangle$, 那么得到结果 m 的概率是 $p(m)=\langle\psi|M_m^\dagger M_m|\psi\rangle$, 测量后系统的状态为 $\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$.

• **基本假设 4.** 复合量子系统的状态空间是每个子物理系统状态空间的张量积. 如果有一个由编号为 1 到 n 的子系统构成的复合系统, 其中标号为 i 的子系统状态为 $|\psi_i\rangle$, 那么整个复合系统的状态就是 $|\psi_1\rangle\otimes\cdots\otimes|\psi_i\rangle\cdots\otimes|\psi_n\rangle$. 这样的状态被称为乘积态. 进一步地, 乘积态的线性叠加会产生纠缠态.

3 量子 while 语言

为了帮助读者更好地理解量子程序设计环境 $Q|SI\rangle$, 首先简要介绍一下其中的量子 **while** 语言. 这个量子程序设计语言是一个无经典变量的纯量子语言, 它假定了只存在一组标记为 $q_0, q_1, q_2, \ldots$ 的量子变量. 但是, 现在几乎所有的量子算法都同时包含了经典计算部分和量子计算部分. 因此, 从实现上来说, 量子语言必须和经典语言一起使用. 在 $Q|SI\rangle$ 中, 量子 **while** 语言被嵌入到了 C# 语言中, 这带来了程序设计上的便利性. 现在, 让我们解释一下量子程序的一些结构. 相关细节上的描述或者例子, 读者可以参考文献 [13,14] 的第 3 章.

量子 **while** 语言由以下几个简单的句法生成:

$$\begin{aligned}\mathbf{S} ::= \mathbf{skip} \mid q:=|0\rangle \mid \bar{q}=U[\bar{q}] \mid S_1;S_2 \mid &\mathbf{if}\ (\square m\cdot M[\bar{q}]=m\to S_m)\ \mathbf{fi}\\ \mid &\mathbf{while}\ M[\bar{q}]=1\ \mathbf{do}\,\mathbf{S}\,\mathbf{od}.\end{aligned}$$

Skip. 如同经典程序设计语言一样, **skip** 语句什么都不做, 仅仅是立刻终止当前语句.

初始化. 初始化语句 "$q:=|0\rangle$" 设立变量 q, 并初始化量子态为 $|0\rangle$.

酉变换. $\bar{q}$ 表示量子寄存器, 语句 "$\bar{q}:=U[\bar{q}]$" 意味着一个酉变换 U (量子门) 作用在量子寄存器 $\bar{q}$ 上, 其他的部分做恒等变换.

顺序复合. 如同经典程序一样, 在复合语句 $S_1;S_2$ 中, 首先执行 S_1, 结束后再执行 S_2.

条件语句. 在条件语句 **if** $(\square m\cdot M[\bar{q}]=m\to S_m)$ **fi** 中, M 是一个量子测量, m 代表了它可能的结果. 为了执行这个语句, M 首先作用在量子寄存器 $\bar{q}$ 上, 并以一定的概率得到一个测量结果 m. 接下来子程序 S_m 依据测量结果 m 而被选中执行. 与经典的条件语句不同的是, 量子条件语句的量子寄存器 $\bar{q}$ 经过测量后发生了改变.

while 循环语句. 在循环语句 **while** $M[\bar{q}]=1$ **do S od** 中, M 是一个只有两个可能结果的 "是 – 非" 测量: 结果为 0 或者 1. 在执行过程中, 首先检测循环条件, 在量子寄存器 $\bar{q}$ 上执行测量 M. 如果结果是 0, 程序结束; 如果结果为 1, 程序则执行循环体 S, 并继续执行. 需要注意的是量子寄存器 $\bar{q}$ 在执行测量 M 后, 其值也发生了改变.

4 $Q|SI\rangle$ 的结构

本节将介绍 $Q|SI\rangle$ 的基本结构, 并留下部分细节在后续章节中描述. $Q|SI\rangle$ 旨在提供一个支持量子 **while** 语言的通用程序设计环境. 它包括了一个用于量子 **while** 程序的编译器、量子计算的模拟器以及用于量子程序分析和验证的模块. 本文将 $Q|SI\rangle$ 实现为深度嵌入的特定域语言 (domain-specific language, DSL) 量子编程平台, 使用 C# 作为宿主语言.

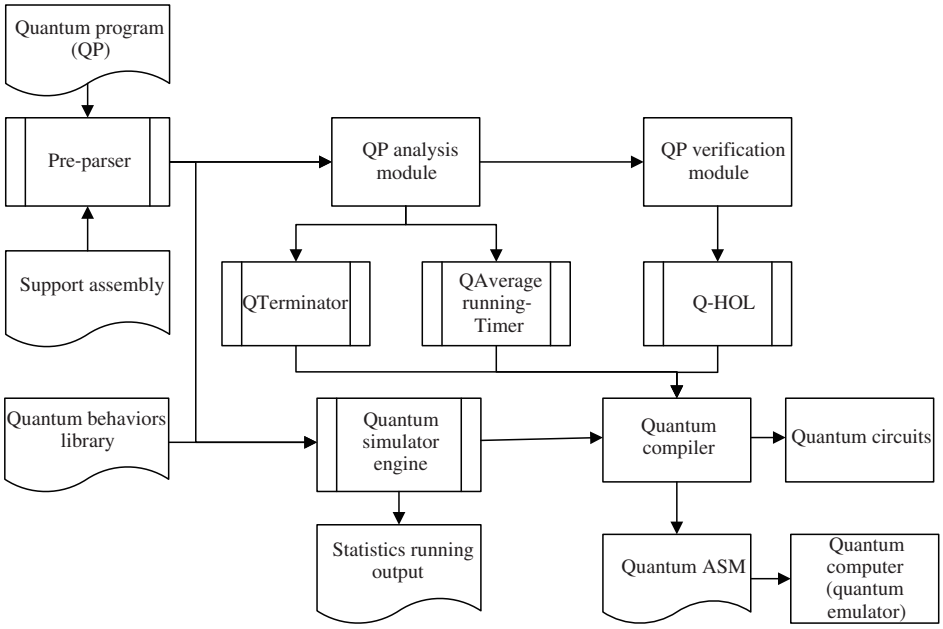


图 1 $Q|SI\rangle$ **架构**

Figure 1 $Q|SI\rangle$ framework

$Q|SI\rangle$ 的整个框架结构如图 1 所示, 下面将详细解释框架中的组件.

4.1 $Q|SI\rangle$ 的基本特征

$Q|SI\rangle$ 的基本特征如下.

语言支持. $Q|SI\rangle$ 是支持量子 **while** 语言的第一个平台. 特别是它允许使用基于测量的条件语句和 **while** 循环语句来编程. 这两个程序结构提供了更高效更细致的手段来描述一些量子算法, 如量子随机游走和 Grover 搜索等.

