Report from Dagstuhl Seminar 18381

Quantum Programming Languages

Edited by

Michele Mosca[1], Martin Roetteler[2], and Peter Selinger[3]

1 **University of Waterloo, CA,** michele.mosca@uwaterloo.ca
2 **Microsoft Corporation – Redmond, US,** martinro@microsoft.com
3 **Dalhousie University – Halifax, CA,** selinger@mathstat.dal.ca

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 18381 "Quantum Programming Languages", which brought together researchers from quantum computing and classical programming languages.

Seminar September 16–21, 2018 – http://www.dagstuhl.de/18381
2012 ACM Subject Classification Computer systems organization → Quantum computing, Hardware → Quantum computation, Theory of computation → Quantum computation theory
Keywords and phrases compilers, functional programming, quantum computing, reversible computing, verification
Digital Object Identifier 10.4230/DagRep.8.9.112

1 Executive Summary

Michele Mosca (University of Waterloo, CA)
Martin Roetteler (Microsoft Corporation – Redmond, US)
Peter Selinger (Dalhousie University – Halifax, CA)

This report documents the program and the outcomes of Dagstuhl Seminar 18381 "Quantum Programming Languages".

The aim of the seminar was to bring together researchers from quantum computing—in particular those focusing on quantum algorithms and quantum error correction—and classical programming languages. Open questions that were of interest to this group include new methods for circuit synthesis and optimization, compiler optimizations and rewriting, embedded languages versus non-embedded languages, implementations of type systems and error reporting for quantum languages, techniques for verifying the correctness of quantum programs, and new techniques for compiling efficient circuits and protocols for fault-tolerant questions and their 2D layout.

Quantum computing is getting real. Several laboratories around the world are implementing hardware platforms. For instance, systems based on superconducting qubits, such as those at IBM, Google, Intel, the University of Maryland, ionQ, and Rigetti are now scaling into the 50-150 qubit range.

While research on the theoretical side of the field addressed fundamental questions such as how to best leverage this new model of computation for algorithmic applications, a topic that has received significantly less attention is how to actually program quantum computers. To take advantage of the immense computing power offered by quantum computers as they

come online in the coming years, software tools will be essential. We want these tools to be available, efficient and reliable, so that we can quickly and reliably reap the positive benefits that quantum computers have to offer.

It is clear that quantum programming will require tools for automatically generating large-scale circuits and for synthesizing circuits from elementary fault-tolerant gates which then can be carried out by a future quantum computer. However, it is less clear what the best way will be to go about these challenging issues. Questions that were discussed at the seminar include the following:

- How can we program a quantum computer? What are the basic structures that a language should support and how can a compiler help a user develop abstract/high-level reasoning about algorithms?
- How do we model the underlying instruction set? As currently the underlying hardware is quickly evolving, how can we best model a fault-tolerant quantum computer?
- How to compile and optimize quantum programs? Automatic translation of high-level programs into circuits will be key to program quantum computers. How to design good tools for this?
- How to we test and verify quantum programs? Given that it is hard for classical computers to simulate the time evolution of a quantum computer, how can we ascertain correctness of a circuit?

The seminar brought together some 44 researchers with diverse skill sets from quantum computing, mathematical foundations of programming languages, implementation of programming languages, and formal verification. The seminar consisted of 23 talks, as well as a number of vibrant discussion sessions and a software demonstration session. The sessions where:

- Wine Cellar discussion, moderated by Sabine Glesner. This was our first discussion session. We discussed the questions raised by Sabine Glesner during her talk: Why do we need quantum programming languages? Which "killer applications" would make quantum programming languages successful? What are appropriate abstractions from quantum hardware? What are theoretical models for quantum computing?
- Discussion session on Debugging, moderated by Rodney Van Meter. This session focused on what are appropriate debugging techniques for quantum computing. The issue arises because the most common classical debugging technique, setting break points and examining the program state, cannot be applied in the context of quantum computing.
- Discussion session on Challenge Problems for Quantum Computing, moderated by Earl Campbell. In this session, we discussed coming up with well-defined problems with some success quantifier for quantum computation, similar to the successful SAT competitions.
- Group survey session on a Bird's Eye View on Quantum Languages, moderated by Robert Rand. In this session, the group compiled a list of all quantum programming languages and toolkits we are currently aware of, and classified them according to various criteria, for example, whether the languages are imperative or functional, whether the computational paradigm is circuit generation or Knill's QRAM model, whether the language is high-level or assembly, whether it supports type-safety and/or verification, etc.
- Group survey session on Tools for Quantum Optimization, moderated by Matthew Amy. In this session, the group compiled a list of available tools for optimization of quantum circuits.
- Group discussion on Opportunities for Education and Outreach, moderated by Rodney Van Meter. The discussion centered on new opportunities for public outreach and education that are enabled by the emergence of new quantum tools.

