On Dynamic Lifting and Effect Typing in Circuit Description Languages (Extended Abstract)

Andrea Colledan[1,2] and Ugo Dal Lago[1,2]

[1] University of Bologna, Italy
[2] INRIA Sophia Antipolis, France

Abstract

Despite the undeniable fact that large-scale, error-free quantum hardware has yet to be built [12], research into programming languages specifically designed to target architectures including quantum hardware has taken hold in recent years [11]. Most of the proposals in this sense (see [5, 15, 16] for some surveys) concern languages that either express or can be compiled into some form of *quantum circuit* [9], which can then be executed by quantum hardware. This reflects the need to have tighter control over the quantum resources that programs employ. In this scenario, the idea of considering high-level languages that are specifically designed to *describe* circuits and in which the latter are in fact first-class citizens is particularly appealing.

A typical example of this class of languages is `Quipper` [6, 7], whose underlying design principle is that of enriching a very expressive functional language like `Haskell` with the possibility of manipulating quantum circuits. In other words, programs do not just build circuits, but also treat them like data. `Quipper`'s meta-theory has been studied in recent years through the introduction of a family of research languages that correspond to suitable `Quipper` fragments and extensions, which usually take the form of linear λ-calculi. This family includes languages such as Proto-Quipper-S [14], Proto-Quipper-M [13], Proto-Quipper-D [3, 4] and Proto-Quipper-L [8].

An aspect that so far has only marginally been considered by the research community is the study of the meta-theory of so-called *dynamic lifting*, i.e. the possibility of allowing any classical value flowing in one of the wires of the underlying circuit, naturally unknown at circuit building time, to be *visible* in the host program for control flow. As an example, one could append a unitary to some wires *only if* a previously performed measurement has yielded a certain outcome. This is commonly achieved in many quantum algorithms via classical control, but `Quipper` also offers a higher-level solution precisely in the form of dynamic lifting, as shown in the example program in Figure 1. Notably, such a program cannot be captured by any of the calculi in the Proto-Quipper family, with the exception of Lee et al.'s Proto-Quipper-L [8], arguably the most recent addition to the family.

Looking at the `Quipper` program in Figure 1, one immediately realizes that the two branches of both occurrences of the `if` operator change the underlying circuit in a uniform way, i.e. the number and type of the wires are the same in either branch. What if, for instance, we wanted to condition the execution of a *measurement* on a lifted value, like in Figure 2? Such situations can arise, for example, in one-way quantum computing, and the so-called measurement calculus [2] indeed allows the execution of a measurement to be conditioned on the result of a previous measurement. Unfortunately, `Quipper` does *not* allow the program in Figure 2 to be typed. It is therefore natural to wonder whether this is an intrinsic limitation, or if a richer type system can deal with a more general form of circuits.

In this talk, we introduce a generalization of Proto-Quipper-M, called Proto-Quipper-K, in which dynamic lifting is available in a very general form. The evaluation of a Proto-Quipper-K term M can involve the lifting of a bit value into a variable u and consequently produce in

```
teleport ::  Qubit -> Qubit -> Qubit -> Circ Qubit
teleport b q a = do
 a <- qnot a `controlled` q
 q <- hadamard q
 (x,y) <- measure (q,a)
 (u,s) <- dynamic_lift (x,y)
 b <- if s then gate_X b else return b
 if u then gate_Z b else return b
```

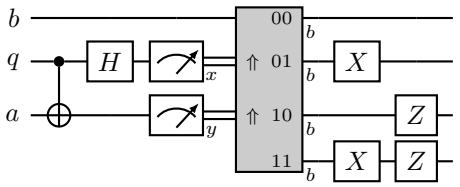

Figure 1: Quantum teleportation circuit with dynamic lifting. The gray box represents the dynamic lifting of bit wires x, y and the extension of the circuit on wire b with one of four possible continuations, depending on the results of the intermediate measurement.

```
oneWay ::  Qubit -> Qubit -> ???
oneWay q a = do
 q <- hadamard q
 x <- measure q
 u <- dynamic_lift x
 if u then measure a else return a
```

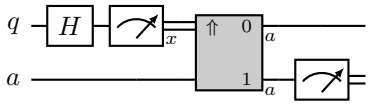

Figure 2: An example of conditional measurement. This program is ill-typed in `Quipper`.

output not a single result in the set VAL of values, but possibly one distinct result for each possible value of u. Therefore, it is natural to think of M as a computation that results in an object in the set $\mathcal{K}_{\{u\}}(VAL)$, where $\mathcal{K}_{\{u\}} = X \mapsto (\{u\} \to \{0,1\}) \to X$ is a functor such that an element of $\mathcal{K}_{\{u\}}(X)$ returns an element of X for each possible Boolean assignment to u.