增强的量子类型. 相比于其他的量子模拟器和分析工具, $Q|SI\rangle$ 支持更多更灵活的量子类型, 这种灵活性使得其在不同的场景下能更好地发挥作用. 本文为不同的输入提供了量子 Bit 和密度矩阵等表示方法, 减轻了用户记忆和使用负担. 同时这些量子类型有着统一的操作接口, 这些使得量子对象的创建和操作高度灵活, 增强了量子类型的可用性、便利性.

两种执行模式. $Q|SI\rangle$ 有两种可执行模式. 一个是 "动态执行", 它通过模拟量子行为显示实验的结果, 另一个模式是 "静态执行", 主要用于量子程序编译、分析和验证.

指令集 f-QASM. QASM (quantum assembly language) [15] 是一组为了量子计算机设计的低级指令集. 本文定义了 QASM 的一个扩展, 使之可以适配不同目的高层次量子电路描述语言. 在这种语言中, 每行只包括一条指令. 与原始的 QASM [15] 和带空间压缩的 QASM-HL [10] 相比, f-QASM 使用了 "goto" 结构, 包含更多的信息. f-QASM 也可以用于程序的进一步优化和分析.

量子电路生成. 与经典计算中的现代数字电路类似, 量子电路是量子算法的低级表示 [15]. $Q|SI\rangle$ 的编译器可以将高级量子 **while** 语言编写的程序生成对应的量子电路.

任意酉算子实现. 在 $Q|SI\rangle$ 平台, 本文实现了二级矩阵分解算法 [20] 和量子选择器 (quantum multiplexor, QMUX) 算法 [21] 以及 Solovay-Kitaev 算法 [22]. 这个功能可以使未来的量子芯片制造商提供的小的预定义的基本门集合, 组成量子电路中所需要的任意旋转门.

4.2 $Q|SI\rangle$ 的主要构件

平台主要由 4 部分组成.

量子模拟引擎 (quantum simulator engine). 量子模拟组件主要包括了支撑集合 (pre-parser and support assembly), 量子力学库 (quantum behaviors library) 和计算引擎 (quantum simulator engine). 支撑集合提供了对量子类型和量子语言的支持. 准确的说, 它提供了一系列量子对象和支持再重入的包装函数来支持量子 **if** 和量子 **while** 程序结构. 量子力学库提供了量子对象的行为的定义和程序解释, 例如: 酉变换、测量的结果输出和测量后的态. 计算引擎被设计为一个可执行的引擎, 它接受量子对象和其在量子力学库里的行为准则作为输入, 然后将它们解释为相应的概率程序的执行步骤. 此部分的相关工作已经完成.

量子程序分析模块 (QP analysis module). 本模块提供了量子程序 (quantum program) 静态分析的相关信息和结果. 目前, 本模块提供了两个已经完成的静态分析子模块: "QTerminator" 和 "QAverage Running-Timer". 前者可提供量子程序终止性的相关信息, 后者可用于估计量子程序的平均运行时间. 它们的输出结果被发送给下一阶段的量子编译器来做进一步的分析处理.

量子程序验证模块 (QP verification module). 本模块为了验证量子程序的正确性而设计, 它依托于量子 Floyd-Hoare 型逻辑 [13]. 本模块仍在持续的建设中. 未来, 也可能连接 Liu 等 [19] 实现的量子证明器 (Q-HOL), 来完成量子程序的验证.

量子编译器 (quantum compiler). 编译器包括了一系列相关工具 (quantum circuits and quantum ASM), 它将表示量子算法的高级源程序映射到量子设备相关语言 [15], 例如, 映射到 f-QASM 或者进一步映射到量子电路. $Q|SI\rangle$ 的设计目标是使用户不考虑设备的细节实现任何源代码, 即基于源代码自动构造量子电路. 此部分已经完成量子汇编输出模块和单 Bit 量子电路绘制模块. 此外, 未来编译器中还将会被加入用来优化量子电路的工具.

4.3 $Q|SI\rangle$ 的实现

在平台实现中, 核心的问题是如何使用概率和经典算法来模拟量子行为. 为了支持更多量子操作, $Q|SI\rangle$ 丰富了量子模拟引擎的数据结构. 图 2 展示了量子模拟引擎的处理流程和方法.

在模拟引擎里, 支持了 3 种语言类型: 经典编程语言、纯量子 **while** 语言和混合语言. 当引擎检测到量子类型时, 启动一系列支持量子特性的处理流程. 引擎首先会检测每个变量和操作并执行对应的支撑集合. 如前所述, $Q|SI\rangle$ 的一个主要特征是支持量子 **while** 语言, 这个特点主要由量子支撑集合提供支持. 所有的量子行为, 在经典计算机上, 被概率算法解释执行. 预编译器输出为可以在 .net 框架执行的 C# 语言, 然后被内置编译器解释为 f-QASM 或者相应的量子电路.

量子模拟引擎通常包括了大量的矩阵计算和操作. 在 $Q|SI\rangle$ 中, Math.net 被用来做矩阵的一些计算分析. Math.net 是一个开源的涵盖基础数据类型和高等数学工具的.net 开发库, 它被科研人员和开发人员广泛的使用[2)]. 它包括了数值计算、计算机代数、信号处理和几何运算等. 当有众核体系构架时 (many integrated core architecture), Math.net 也可以被用来加速模拟计算.

在静态分析部分, 选择 Roslyn 作为主要的代码分析工具. Roslyn 是一套出色的开源的 C# 编译器. 因为平台内嵌在 .net 框架的 C# 语言里, 使用 Roslyn 作为分析器来对代码进行分析, 进一步完成构建抽象语法树 (abstract syntax tree, AST) 等工作.

2) https://www.mathdotnet.com.

