LQPL(Linear Quantum Programming Language)

Brett Giles

Department of Computer Science
University of Calgary

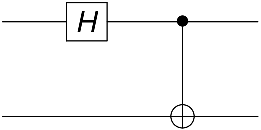
2012-06

Outline

Quantum Circuits

- Transforms (e.g., Hadamard, Not, Pauli)
- Qubit measures
- Controlled Transforms (C-Not, C-Had, Toffoli)

For example — Entanglement:



QPL by Selinger

Data with Classical Control

- Explicitly handle classical control, loops, subroutines
- Denotational semantics
- Discussion and implications of handling product and sum types

LQPL

LQPL is based on QPL's language and semantics. Differences:

- The inclusion of probabilistic integers (e.g., `i` is 1 with 25% probability, 17 with 75%)
- The inclusion of probabilistic algebraic data types (e.g., `list1` has 50% chance of being empty or having one element)
- Language constructs for creating and using these probabilistic items.
- The explicit use of non-probabilistic classical data (integers).
- The removal of controlled transforms and the addition of syntax for quantum control.

LQPL Language

Structure Data type declarations (sum, product, recursive) and subroutines are at top level, in global scope

Qubits (`x = ` $|0\rangle$); transform; measure

Types (`lis = Nil`); case

Integers (`i = 5`); use

Control (`Had q <= r1,r2`); control target (left hand side) any statements; control elements (right hand side) any data type with qubits.

Classical Result of Integer `use`; pass to subroutines; “switch”

Looping is accomplished by subroutine calls; multiple return points by sum types;

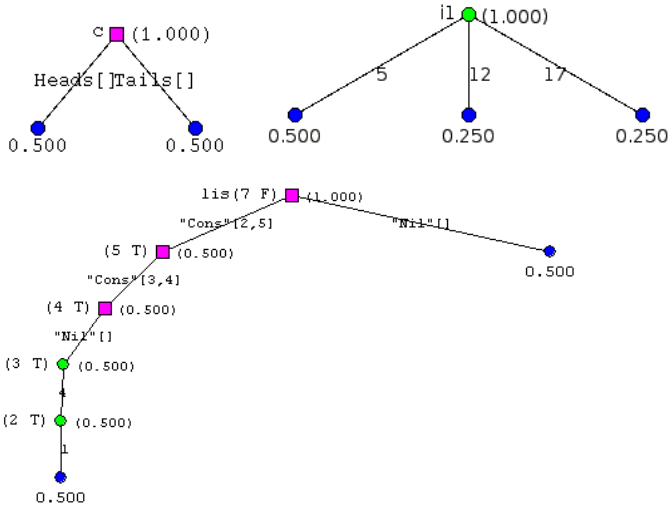
The Compiler

- Performs type inference / checking. (Expressions, subroutine parameters and return values, “classicality” or “quantum” of expressions)
- Enforces linear usage of all variables — i.e., enforce “no-duplication” of qubits, applies the same rule to algebraic data and probabilistic integers.
- Enforces balanced data after measures or cases — e.g., if a qubit `q` is created when a list is `Nil`, it must also be created when the list is `Cons _ _`

A “machine” for LQPL

- The machine state is primarily a *quantum stack*
- Stack is equivalent to a probability distribution of density matrices
- Qubits have up to four substacks, integers a variable number and algebraic datatypes at most one substack per constructor
- All operations *except quantum control* are pushed down to the appropriate entry on the stack
- Quantum control involves “rotating” the stack - expensive

Quantum Stacks - “bits$^+$”



Ice Cream - Algorithm

Problem:

- One ice cream? 3 grand kids — one of whom is a girl.
- Girl has to be first! .. but can't cheat the boys.

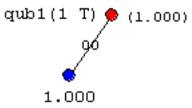
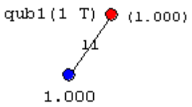
Solution:

- Girl repeatedly flips the coin until she gets heads: she gets the ice cream if she gets heads in an even number of flips. Otherwise, she passes the coin to one of the boys.
- He tosses the coin: If he gets Heads he wins the ice cream, otherwise it goes to the remaining boy.

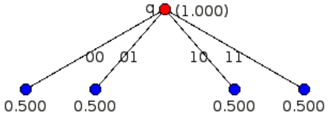
With what probability does the girl get the Ice Cream?
DEMO

(Example due to Carroll Morgan)

Quantum Stacks - Qubits

Standard	Density	QStack
$qub1 = \|0\rangle$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	
$qub1 = \|1\rangle$	$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$	

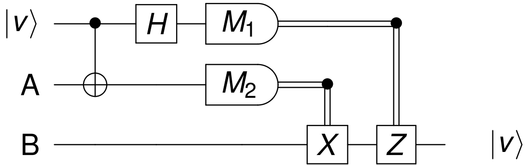
Quantum Stacks - Qubits

Standard	Density	QStack
$q = \frac{1}{\sqrt{2}}\|0\rangle + \frac{1}{\sqrt{2}}\|1\rangle$	$\begin{pmatrix} .5 & .5 \\ .5 & .5 \end{pmatrix}$	

Demo - coinflip

Quantum Teleportation

A(lice) and *B*(ob) are qubits in a Bell (aka EPR) state. Then, Alice can transfer a qubit to Bob by sending two bits of information.



