Full Title[*]

Subtitle[†]

Anonymous Author(s)

Abstract

Text of abstract ....

Keywords keyword1, keyword2, keyword3

1 Introduction

- Quantum computing hot.
- Unitary evolution; many algorithms promote classical reversible functions into unitary gates (examples, including some from Quipper)
- Common case: given a permutation between finite sets, find a reversible program that implements it. Answer depends on the available reversible primitives.
- Example, say we want to transpose a matrix, i.e., write a permutation of type `A*B <-> B*A` where `A` and `B` are the dimensions. Wikipedia example (https://en.wikipedia.org/wiki/In-place_matrix_transposition):

  ```
  11 12 13 14
  21 22 23 24
  ```

  transposed to

  ```
  11 21
  12 22
  13 23
  14 24
  ```

  Here `A` is `Nat x Nat` and `B` is `Nat x Nat x Nat x Nat` and the input and output matrices are:

  ```
  M  = (11 , 21) , (12 , 22) , (13, 23) , (14 , 24)

  trM = (11 , 12 , 13 , 14) , (21 , 22 , 23 , 23)
  ```

  Say we are given a language like Pi that is sound and complete with respect to permutations on finite types, we would write the permutation like so.

  ```
  WRITE PERMUTATION
  ```

  This code does not use additional space, i.e., it performs the matrix transpose in constant space, i.e., performs an in-place matrix transposition. It is well know that with additional space, one can write more efficient matrix transpositions (explain and citations).
- So many reversible/quantum circuits allow the introduction of so-called ancilla wires, which serve as scratch space (goes back to Toffoli). The problem is that there is a discipline for using ancilla bits: they need to be created in a known state and can only be gc'ed when they are returned to the same state (explain).
- No language knows how to automatically guarantee this condition; left to the programmer (Quipper), or some (incomplete?) dynamic checks (Ricercar).
- We solve this using **fractional types**.

[*]Title note

[†]Subtitle note

PL'18, January 01–03, 2018, New York, NY, USA
2018.

2 Extra for now

Complexity perspective: https://www.scottaaronson.com/papers/gates.pdf

Quipper: https://arxiv.org/pdf/1304.3390.pdf; uses ancilla; one use is to compile irreversible circuits to reversible ones; need ancilla bits; more generally several quantum algorithms need ancilla bits (see below); in quipper de-allocation left to programmer.

Ricercar tries to define rules for ancilla but not complete https://msoeken.github.io/papers/2015_rc_2.pdf

Leads to bugs as analyzed by http://drops.dagstuhl.de/opus/volltexte/2019/10196/pdf/OASIcs-PLATEAU-2018-4.pdf:

> Bug type 6: Incorrect composition of operations using mirroring Section 4.5 discussed how bugs in deallocating ancillary qubits can happen due to bad parameters. Here we see how bugs in deallocating ancillary qubits can happen due to incorrect composition of operations following a mirroring pattern. For example, in Table 7, the operations in rows 2 and 3 are respectively mirrored and undone in rows 6 and 5. These lines of code need careful reversal of every loop and every operation.
> A common pattern in quantum programs involves performing operations (e.g., add), contingent on a set of qubits known as control qubits. Without language support, this pattern needs many lines of code and manual allocation of ancillary qubits. In the Scaffold code example in Table 7, rows 3 and 5 are just computing the intersection of qubits q, with the help of ancillary qubits initialized in row 1, in order to realize the controlled rotation operation in row 4. Furthermore, quantum algorithms often need varying numbers of control qubits in different parts of the algorithm, leading to replicated code from multiple versions of the same subroutine differing only by the number of control qubits.