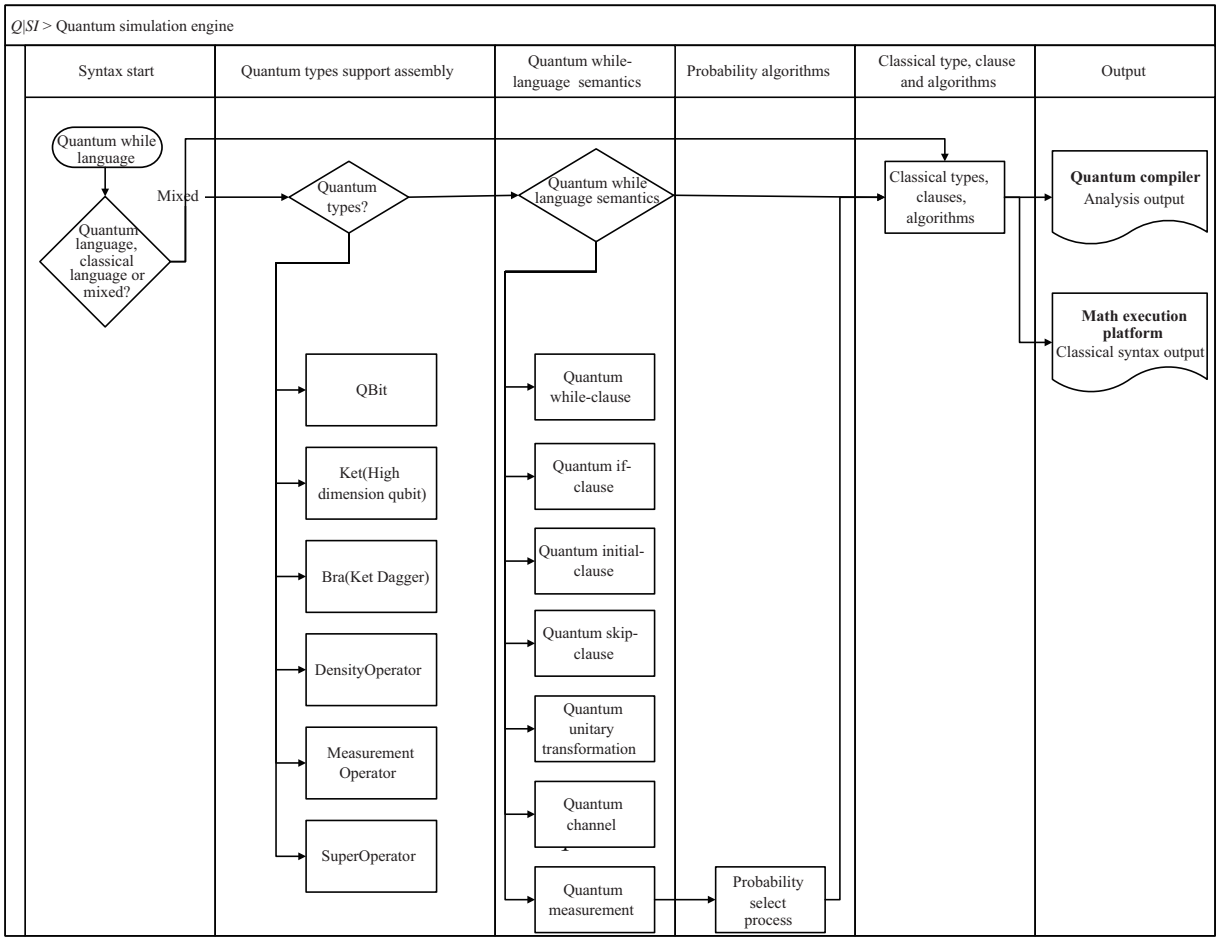


图 2 $Q|SI\rangle$ **预处理器**

Figure 2 $Q|SI\rangle$ Pre-parser

5 量子编译器

本节介绍 $Q|SI\rangle$ 的一个核心部分: 编译器. 作为与不同设备和数据结构的连接桥梁, 编译器有几个方面的应用. $Q|SI\rangle$ 中编译器的首要功能是生成 f-QASM, f-QASM 可以用于仿真实验或者运行在真实 (虚拟) 量子处理器上. 其次, 它能提供量子芯片设计所需要的量子电路. 此外, 它还提供了用于优化量子电路的模块.

量子编译器本身高度依赖于其他模块. 一方面, 它从量子模拟引擎收集程序的数据结构. 引擎初步处理之后, 程序已经被分成几个部分: 量子变量、酉算子、测量算子、每个子程序语句的入口点和出口点位置等. 接着, 编译器从程序的这些部分构造 AST. 生成 AST 之后, 它将程序重建为 f-QASM 指令序列以供进一步使用. 基于 f-QASM, 编译器提供了用于分解酉运算符的方法. 它可以将任意酉运算符 $U(n)$ 以任意给定的误差限 ϵ 分解到一系列基本量子门. 这些基本门来自一个预定义的集合 $\{U_1, U_2, \ldots, U_m\}$, 其中 $U_1, U_2, \ldots, U_m \in U(2)$ (即单量子 Bit 门) 或者是控制非门 (CNOT). 这对应于量子器件开发中的场景: 尽管生产商只能生产极少的几个量子门电路, 但是, 人们通常需要的是通用

计算. 另一方面, 量子 **while** 语言提供了循环的能力, 但同时也增加了编译程序的复杂性: 相比于没有循环的程序, 带有循环的量子程序更难被跟踪调试. 量子程序分析模块提供了静态分析工具, 包括用于终止性检查的 "QTerminator" 和用于计算期望运行时间的 "QAverage Running-Timer". 此外, 正在开发的量子程序验证模块将被设计用于量子程序的验证, 使得人们可以插入断言进一步地调试量子程序的行为.

5.1 f-QASM

QASM 被广泛应用于量子模拟器和量子计算机. 它在文献 [15] 中首先被提出, 并且被定义为通过基于量子电路模型描述的一组设备无关简化指令集. 文献 [23] 进一步研究了一些类似于 QASM 的指令的理论性质. 之后, JavadiAbhari 等 [10] 定义了一个松散平面描述 QASM 和一个被称为 QSAM-HL 的密集的分层描述结构 QASM. 最近, Smith 等 [24] 提出了一种用于经典、量子混合算法的 QASM, 并将其应用于 Quil. Quil 作为前端为 Forest 提供支持, 而 Forest 是学习量子编程开发和模拟的一套基于云计算的工具.

本文提出另一种称为 f-QASM 的扩展 QASM 格式. 一个动机是使程序的内在逻辑可以被转译为最简单的指令, 即 f-QASM 应当在每个行或周期中只有一个指令. 另一个动机是解决 IBM QASM 2.0 列表中提出来的一个问题: 如何定义实现基于测量结果反馈的操作的指令?

5.1.1 f-QASM 的基本定义

首先, 定义一些寄存器.

- 定义 $\{r_1, r_2, \ldots\}$ 为有限个经典寄存器.
- 定义 $\{q_1, q_2, \ldots\}$ 为有限个量子寄存器.
- 定义 $\{fr_1, fr_2, \ldots\}$ 为有限个标志位寄存器, 标志位寄存器 (flag register, fr) 是一类特殊的经典寄存器, 它通常用来表示一些运算结果, 无法被用户程序的代码直接修改.

然后, 定义两类基础操作.

- 定义指令 $q := op(q)$ 为 $op(q)$, 其中 op 是一个酉算子, q 是一个量子寄存器.
- 定义指令 $r := \{op\}(q)$ 为 $\{op\}(q)$, 其中 $\{op\}$ 是一组测量算子, q 是一个量子寄存器, r 是一个经典寄存器.

定义寄存器和操作后, 可以定义一些集合函数.