This approach, though apparently straightforward, scales up in a nontrivial way. Since the lifting of a bit can happen conditionally on the value of a previously lifted variable, in the event that *more than one* variable is lifted, we have to resort to a tree-like structure to keep track of the eventual dependencies between consecutive liftings. We call such a structure a *lifting tree* and employ it pervasively throughout our work. In general, a program whose lifting pattern is captured by a lifting tree $\mathfrak{t}$ produces a *lifted value* as a result, namely an element of $\mathcal{K}_{\mathfrak{t}}(VAL)$, where $\mathcal{K}_{\mathfrak{t}} = X \mapsto (\mathscr{P}_{\mathfrak{t}} \to X)$ and $\mathscr{P}_{\mathfrak{t}}$ is the set of all assignments of lifted variables which describe valid root-to-leaf paths in $\mathfrak{t}$. Since by design we want to handle situations in which a circuit, and by necessity the term building it, can produce results which have distinct *types*—and not only distinct *values*—depending on the values of the lifted variables, we also employ (in the spirit of the type and effects paradigm [10]) an effectful notion of *type*, typing computations according to some *lifted type*, that is, an element of $\mathcal{K}_{\mathfrak{t}}(TYPE)$, where $TYPE$ is the set of Proto-Quipper-K types.

Proto-Quipper-K is thus capable of producing not only circuits like the one in Figure 1, but rather a more general class of circuits whose structure and type *essentially depend* on the values flowing through the lifted channels, like the one in Figure 2. The main results that we give, beside the introduction of the language itself, its type system, and its operational semantics, are the type soundness results of subject reduction and progress, which together let us conclude that well-typed Proto-Quipper-K programs do not go wrong from an operational point of view. An extended version of our work is already available in [1].

References

[1] Andrea Colledan and Ugo Dal Lago. On dynamic lifting and effect typing in circuit description languages (extended version), 2022.

[2] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. *J. ACM*, 54(2), April 2007.

[3] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. A tutorial introduction to quantum circuit programming in dependently typed proto-quipper. In Ivan Lanese and Mariusz Rawski, editors, *Proc. of RC*, pages 153–168, Cham, 2020.

[4] Peng Fu, Kohei Kishida, and Peter Selinger. Linear dependent type theory for quantum programming languages: Extended abstract. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proc. of LICS*, pages 440–453, 2020.

[5] Simon J. Gay. Quantum programming languages: Survey and bibliography. *Math. Struct. Comput. Sci.*, 16(4):581–600, August 2006.

[6] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. An introduction to quantum programming in quipper. In *Proc. of RC*, pages 110–124, 2013.

[7] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper. In *Proc. of PLDI*, page 333–342, June 2013.

[8] Dongho Lee, Valentin Perrelle, Benoît Valiron, and Zhaowei Xu. Concrete Categorical Model of a Quantum Circuit Description Language with Measurement. In *Proc. of FSTTCS*, volume 213 of *LIPIcs*, pages 51:1–51:20, 2021.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[10] Flemming Nielson and Hanne Riis Nielson. Type and effect systems. In Ernst-Rüdiger Olderog and Bernhard Steffen, editors, *Correct System Design: Recent Insights and Advances*, pages 114–136. Springer Berlin Heidelberg, 1999.

[11] Jens Palsberg. Toward a universal quantum programming language. *XRDS: Crossroads*, 26(1):14–17, September 2019.

[12] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[13] Francisco Rios and Peter Selinger. A categorical model for a quantum circuit description language. In *Proc. of QPL*, pages 164–178, June 2017.

[14] Neil Ross. *Algebraic and Logical Methods in Quantum Computation*. PhD thesis, 2015.

[15] Peter Selinger. A brief survey of quantum programming languages. In *Proc. of FLOPS*, pages 1–6, 2004.

[16] Mingsheng Ying. *Foundations of Quantum Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2016.