Toward certified quantum programming*

Sébastien Bardin[1], François Bobot[1], Christophe Chareton[1], Valentin Perelle[1], and Benoît Valiron[2]

[1]CEA, LIST, Software Reliability and Security Laboratory
F-91911 Gif-sur-Yvette Cedex, France
firstname.lastname@cea.fr
[2]LRI, CentraleSupélec, Université Paris-Saclay, Orsay, France
benoit.valiron@lri.fr

1 Introduction

Developers of quantum programming languages have the difficult task to create programs or languages that they can't easily test or debug. Indeed, quantum programming is, by nature, dedicated to run algorithms that cannot be run by classical computers. Furthermore, at least in the first ages of it, quantum computing must be costly, so that test runs might not be available at will as it is the case with classical computing. In addition, a future quantum developer won't have the possibility to suspend the execution of a system to inspect its state at some point.

The question of correctness of quantum programs is then paramount. In the literatures, several techniques have been proposed to tackle this issue. A first approach attempts at extending model-checking techniques to the quantum case [8, 15]. The drawback is the non-parametricity of the specifications, and their limitations in terms of number of qubits, A second approach is the language Qwire [11, 12], embedded in the Coq proof assistant which features a powerful dependent type-system. The first problem of this approach is the lack of automation: in Coq, proofs are automatically checked but must be given by the user. The second issue comes with the combination of types and specifications: the type annotations are complex and their proofs need to be given up front for the program to compile. A last approach is the extension of Hoare Logic (QHL) to quantum programming [1,4,13]. QHL [13] interprets density operators as *quantum predicates* which are atoms in an Hoare-style logic. Efforts are currently made for the generation of invariants in QHL [14, 16] and theorem proving for checking QHL formulas in Isabelle/HOL [10]. The shortcoming of the QHL approach lies in the restrictive specification formalism, as predicates are limited to positive operators.

1.1 Our goal

Our goal is to answer the shortcomings of the existing approaches and provide a scalable verification framework for quantum programming. In particular, we aim at

*This work was supported by the French National Research Agency (ANR), project SoftQPro, ANR-17-CE25-0009

- intuitive specifications with a clear separation between type and specification,
- support for proof automation,
- and reuse of best practices in classical program verification.

Our approach relies on the embedding of the dedicated language and logic in a robust tool that have been used in conventional, industrial settings: Why3 [2, 6]. Why3 has been designed for proving properties of classical programs written in several real-life languages: algorithm [3], C, Java [5], ADA through traduction [7, 9].

1.2 Why3 in a nutshell

In deductive program verification, programs are decorated with assertions such as pre and post-conditions for functions or loop invariants. These decorations form contracts for the user, ensuring that, providing the inputs of a function respects this function preconditions, then its outputs respect the post condition. That the execution of the function on a preconditioned input ensures the postcondition constitutes a proof obligation for the developer.

The Why3 tool [2, 5] is a verification environment featuring an ML like language both for programming and for writing specifications. From a specified language, it generates a set of proof obligations to be fulfilled for the certification of a program. These proofs can be manipulated through a dedicated GUI .

Why3 also provides an interactive proof assistant. To prove a theorem thanks to Why3, one can indeed call for a set of *automatic* SMT solvers (CV, Z3, Alt-ergo...) accessible from the GUI. One can also help the solvers by indicating proof steps, or proposing proof simplifications. The proof steps could be expressed using a Why3 program, so even if it is possible, it is exceptional to resort to send the proof obligation to a proof assistant such as Coq.

2 Why3 for certifying quantum programming

We believe that Why3 offers a particularly good user experience as the pre-post conditions formalism is particularly intuitive and fits well with an interactive process of program specifications and verifications, as witnessed by several industrial success stories [9].

The current state of the work consists in the design of a core functional programing language embedded in Why3 language: a minimum circuit description language with a matrix-based semantics written and certified in Why3, using an algebra library, offering the standard Hilbert space and matrix formalism of quantum computing.

So far, it consists in approximatively 2000 lines of Why3 code. We have written the algebra library, with basic matrix operations: addition, scalar product, matrix product and tensor product. These are used to develop a set of high-level reasoning facilities to prove the natural properties on these objects.

The next steps are to link the library with the circuit-description language to develop a full verification framework. In parallel, we plan to validate the overall concepts and choices on standard quantum algorithms, such as Shor's factoring algorithm.

References

[1] Alexandru Baltag and Sonja Smets. Lqp: the dynamic logic of quantum information. *Mathematical structures in computer science*, 2006.

[2] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd your herd of provers. *Boogie*, 2011.

[3] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Let's verify this with why3. *STTT, 2015*, 2015.

[4] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. Proof rules for the correctness of quantum programs. *Theoretical Computer Science*, 2007.

[5] Jean Christophe Filliâtre and Claude Marché. The why/krakatoa/caduceus platform for deductive program verification. In *ICCAD*, CAV'07, 2007.

[6] Jean-Christophe Filliâtre and Andrei Paskevich. Why3—where programs meet provers. In *European Symposium on Programming*. Springer, 2013.

[7] Clément Fumex, Claude Marché, and Yannick Moy. Automating the verification of floating-point programs. In *VSTTE*, 2017.

[8] Simon J Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. QMC: A model checker for quantum systems. In *ICCAD*, 2008.

[9] Duc Hoang, Yannick Moy, Angela Wallenburg, and Roderick Chapman. Spark 2014 and gnatprove. *STTT*, 17(6), 2015.

[10] Tao Liu, Yangjia Li, Shuling Wang, Mingsheng Ying, and Naijun Zhan. A theorem prover for quantum hoare logic and its applications. *arXiv:1601.03835*, 2016.

[11] Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. *ACM SIGPLAN Notices*, 2017.

[12] Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice: Formal verification of quantum circuits in coq. *arXiv:1803.00699*, 2018.

[13] Mingsheng Ying. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6):19, 2011.

[14] Mingsheng Ying. Toward automatic verification of quantum programs. *Formal Aspects of Computing*, pages 1–23, 2016.

[15] Mingsheng Ying, Yangjia Li, Nengkun Yu, and Yuan Feng. Model-checking linear-time properties of quantum systems. *ACM TOCL*, 2014.

[16] Mingsheng Ying, Shenggang Ying, and Xiaodi Wu. Invariants of quantum programs: characterisations and generation. *ACM SIGPLAN Notices*, 2017.