- 定义 INIT(q) 为 $q := |0\rangle\langle 0|$, 其中 q 是一个量子寄存器, 即 q 被赋值为 $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$.
- 定义 OP(q, num), 其中 q 是量子寄存器, num $\in \mathbb{N}$, OP 是运算符, 即 $q := op(q)$. 当 num 的值为 0 时, 表示当前运算符在预定义的基本量子门集合中, 可以由制造商或用户准备, 否则只能在分解成基本门之后使用, 或被忽略其实现意义.
- 定义 MOV(r_1, r_2): 将 r_2 的值赋给 r_1, 并清空 r_2 的值, 其中 r_1, r_2 是经典寄存器.
- 定义 CMP(r_1, r_2): $fr_1 = \delta(r_1, r_2)$ 或 $fr_1 = (r_1 == r_2)$, 其中 r_1, r_2 是两个经典寄存器, δ 是对比 r_1 和 r_2 的函数: 如果 r_1 的值和 r_2 相等, 那么 $fr_1 = 1$; 否则 $fr_1 = 0$.
- 定义 JMP l_0: 当前指令跳跃至 l_0 行处.
- 定义 JE l_0: 查询 fr_1 寄存器的内容并进行跳转, 即如果 fr_1 的值为 1 那么就执行 JMP l_0, 否则什么也不做.

5.1.2 f-QASM 例子

为了帮助读者理解 f-QASM, 本小节给出一些简单的例子.

(1) 初始化. $\mathbf{q} := |\mathbf{0}\rangle$ 意味着初始化量子寄存器 q 为量子态 $|0\rangle$. 使用 f-QASM 来格式化两个量子寄存器 $q1$, $q2$ 为量子态 $|0\rangle$, 可以写为

```
INIT(q1);
INIT(q2);
```

(2) 酉变换. $\bar{\mathbf{q}} := \mathbf{U}[\bar{\mathbf{q}}]$ 代表着一个酉变换作用在寄存器 $\bar{q}$ 上. 编译器检查酉变换对应的量子门是否是一个基本的门电路, 即是否是一个生产商可以提供的门电路. 例如, 如下的程序片段:

```
hGate(q1);
```

假设 $hGate$ 是单量子 Bit Hadamard 门, 即 $hGate = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$, 它可以被转义为 f-QASM 指令

```
hGate(q1, 0);
```

(3) 条件语句. 如下是一个用量子 **while** 语言书写的条件语句:

```
QIf(M(q1)
() =>
{
xGate(q1);
},
() =>
{
hGate(q1);
}
);
zGate(q1);
```

其中 $hGate$ 是单量子 Bit Hadamard 门, $xGate$ 是单量子 Bit 翻转门, 即 $xGate = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$, $zGate$ 是相位翻转门, 即 $zGate = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$, 且假设它们为基本的门. M 是一个用户定义的测量. 编译器会将这段代码翻译为 f-QASM 的指令集:

```
MOV(r,{M}(q1));
CMP(r,0);
JE L1;
CMP(r,1);
JE L2;
L1:
xGate(q1,0);
JMP L3;
L2:
```

```
hGate(q1,0)
JMP L3;
L3:
zGate(q1,0);
```

(4) 循环. 循环结构在 $Q|SI\rangle$ 里用 $QWhile(M(q))$ 表示, 其中 $QWhile$ 是一个关键词, M 是一个测量, q 是一个量子变量. 例如, 如下是一个包含量子循环语句的代码片段:

```
QWhile(M(q1),
() =>
{
xGate(q1);
}
);
hGate(q1);
```

假设 $hGate$ 和 $xGate$ 是如前所述的基本的门, M 为一个用户定义的测量, 它可以被转义为 f-QASM 指令集:

```
L1:
MOV(r,{M}(q1));
CMP(r,0);
JE L2;
xGate(q1,0);
JMP L1;
L2:
hGate(q1,0);
```

5.2 酉算子分解算法

编译器中, 本文同时内建了两个分解算法, 一个是文献 [20,25] 提到的 QR 分解算法, 它包括了如下几个步骤:

(1) 将任意的酉算子分解为一组酉算子的序列, 该序列中的每个算子只能非平凡地作用在由两个计算基扩展的子空间上.

(2) 序列中, 每个算子将被进一步地分解为单 Bit 量子门 (即 $U(2)$ 中的算子) 和控制非 (CNOT) 门.

(3) 每个单 Bit 量子门可以进一步地使用 Solovay-Kitaev 算法 [22] 在给定的误差限 ϵ 内分解为一系列的给定的基本门 (一些特殊的单 Bit 门).

另外一个算法是文献 [21] 中的 QSD 分解方法, 主要包括了以下几步:

(1) 将任意 $U(2^d)$ 算子分解为 3 个量子选择器的旋转门 (quantum multiplexor, QMUX) 和 4 个一般的 $U(2^{d-1})$ 算子, 其中 d 是量子 Bit 的个数.

(2) 重复的执行第一步, 直到得到 $U(4)$ 算子.

(3) 将 $U(4)$ 的算子分解为 $U(2)$ 的算子和两个额外的控制非 (CNOT) 门.

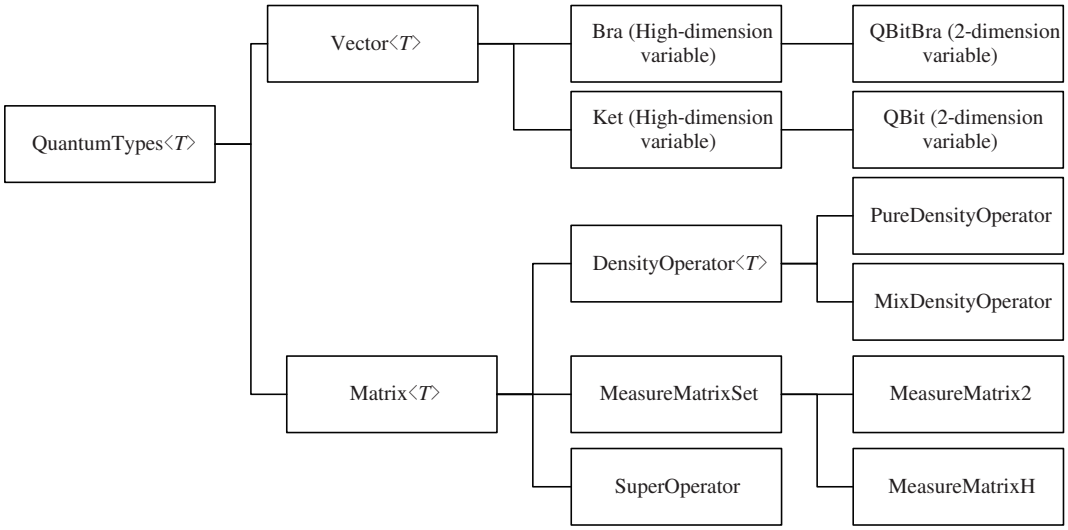


图 3 $Q|SI\rangle$ **量子变量类型**

Figure 3 $Q|SI\rangle$ Quantum types layer

(4) 每个单量子 Bit 门 ($U(2)$ 中的算子) 使用 Solovay-Kitaev 算法 [22] 在给定的误差限 ϵ 内进一步地分解为一系列给定的基本门 (单 Bit 门).