- Software demonstration session, moderated by Martin Roetteler. In this session, 10 researchers gave rapid demonstrations, of a about 10 minutes each, of various software tools they have designed.

Most of the participants rated the seminar as a success. We managed to connect researchers from different communities, and engaged in a vibrant exchange of novel ideas, and started to tackle important problems such as the analysis of quantum algorithms for real-world computational problems, compiler optimizations, reversible computing, and fault-tolerant quantum computing.

2 Table of Contents

3 Overview of Talks

3.1 Functional Verification of Quantum Circuits

Matthew Amy (University of Waterloo, CA)

Main reference Matthew Amy: "Towards Large-scale Functional Verification of Universal Quantum Circuits", CoRR, Vol. abs/1805.06908, 2018.
URL https://arxiv.org/abs/1805.06908

We introduce a framework for the formal specification and verification of quantum circuits based on the Feynman path integral. Our formalism, built around exponential sums of polynomial functions, provides a structured and natural way of specifying quantum operations, particularly for quantum implementations of classical functions. Verification of circuits over all levels of the Clifford hierarchy with respect to either a specification or reference circuit is enabled by a novel rewrite system for exponential sums with free variables.

We evaluate our methods by performing automated verification of optimized Clifford+T circuits with up to 100 qubits and thousands of T gates, as well as the functional verification of quantum algorithms using hundreds of qubits. We further show that our method can perform the simulation of a Hidden Shift algorithm due to Roetteler with 100 qubits in just minutes on a tablet computer.

3.2 Phase polynomials, T-count optimisation and Lempel's algorithm

Earl Campbell (University of Sheffield, GB)

Main reference Luke E Heyfron, Earl T Campbell: "An efficient quantum compiler that reduces T count", in Quantum Science and Technology, Vol. 4 (1), p. 015004, 2018.
URL https://doi.org/10.1088/2058-9565/aad604

I review the basics of the phase polynomials and the connection to T-count optimisation, summarising the work of Amy and Mosca as well as my own work on this with Luke Heyfron. This leads to a 3-tensor optimisation problem that is hard. But is it closely related to an easier (relaxed) optimisation problem solved in the 1970s by Lempel (https://epubs.siam.org/doi/abs/10.1137/0204014) All of my work on the problem has been based on modifying Lempel's algorithm to build a heuristic for the harder quantum problem. Rather than getting into the details of the hard quantum problem, I sketch Lempel's method as I believe it is not well known and could have further applications in the field.

3.3 Low overhead quantum computation using lattice surgery

Austin G. Fowler (Google Research – Mountain View, US)

The surface code is a method of detecting errors in a quantum computer. Many different methods of computing using this code exist. We fully analyze lattice surgery and show that this method is unambiguously better than braiding defects, the previous standard method.

3.4 Dependent types in Proto-Quipper

Frank Fu (Dalhousie University – Halifax, CA)

Joint work of Peter Selinger
Main reference Francisco Rios, Peter Selinger: "A categorical model for a quantum circuit description language", in Proc. of the 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017., EPTCS, Vol. 266, pp. 164–178, 2017.
URL https://doi.org/10.4204/EPTCS.266.11

Are dependent types useful for quantum circuit programming? In this talk, I will present an implementation of dependently typed Proto-Quipper, a stand-alone language for quantum circuit description. I will argue that dependent types are useful in three aspects:

1. Precise types for quantum circuit description functions.
2. Precise notion of boxing a family of circuits.
3. Encapsulating unused wires via existential dependent data types.