DEMO “teleport.qpl”

Grover's search

- Determine for which $x \in \mathbb{B}^n$ is $f : \mathbb{B}^n \to \mathbb{B}$ is 1.
- Classically, this requires the 2^n applications of f. The quantum algorithm requires $\mathcal{O}(\sqrt{2^n})$ applications.
- For the algorithm, first define:

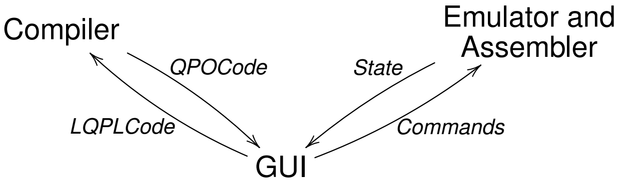
$$U_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle \text{ and } U_0 \left|x\right\rangle = \begin{cases} \left|x\right\rangle & \text{if any } x \neq 0 \\ -\left|x\right\rangle & \text{if } x = 0^n \end{cases}$$

then:

- Start with n zeroed qubits and apply Hadamard to them.
- Apply $G = -H^{\otimes n} U_0 H^{\otimes n} U_f$ approximately $\sqrt{2^n}$ times.
- Measure the qubits, forming an integer and check the result.

DEMO "Grover"

Overall design



Data structure for the emulator

Base Tuple of the QuantumStack, ClassicalStack, et. al..

Control Add list of controlling qubits and functions to move back and forth from Base.

Stream An infinite list of (Integer, Control) pairs that approximate the end result — the further down the list, the closer the approximation.

The majority of QPO instructions are defined on “Base”. Transforms are defined at the “Control” stage and subroutine calls are defined at the “Stream” stage.

Modules and Interfaces

GUI Formerly in Haskell, using Gtk2Hs. Complex to build, tightly coupled to the emulator and compiler. Now in Swing (Java), the GUI provides visualization of the quantum stack and allows inspection of the other data stored in the LQPL emulator.

Emulator Written in Haskell, extensive use of laziness (e.g., infinite lists)

Compiler Also in Haskell, follows standard compiler construction practices.

I/f The GUI connects to both the emulator and the compiler is via TCP/IP based messaging, significantly reducing the coupling.

Operational Semantics

The machine language has an operational semantics, defined as state transitions dependant upon the next instruction to be executed.

$$
\begin{aligned}
&(\text{QLoad } x\ \ |k\rangle : \mathcal{C}, S, Q, D, N) \\
&\qquad\qquad\qquad \Longrightarrow\ (\mathcal{C}, S, x{:}[|k\rangle \to Q], D, N) \\
&(\text{QCons } x\ c{:}\mathcal{C}, S, Q, D, N) \\
&\qquad\qquad\qquad \Longrightarrow\ (\mathcal{C}, S, x{:}[c\{\} \to Q], D, N) \\
&(\text{QMove } x{:}\mathcal{C}, v{:}S, Q, D, N) \\
&\qquad\qquad\qquad \Longrightarrow\ (\mathcal{C}, S, x{:}[\bar{v} \to Q], D, N) \\
&(\text{QBind } z_0{:}\mathcal{C}, S, x{:}[c\{z'_1, \ldots, z'_n\} \to Q], D, N) \\
&\Longrightarrow\ (\mathcal{C}, S, x{:}[c\{z(N), z'_1, \ldots, z'_n\} \to Q[z(N)/z_0]], D, N')
\end{aligned}
$$

Machine language

- Instruction oriented — Assembler-like language with thirty opcodes
- Qubit instructions — QLoad, AddCtrl, UnCtrl, QApply
- QStack manipulations — QPullup, Rename, EnScope, DeScope, SwapD
- Data Types — QCons, QBind, QUnbind,...
- Measure / deconstruction — Measure, Split, QUnbind, Use, QDelete,...
- Classical ops — CGet, CPut, CApply, CLoad, CPop
- Branches / Subroutines — Jump, CondJump, Call, Return

Observations

- Can write quantum algorithms at a reasonably good level of abstraction ...
- Can test SMALL quantum programs (factoring primes totally beyond current LQPL implementation)
    - teleportation
    - quantum arithmetic
    - Grover search
    - Simon's
- Quantum programming (with one Qubit) contains probabilistic programming!
- Can program (small) probabilistic algorithms.

Next...

- LQPL possible enhancements
  - Create transforms
  - Speed and memory improvements
  - other features...
- Revisit semantics

Thanks!

Thank You