6 量子模拟器

本节介绍 $Q|SI\rangle$ 的另一个核心部分: 模拟器.

6.1 量子类型

本文将经典数据类型结构进行了一些量子化的扩展. 例如, 量子版的 Boolean 类型和整型 [13]: 量子 Boolean 类型的量子态所在的空间是 2 维的 Hilbert 空间即 $\mathbf{Boolean} = \mathcal{H}_2$, 量子整型的量子态是无限维的 Hilbert 空间即 $\mathbf{Integer} = \mathcal{H}_\infty$. 在 $Q|SI\rangle$ 中, 这两种量子类型的变量, 都有自己相应的初始化方法和操作. 目前, $Q|SI\rangle$ 只支持了有限维的量子变量, 但在以后的开发计划中, 无限维的变量也会被置入开发议程. $Q|SI\rangle$ 所用的量子类型和其依赖关系如图 3 所示.

全部的量子态被定义为一个名为 $QuantumTypes\langle T\rangle$ 虚基类的子类. 引入虚基类是为了预编译器能及时的区分出来量子对象和非量子对象, 方便后面的处理. 有两个子类继承于此虚基类: $Vector\langle T\rangle$ 代表了一类共享一系列向量规则的量子对象, $Matrix\langle T\rangle$ 代表了一类共享一些算子规则的量子对象.

量子变量中有两个基本类型: *Ket* 用来标记一个任意维度的量子变量, *Bra* 被用来标记 *Ket* 的转置共轭. 特别地, 本文将两个特殊的 (子) 类 *QBit* 和 *QBitBra* 作为特殊的二维量子变量提供给用户. 需要指出的是, 当 Boolean 型是一个整型的特例时, 它们实际上是完全兼容的. 除此之外, 这些类型必须有一些额外的约束, 如下所示.

单位化量子态. 例如, 一个量子 Bit 可以写做 $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. 如果用计算基 $\{|0\rangle, |1\rangle\}$ 去测量它, 以 $|\alpha|^2$ 的概率得到 0, 以 $|\beta|^2$ 概率得到 1. 由于测量的概率之和必须等于 1, 因此, 它必须遵守 $|\alpha|^2 + |\beta|^2 = 1$ 的规则. 所以, 向量必须在初始化和计算中保持长度为 1. 为了方便, 在量子模拟的底层, 对 *QBit* 和 *Ket* 提供了一个内置函数 *QBit.NormlizeSelf*(), 来保持量子变量的单位长度.

隐藏的状态. 众所周知, *QBit* 或者 *Ket* 里面的量子状态除了测量以外, 是不应该被直接提取的. 另外, 如 Nielsen 与 Chuang 在文献 [20] 中所说, "对封闭的量子系统来说, 自然演化不用通过测量, 它就似乎已经洞悉了一切, 包括所有的连续变量的值, 例如一个量子态的 α 和 β ". 在平台 $Q|SI\rangle$ 中, 本文使用以下技巧来模拟量子计算: 量子状态就像一个黑盒子, 盒子中的每个部分与其他部分密切配合, 但是外部的观察者任何信息都得不到, 同时包括酉变换或量子信道在内的函数和其他对象方法都可以精确地了解量子态. 除非测量, 否则观察者无法获取这些隐藏的信息. 因此, 本文将状态信息存储分为 "*Protect*" 类型, 这意味着量子态的信息不能轻易地被获取.

矩阵形式广泛应用于量子 **while** 语言的 (语义) 表达. 它们分为 3 种: $DensityOperator\langle T\rangle$, *MeasureMatrixSet* 和 *SuperOperator*. $DensityOperator\langle T\rangle$ 也有两个子类: *PureDensityOperator* 和 *MixDensityOperator*. 事实上, *PureDensityOperator* 和 *MixDensityOperator* 之间只有微小的区别 —— 只有 *MixDensityOperator* 接受一个系综, 即一组概率集合和可以由 *PureDensityOperator* 或 $Vector\langle T\rangle$ 表述的量子态. $DensityOperator\langle T\rangle$ 的对象量子态 ρ 需要满足以下两个条件: (1) ρ 的迹为 1; (2) ρ 是一个正算子. $Q|SI\rangle$ 中设置量子对象的每个操作都会触发这些验证条件, 以便检查对象是否是真实的密度算子. *MeasureMatrixSet* 是一个测量集合, 它包含了一组算子 $M = \{M_0, M_1, \ldots, M_n\}$ 并满足条件 $\sum_i M_i^\dagger M_i = I$. 用这种方式定义的量子测量是非常灵活的, 可以轻易地定义出包括对角基测量 $\{|+\rangle, |-\rangle\}$ (其中 $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$) 和计算基测量 $\{|0\rangle, |1\rangle\}$ 等测量算子集合, 用户还可以很容易实现自定义的测量方法. *SuperOperator* 经常被用来模拟开放量子系统. 在 $Q|SI\rangle$ 中, 使用一组 Kraus 算子 $\mathcal{E} = \{E_0, E_1, \ldots, E_n\}$ 来描述开放量子系统或者量子信道, 这些算子满足 $\sum_i E_i^\dagger E_i \leqslant I$ 的条件.

6.2 模拟量子行为

量子计算模拟的基础是模拟量子力学的 4 个基本假设, 这些假设定义了量子行为 [20].

- **基本假设 1.** 在平台 $Q|SI\rangle$ 上, 由 Math.net 提供的函数

$$\text{double } ConjugateDotProduct(Vector\langle T\rangle other),$$

来支持内积运算.

- **基本假设 2.** 为了在平台上模拟这个特性, 本文在一些底层支持集里的量子对象上, 如 *QBit*, *Ket* 和 $DensityOperator\langle T\rangle$ 加入了函数 $UnitaryTrans$ 来描述酉变换. 在全局系统上, 提供了一个静态函数 *SuperMatrixTrans* 来描述开放量子系统的动态过程.

- **基本假设 3.** 量子测量过程被一个修改后的 Monte Carlo 算法模拟执行. 详细的过程在下一小节解释.

- **基本假设 4.** 为了模拟张量积, 本文使用了 Math.net 库自带的函数

$$\text{void } KroneckerProduct(Matrix\langle T\rangle other, Matrix\langle T\rangle result).$$

6.3 使用伪随机数抽样方法模拟量子测量

在平台 $Q|SI\rangle$ 中, 本文采用抽样获得符合给定概率分布的伪随机数, 并以此来模拟量子测量 [26].

在量子计算中, 量子测量由一组有界线性算子 $\{M_m\}$ 来描述, 其满足条件 $\sum_m M_m^\dagger M_m = I$. 用 m 表示测量结果, 它的对应的离散概率的集合为 P, 其中 $P = \{p_1, p_2, \ldots, p_m\}$. 测量结果对应的指示变量集合记为 Y, Y 取值为 $\{0, 1\}$, 分别表示未选中此变量和选中此变量. 进一步地, 假设当前的量子态为

$|\psi\rangle$, 指示随机变量为 $Y_1,\ldots,Y_m$, 并且假设 $\Pr[Y_i=1]=p_i$, 其中 $p_i=\langle\psi|M_i^\dagger M_i|\psi\rangle$, $P=\{p_1,\ldots,p_m\}$. 下面利用 $Q|SI\rangle$ 平台中的均匀分布 X 来模拟随机变量 Y.