Uses of ancillas: https://quantum.country/search:

> There's a rough heuristic worth noting here, which is that you can often convert if-then style thinking into quantum circuits. You introduce an ancilla qubit to store the outcome of evaluating the if condition. And then depending on the state of the ancilla, you perform the appropriate state manipulation. Finally, when possible you reverse the initial computation, resetting the ancilla to its original state so you can subsequently ignore it.

https://www.nap.edu/read/25196/chapter/5#73

> For error correction, one needs to replicate the state of a qubit onto a number of qubits. While the no cloning theorem prevents one from copying the state of a qubit directly onto another, one can create a redundant entangled qubit state of many qubits. The key is that the qubits to be entangled must start out in a known state. Qubits with a known state (for purposes of this discussion, it will be the state `|0>`), called "ancilla qubits," may be added to a computation for this purpose. Since the state of ancilla qubits are known, it is possible to create a simple circuit that makes the output state of all these ancilla qubits match the protected qubit: run each ancilla through a controlled-NOT gate, where the control is driven by the qubit that needs to be replicated. Assume that there is a qubit with state `PSI` that we want to protect, where `|PSI>` represents an arbitrary superposition state `|PSI> = a0 |0> + a0 |1>` In the CNOT gate, the ancilla `|0>` state will remain a `|0>` state by the `|0>` component of `|PSI>`, but it will be converted to `|1>` by the `|1>` component of `|PSI>` The result of this operation is the newly entangled two-qubit state `a0 |00> + a1 |11>`, creating a system in which the ancilla qubit is now perfectly entangled with the first qubit. Adding more ancillas increases the distance of the repetition code.

https://www.sigarch.org/the-case-for-quantum-computing/

> Imagine a 3-qubit quantum majority code in which a logical "0" is encoded as "000" and a logical "1" is encoded as "111." Just as with a classical majority code, a single bit-flip error can be corrected by restoring to the majority value. Unlike a classical code, however, we can not directly measure the qubits else their quantum state will be destroyed. Instead, we measure syndromes from each possible pair of qubits by interacting them with an ancilla, then measure each ancilla. Although the errors to the qubits are actually continuous, the effect of measuring the ancilla is to discretize the errors, as well as inform us whether an error occurred so that it can be corrected. With this methodology, quantum states are restored in a modular way for even a large quantum computer. Furthermore, operations on error-corrected qubits can be viewed as digital rather than analog, and only a small number of universal operations (H, T, CNOT) are needed for universal quantum computation. Through careful design and engineering, error correction codes and this small set of precise operations will lead to machines that could support practical quantum computation.

https://homepages.cwi.nl/~rdewolf/qcnotesv2.pdf

> Using an ancilla qubit, it is possible to avoid doing any intermediate measurements in a quantum circuit. Show how this can be done. Hint: instead of measuring the qubit, apply a CNOT that "copies" it to a new `|0>`-qubit, which is then left alone until the end of the computation. Analyze what happens.

3 Reversible/Quantum Circuits and Ancilla Wires

Early use of ancillas in Toffoli's paper to implement arbitrary reversible functions using a fixed number of 3 input gates https://link.springer.com/content/pdf/10.1007%2F3-540-10003-2_104.pdf

Use examples from Ricercar

Is the below a possible example?

```
Say we already have a permutation A <-> B
we can implement a permutation X <-> Z
when there exists Y such that A/X = Y = B/Z

X -> X * Y * 1/Y
  -> A * 1/Y
  -> B * 1/Y
  -> Y * Z * 1/Y
  -> Z
```

4 A Comonad for Pi

Adding variables to Pi first?

Plain Pi over finite types:

Value types	t	$::=$	$0 \mid 1 \mid t + t \mid t * t$
Values	v	$::=$	$\star \mid inl(v) \mid inr(v) \mid (v, v)$
Level-1 types			$t \leftrightarrow t$
Level-1 programs	c	$::=$	$\cdots$
Level-2 types			$c \Leftrightarrow c$
Level-2 programs	α	$::=$	$\cdots$

Now we define pointed types $[t \bullet v]$ which are a record consisting of a value type together with a value of that type. Both the level-1 and level-2 program lift naturally to the world of pointed types but instead of manually rewriting everything we use the fact that pointed types form a comonad. So for example, given a level-1 program we should be able to just use extend to compose instead of plain composition of combinators??

Then we can define η and ϵ in the comonadic world.

5 Fractional Pi

Adding variables to Pi first?

6 Denotations

Positive rational numbers are a model. Apparently there is a categorification https://alistairsavage.ca/pubs/Copelli-Categorification_of_the_Nonnegative_Rational_Numbers.pdf

Use all the constructions name, coname, etc. and see what they do in this context!

A Appendix

Text of appendix …