UNIVERSITY OF COPENHAGEN

DIKU, Department of Computer Science

Design of quantum circuits with RFun

Michael Kirkedal Thomsen
m.kirkedal@di.ku.dk

Abstract

Languages for describing quantum circuits [1] follow standard HDLs with extension for quantum constructs, thus not exploiting the properties of the quantum logic model. Here we explore how DSLs for a reversible model can be transferred to design of quantum circuits.
Specifically, we implement a quantum extension to combinator language [3] in a reversible functional language, RFun. This gives the strength of circuits described in a combinator languages with type safety of RFun.

RFun, a Reversible Functional Language

RFun [4, 2] is a *history-free* reversible functional language. Fundamentally, a reversible computation can be considered as an injective transformation of a state into an updated state, thus RFun implements (often) injective *partial* functions. Noticeable for RFun is a type system supporting:

- linear usage of resources – no duplication
- ancilla usage – guaranteed restore
- local inverse semantics – invertibility

Tutorial and interpreter found at:
`http://bit.ly/rfun-lang`

```
zip :: ([a], [b]) ↔ [(a, b)]
zip ([], []) = []
zip ((a:as), (b:bs)) =
  let ls = zip (as, bs)
  in ((a, b):ls)

unzip :: [(a, b)] ↔ ([a], [b])
unzip l = zip! l

map :: (a ↔ b) → [a] ↔ [b]
map fun [ ] = [ ]
map fun (l:ls) =
  let l' = fun l
      ls' = map fun ls
  in (l':ls')
```

Describing Quantum Circuits in RFun as Combinators

Here we present and implements a combinator-style functional language designed to be close to the quantum logical gate-level. The combinators include high-level constructs such as ripples, but also the recognisable inversion combinator f^{-1}, which defines the inverse function of f using an efficient semantics.
It is important to ensure that all circuits descriptions follows model constraints, and furthermore we must require this to be done statically. This is ensured by the type system, which also allows the description of arbitrary sized circuits. The combination of the functional language and the restricted reversible model results in many arithmetic laws, which provide more possibilities for term rewriting and, thus, the opportunity for good optimisation.

The combinator language is based on the following slightly simplified syntax.

$D ::=$	$(funcName = R)^*$	definition
$R ::=$	Id | Not | H | T | ...	basic gates
	Zip | Split | Concat	reordering
	$funcName$	function use
	$R;R \mid [R,R]$	composition
	$\alpha R \mid \diagdown R \mid \diagup R$	map, ripples
	R^{-1}	inverse

Figure 1:Syntax for central subset of combinator language.

Note, that this includes only resource preserving combinators and basic quantum gates and, thus, is a restriction to unitarity operations. Measurements and setup should be handled externally.
The following implementation examples define the combinator language in RFun.

```
-- Wire is defined as a number
data Wire = Z | S Wire

data Gate = Id Wire | Not Wire
   | H Wire | T Wire | Cnot ([Wire], Wire)

hadamard :: Wire ↔ Gate
hadamard w = (H w)

cnot :: Wire → Wire ↔ Gate
cnot q1 q2 = Cnot [q1] q2

-- Simple example circuit
qec :: ([Wire] ↔ [Gate])
     → [Wire] ↔ [Gate]
qec phase c =
  let ch = map hadamard c
      p  = phase ch
      ci = map hadamard c
  in ci
```

References

[1] Chong, F. T., Franklin, D., and Martonosi, M. Programming languages and compiler design for realistic quantum hardware.
Nature 549, 7671 (09 2017), 180–187.

[2] Kaarsgaard, R., and Thomsen, M. K. Rfun revisited.
In *Nordic Workshop in Program Theory, NWPT '17* (2017), LNCS, pp. 65–67.

[3] Thomsen, M. K. Describing and optimizing reversible logic using a functional language.
In *Implementation and Application of Functional Languages, IFL '12* (2012), vol. 7257 of *LNCS*, pp. 148–163.

[4] Yokoyama, T., Axelsen, H. B., and Glück, R. Towards a reversible functional language.
In *Reversible Computation, RC '11* (2012), vol. 7165 of *LNCS*, pp. 14–29.