Math.net 提供一个随机变量 X, 称为 *RandomSource*, 它均匀分布在 $(0,1)$ 上. 将 $[0,1]$ 区间分为 m 个小区间 $[0,p_1],(p_1,p_1+p_2],\ldots,(\sum_{i=1}^{m-1}p_i,1]$. 其中, 第 i 个小区间的宽度等于概率 p_i.

因此, 每个测量过程都依照以下策略执行:

(1) 假设给定一个测量 $\{M_m\}$ 和量子态 $|\psi\rangle$, 计算 $P=\{p_1,p_2,\ldots,p_m\}$, 其中 $p_i=\langle\psi|M_i^\dagger M_i|\psi\rangle$, 这给出随机变量 Y 的概率分布: $\Pr[Y=i]=p_i$.

(2) 检查集合 P 的值并进行筛选. 如果 $p_i=0$, 则在下一步丢弃掉 i 的值. 如果 $p_i=1$, 则返回索引 i 作为最终的结果, 并跳过下面的步骤.

(3) 假设 P' 是一个和筛选过的 P 具有相同数量的集合. 把 P 累加到 P' 上用以下方式: 对于每个在 P' 中的 p'_i, $p'_i=\sum_i p_i$.

(4) 使用 *RandomSource*, 生成一个在 $(0,1)$ 上均匀分布的伪随机数 x.

(5) 找到 p'_i, 使得 $p'_{i-1}\leqslant x$ 且 $p'_i\geqslant x$, 并返回索引 i. 需要注意的是当 $x<p'_1$ 时返回 $i=1$, 当 $x>p'_{m-1}$ 时返回 $i=m$.

在以 $P=\{p_1,\ldots,p_m\}$ 分布选择了 i 后, 函数返回了 i 的值, 并且作为原子操作, 量子态将被改变. 根据量子力学, 量子态 $|\psi\rangle$ 将会被改变为 $|\psi'\rangle=\frac{M_i|\psi\rangle}{\sqrt{\langle\psi|M_i^\dagger M_i|\psi\rangle}}$.

6.4 模拟量子 while 语言的操作语义

模拟用量子 **while** 语言编写的程序的计算是基于对语言的操作语义的模拟. 在 $Q|SI\rangle$ 的量子模拟引擎里, 为了在与经典量子混合程序中保持区分性和保留一定的方便性, 本文将量子的 **if** 子句记为 **cif**; 将量子 **while** 子句记为 **cwhile**. 为了模拟这两个函数, 对量子行为库中的函数方法和构造这些语法子句的函数进行包装.

量子程序的执行, 可以用组态 (configuration) 转移的描述方式进行刻画.

定义1 一个量子组态是一个有序对 $\langle S,\rho\rangle$. 其中, S 是一个量子程序或者是一个空程序 **E** (表示终止); ρ 是一个部分密度算子, 用来刻画量子变量的 (全局) 状态.

通过前面小节的准备, 可以定义量子 **while** 语言的操作语义的组态转移规则, 如下所述.

Skip. $\frac{}{\langle\mathbf{skip},\rho\rangle\to\langle\mathbf{E},\rho\rangle}$. **skip** 语句什么都不做然后停机. 在 $Q|SI\rangle$ 里, I 算子 (恒等变换) 和空语句都满足这个过程.

初始化. $\frac{}{\langle q:=|0\rangle,\rho\rangle\to\langle\mathbf{E},\rho_0^q\rangle}$, 其中,

$$\rho_0^q=\begin{cases}|0\rangle_q\langle 0|\,\rho\,|0\rangle_q\langle 0|\;+\;|0\rangle_q\langle 1|\,\rho\,|1\rangle_q\langle 0| & \text{if } type(q)=\mathbf{Boolean},\\ \displaystyle\sum_{n=-\infty}^{\infty}|0\rangle_q\langle n|\quad\rho\quad|n\rangle_q\langle 0| & \text{if } type(q)=\mathbf{Integer},\end{cases}$$

初始化语句中, "$q:=|0\rangle$" 把量子寄存器里面的状态设置为 $|0\rangle$ 基态.

在 $Q|SI\rangle$ 中, 初始化语句有两种形式: 当变量 q 是一个 *QBit* 类型时, 初始化语句将会被解释为 $[\![q:=|0\rangle]\!](\rho)=|0\rangle\langle 0|\,\rho\,|0\rangle\langle 0|+|0\rangle\langle 1|\,\rho\,|1\rangle\langle 0|$; 否则它将会被解释为 $[\![q:=|0\rangle]\!](\rho)=\sum_{n=0}^{d-1}|0\rangle\langle n|\,\rho\,|n\rangle\langle 0|$, 其中 d 是量子变量 q 的维度. 更重要的是, 在酉变换语句的帮助下, 本文提供了一些非常灵活的初始化方式.

酉变换. $\frac{}{\langle\bar{q}:=U[\bar{q}],\rho\rangle\to\langle\mathbf{E},U\rho U^\dagger\rangle}$. "$\bar{q}:=U[\bar{q}]$" 语句意为在量子寄存器 $\bar{q}$ 上作用酉变换 U.

在 $Q|SI\rangle$ 中, 对 $QBit$, Ket, $DensityOperator\langle T\rangle$ 这些对象, 设计了它们对应的酉变换的方法: $QuantumTypes\langle T\rangle.UnitaryTrans(Matrix\langle T\rangle\ other)$. 这些函数接受一个酉变换算子, 然后作用在量子变量上并返回空值. 同时, 本文还提供了一个全局函数

$$UnitaryGlobalTrans(QuantumTypes\langle T\rangle, Matrix\langle T\rangle),$$

来作用任意酉算子到任意合法的量子类型上.

量子 **while** 语言并不包括对纯态的任何声明语句, 这是因为对于一个纯态 $|\psi\rangle$ 来讲, 总是存在一个酉算子 U 满足 $|\psi\rangle = U|0\rangle$, 因此综合利用初始化语句和酉变换语句, 所有的纯态都可以被产生出来. 但是, 为了方便, $Q|SI\rangle$ 提供了非常灵活的初始化方法. 对 $QBit$, Ket 提供了向量的初始化方法, 对 $DensityOperator\langle T\rangle$ 提供了正算子的初始化方法.

顺序语句. $\frac{\langle S_1,\rho\rangle\to\langle S_1',\rho\rangle}{\langle S_1;S_2,\rho\rangle\to\langle S_1';S_2,\rho'\rangle}$. 当前版本的量子 **while** 语言不是为了并行的程序而设计, 因此, 程序语句的顺序复合是自发形成的.

条件语句. $\frac{}{\langle \mathbf{if}(\square m\cdot M[\bar{q}]=m\to S_m)\mathbf{fi},\rho\rangle\to\langle S_m, M_m\rho M_m^\dagger\rangle}$, 这里假定测量 $M=\{M_m\}$. 因此, 对每个可能的结果 m 而言, 有一条相应的规则.

执行条件语句的第一步是执行一个测量 M 在量子寄存器 $\bar{q}$ 上, 并观测输出结果 m. 然后根据输出的索引结果程序选择对应的子程序 S_m.

在 $Q|SI\rangle$ 里, 实现条件语句的是一个包装函数, 函数的原型是

$$\mathbf{cif}(QuantumTypes\langle T\rangle, MeasureMatrixSet, Func\langle T\rangle, Func\langle T\rangle\ldots).$$

在默认的配置里, $Func\langle T\rangle$ 的序列对应到输出结果的序列. 也就是说, 第 n 个子程序 $Func\langle T\rangle$ 对应到第 n 个测量输出结果. 同时, 本文也考虑了当用户没有提供所有的对应测量输出的子程序的情形. 程序逻辑的策略是, 当结果超出了用户定义的 $Func\langle T\rangle$ 数量时, $Q|SI\rangle$ 自动跳过这个条件语句, 这样仅仅有一个测量被作用在变量上.

量子条件语句与经典条件语句不同的是量子条件语句中的量子态可能被测量所改变. 在 $Q|SI\rangle$ 的设计中, 仅使用函数

$$\text{int } Measu2ResultIndex(MeasureMatrixSet),$$

来返回测得的值并执行对应的子程序. 在函数中, 实质上使用了函数 void $StateChange$(int) 来改变量子寄存器 $\bar{q}$, 使之成为测量后的态.

循环语句. (L0) $\frac{}{\langle \mathbf{while}(M[\bar{q}]=1)\mathbf{do}\ S\ \mathbf{od},\rho\rangle\to\langle \mathbf{E}, M_0\rho M_0^\dagger\rangle}$,

(L1) $\frac{}{\langle \mathbf{while}(M[\bar{q}]=1)\mathbf{do}\ S\ \mathbf{od},\rho\rangle\to\langle S;\mathbf{while}(M[\bar{q}]=1)\mathbf{do}\ S\ \mathbf{od}, M_1\rho M_1^\dagger\rangle}$.

为了实现这个循环语句, 使用了包装函数, 函数原型是

$$\mathbf{cwhile}(QuantumTypes\langle T\rangle, MeasureMatrixSet, \text{int}, Func\langle T\rangle).$$

这个函数接受一个量子类型对象, 一个测量算子和一个给定的整数. 函数首先比对测量结果和给定的整数是否一致, 如果和给定的一致, 则它进入循环体 $Func\langle T\rangle$, 否则此语句停止. 另外, 状态在测量后发生了改变. 如前所述, 首先调用函数 int $Measu2ResultIndex(MeasureMatrixSet)$, 并返回一个值, 然后调用 void $StateChange$(int) 改变量子寄存器.

7 实验

本文主要呈现了 3 个实验来展示平台的能力: Qloop, BB84 协议和 Grover 搜索算法. 读者可以在

我们发布的程序里[3] 找到实验的细节.

Qloop. Qloop 实验是一个初级量子算法的例子, 包括了量子信道、量子测量、量子 **while** 语句和一些量子变量. 本质上, 它可以被视作一个精简版的量子游走. 在这个测试中, 展示了平台 $Q|SI\rangle$ 的 3 个特性: 超算子、酉变换和量子测量.

Qloop 的基本思想是在量子态上执行超算子, 使状态发生改变. 使用计数器记录每次进入不同分支的次数. 之后, 对这种状态进行多次实验并进行测量, 计数器将显示和量子态所表述一致的概率分布.

BB84. BB84 是由 Bennett 和 Brassard 于 1984 年提出的量子密钥分发 (QKD) 协议 [27]. 该协议是经过了安全性证明的 [1], 依赖于不可克隆定理的密钥分发协议. 使用该协议, Alice 和 Bob 可以协议达成一个经典密钥串, 这个经典密钥串可以用于对经典 Bit 执行加密.

本实验考虑了几种不同的场景. 简单版的 BB84 案例是只考虑理想情况下最原始的 BB84 通信协议例子. 多客户端 BB84 例子展示了一个只有 Alice (服务端) 生成原始密钥, 多个 Bob (客户端) 与 Alice 协商协议的情形. 更为有趣的例子是在具有量子噪声的通道下的 BB84 协议. 一般来说, 没有量子系统真正的完全封闭, 所以需要使用描述开放量子系统的工具 —— 超算子, 来进行 QKD 影响因素的研究. QKD 的封装长度和采样百分比对量子噪声下的实际 QKD 协议传输能力至关重要. 使用 $Q|SI\rangle$ 平台, 可以对不同的量子通道测试不同的参数, 以期达到协议实用化的目的.

Grover 搜索算法. Grover 搜索算法 [2] 是量子计算领域中一个非常有趣而且令人印象深刻的算法. 它通过预言机 (Oracle) 搜索由 $0, 1, \ldots, N-1$ 共 N 个元素组成的数据库, 其中的预言机可以产生一个抽象的位置. 它可以以很高的成功率 ($O(1/N)$ 的错误率) 用 $O(\sqrt{N})$ 次查询搜索到问题的解.

本实验还考虑了更一般 Grover 搜索算法: 多对象 Grover 搜索算法, 即假设预言机中包含多个答案. 本文使用这样的策略: 每发现一个结果, 添加一个对应的盲箱, 来翻转预言机中发现的答案位置. 本实验发现了多对象 Grover 搜索算法的雪崩效应, 这表明该算法在某种意义上应该进行修改.

8 结语

本文设计并实现了一个用于量子计算机编程的新的设计环境 $Q|SI\rangle$. 它包括嵌入式量子 **while** 语言、量子模拟器、量子程序分析和验证的工具链. $Q|SI\rangle$ 可用于模拟量子算法, 验证量子程序正确性, 以及分析量子程序的终止和平均运行时间等方面.

本文展示了如何在经典计算机中用各个组件的协调搭配完成模拟量子行为, 讨论了采用伪随机数采样的量子测量仿真方法, 还在 $Q|SI\rangle$ 中实现了量子 **while** 语言的语法和语义.

尽管如此, 还有一些工作需要在近期完成. 首先, 目前本文使用了张量计算来相当笨拙地运行并模拟量子电路, 所以可能需要考虑文献 [10] 中所提及的时序分析和纠缠分析方法, 以期扩展模拟量子计算的能力. 其次, 考虑到混合程序 (经典程序和量子程序) 的验证, 需要进一步地研发经典程序分析组件与量子程序分析模块的结合. 同时, 还需要考虑不同量子计算程序的接口, 如 LIQU$i|\rangle$, ScaffCC 甚至实际运行于量子计算的平台 —— IBM 量子实验的接口. 通常这些多样化的平台可以提供给研究人员不同的视角来审视量子算法.

3) https://github.com/klinus9542/QSI.

参考文献

1 Shor P W, Preskill J. Simple proof of security of the bb84 quantum key distribution protocol. Phys Rev Lett, 2000, 85: 441–444

2 Grover L K. A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, 1996. 212–219

3 Harrow A W, Hassidim A, Lloyd S. Quantum algorithm for linear systems of equations. Phys Rev Lett, 2009, 103: 150502

4 Ömer B. A Procedural Formalism for Quantum Computing. Wien: Technical University of Vienna, 1998

5 Selinger P. A brief survey of quantum programming languages. In: Proceedings of International Symposium on Functional and Logic Programming. Berlin: Spring, 2004. 1–6

6 Bettelli S, Calarco T, Serafini L. Toward an architecture for quantum programming. Eur Phys J D-Atomic Mol Opt Plasma Phys, 2003, 25: 181–200

7 Sanders J W, Zuliani P. Quantum programming. In: Proceedings of the 5th International Conference on Mathematics of Program Construction. Berlin: Springer, 2000. 80–99

8 Green A S, Lumsdaine P L, Ross N J, et al. Quipper: a scalable quantum programming language. ACM SIGPLAN Notice, 2013, 48: 333–342

9 Wecker D, Svore K M. Liquid: a software design architecture and domain-specific language for quantum computing. arXiv:1402.4467

10 JavadiAbhari A, Patil S, Kudrow D, et al. Scaffcc: a framework for compilation and analysis of quantum computing programs. In: Proceedings of the 11th ACM Conference on Computing Frontiers, Cagliari, 2014

11 JavadiAbhari A, Patil S, Kudrow D, et al. Scaffcc: scalable compilation and analysis of quantum programs. Parall Comput, 2015, 45: 2–17

12 Smelyanskiy M, Sawaya N P D, Aspuru-Guzik A. qHiPSTER: the quantum high performance software testing environment. arXiv:1601.07195

13 Ying M S. Floyd-hoare logic for quantum programs. ACM Trans Program Lang Syst, 2011, 33: 1–49

14 Ying M S. Foundations of quantum programming. In: Proceedings of the 8th Asian Symposium on Programming Languages and Systems, Shanghai, 2010. 16–20

15 Svore K M, Aho A V, Cross A W, et al. A layered software architecture for quantum computing design tools. IEEE Comput, 2006, 39: 74–83

16 Ying M S, Feng Y. Quantum loop programs. Acta Inf, 2010, 47: 221–250

17 Ying M S, Yu N K, Feng Y, et al. Verification of quantum programs. Sci Comput Program, 2013, 78: 1679–1700

18 Ying M S, Ying S G, Wu X D. Invariants of quantum programs: characterisations and generation. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, Paris, 2017. 818–832

19 Liu T, Li Y J, Wang S L, et al. A theorem prover for quantum hoare logic and its applications. Comput Sci, 2016, arXiv:1601.03835

20 Nielsen M A, Chuang I L. Quantum Computation and Quantum Information. Cambridge: Cambridge University Press, 2010

21 Shende V V, Bullock S S, Markov I L. Synthesis of quantum-logic circuits. IEEE Trans Comput-Aid Des Integr Circ Syst, 2006, 25: 1000–1010

22 Dawson C M, Nielsen M A. The solovay-kitaev algorithm. Quant Inf Comput, 2006, 6: 81–95

23 Ying M S, Feng Y. A flowchart language for quantum programming. IEEE Trans Softw Eng, 2011, 37: 466–485

24 Smith R S, Curtis M J, Zeng W J. A practical quantum instruction set architecture. arXiv:1608.03355

25 Barenco A, Bennett C H, Cleve R, et al. Elementary gates for quantum computation. Phys Rev A, 1995, 52: 3457–3467

26 Devroye L. Sample-based non-uniform random variate generation. In: Proceedings of the 18th Conference on Winter Simulation, Washington, 1986. 260–265

27 Bennett C H, Brassard G. Quantum cryptography: public key distribution and coin tossing. Theor Comput Sci, 2014, 560: 7–11

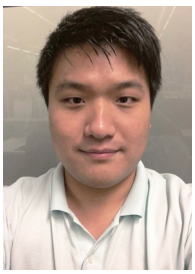
$Q|SI\rangle$: a quantum programming environment

Shusen LIU[1,4*], Li ZHOU[2], Ji GUAN[1], Yang HE[1], Runyao DUAN[1] & Mingsheng YING[1,3,2*]

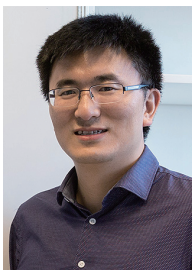
1. *Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney* 2007*, Australia*;
2. *Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China*;
3. *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190*, China*;
4. *School of Data and Computer Science, Sun Yat-sen University, Guangzhou* 516000*, China*
* Corresponding author. E-mail: shusen88.liu@gmail.com, Mingsheng.Ying@uts.edu.au

Abstract This paper describes a quantum programming environment called $Q|SI\rangle$. It is a platform created in the .NET language to support quantum programming as a quantum extension of the **while**-language. This platform includes a compiler for the quantum **while**-language and a chain of tools for the simulation of quantum computations, optimization of quantum circuits, and analysis and verification of quantum programs. Throughout the paper, $Q|SI\rangle$ is used to simulate quantum behaviors on classical platforms and demonstrate the combination of components. The scalable framework allows users to program customized functions on the platform. The compiler works as the core of $Q|SI\rangle$, bridging the gap between quantum hardware and quantum software. The built-in decomposition algorithms enable universal quantum computation on current quantum hardware.

Keywords quantum programming, quantum compilation, quantum simulation, quantum program analysis, quantum program verification



Shusen LIU was born in Xuzhou Jiangsu. He is a Ph.D. candidate at the Centre for Quantum Software and Information at the University of Technology, Sydney. He is interested in technologies related to quantum programming. He also focuses on quantum simulation and emulation.



Runyao DUAN was born in Dali Yunnan. He is currently a professor and the director of the Centre for Quantum Software and Information at the University of Technology, Sydney. His research interests include quantum computing and quantum information theory. From December 2008 to September 2016, he worked at the Centre for Quantum Computation and Intelligent Systems, Faculty of Engineering and Information Technology (FEIT), University of Technology, Sydney (UTS), Australia. He was initially as senior lecturer, then became an associate professor in August 2010, and from July 2012 worked as a future fellow and professor of the Australian Research Council.

Mingsheng YING graduated from the Department of Mathematics, Fuzhou Teachers College, Jiangxi, China, in 1981. He is a distinguished professor and the research director at the Centre for Quantum Software and Information at the University of Technology, Sydney, and Cheung Kong professor with the State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are quantum computation, programming theory, and the foundations of artificial intelligence.