3.5 Reflections on what programming languages are good for – traditionally and in the face of quantum computing

Sabine Glesner (TU Berlin, DE)

When I first heard about quantum computing, which was in 1994, it was at about the same time I first heard about internet browsers. At that time, I could not tell which of these two developments would be faster. Today we know that it was not quantum computing. Nevertheless, quantum computing has made enormous progress during the last years, so much that we even have a Dagstuhl seminar on quantum programming languages. In this talk, I want to sum up what traditional programming languages have been good for and raise the question of what the situation looks like for quantum computing.

3.6 Reversible Programming Languages – From Classical Results to Recent Developments

Robert Glück (University of Copenhagen, DK)

Joint work of Martin Holm Cservenka, Robert Glück, Tue Haulund, Torben Ægidius Mogensen
Main reference Martin Holm Cservenka, Robert Glück, Tue Haulund, Torben Ægidius Mogensen: "Data Structures and Dynamic Memory Management in Reversible Languages", in Proc. of the 10th International Conference on Reversible Computation, RC 2018, Leicester, UK, September 12–14, 2018, Lecture Notes in Computer Science, Vol. 11106, pp. 269–285, Springer, 2018.
URL https://doi.org/10.1007/978-3-319-99498-7_19

This talk highlighted the principles and main ideas of reversible programming languages with which we have been working for the past several years (the "Copenhagen Interpretation of Reversible Computing"). Reversible languages form their own distinct class of programming languages because they are deterministic in both computation directions. They complement

the mainstream programming languages like C and Haskell that are backward nondeterministic. Recent developments with dynamic memory management allowed the design and implementation of reversible object-oriented and functional languages. This enables, for the first time, the reversible manipulation of high-level dynamic data abstractions such as binary trees, lists and queues.

3.7 Quantum Linguistic Relativity

Christopher Granade (Microsoft Corporation – Redmond, US)

In this talk, I will consider a set of goals for new quantum programming languages, motivated by applications and with an eye to making it easier for new quantum programmers to get started. To meet these goals, I propose thinking of quantum language design in terms of linguistic relativity, the hypothesis that the language in which we express an idea affects how we think about that idea. Finally, I present Q# as a case study for this approach to design, and discuss how we chose Q# features according to linguistic relativity.

3.8 The OpenQL programming framework

Nader Khammassi (TU Delft, NL)

Quantum computing is rapidly evolving, especially after the discovery of several efficient quantum algorithms solving intractable classical problems. Expressing these quantum algorithms using a high-level programming language and making them executable on a quantum processor, while abstracting hardware details and targetting different qubit technologies, is an important problem. After discussing the different compilation challenges, we present the OpenQL programming framework, and show how its modular design allows the integration of a full-stack quantum computer architecture for different qubit technologies.

3.9 Cheaper alternative to Euler decomposition for SU(2) gates and fall-back circuits

Vadym Kliuchnikov (Microsoft Corporation – Redmond, US)

We give an alternative to Euler decomposition that leads to average case circuit complexity scaling as $7\log_5(1/\varepsilon)$ as opposed to $9\log_5(1/\varepsilon)$ for Pauli+V gate sets. The idea readily generalizes to many other gate sets.

3.10 Operator algebras and their role in quantum programming languages

Albertus Johannis Lindenhovius (Tulane University – New Orleans, US)

Quantum systems are usually described in the Hilbert space formalism. Operator algebras, which were introduced by von Neumann, form an alternative formalism with several advantages over the Hilbert space formalism, such as the possibility of describing the interaction between quantum and classical phenomena in one framework.

We discuss how operator algebras, which are algebras of operators on a Hilbert space, can be used in the semantics of quantum programming languages. Furthermore, we discuss which categorical properties of a certain class of operator algebras correspond to what features of the quantum programming language for which it is used in the semantics. Can we single out one class of operator algebras that has all categorical properties sufficient for a higher-order quantum programming language with recursion?

3.11 NISQ optimization for CNOT and CNOT+T circuits

Beatrice Nash (MIT – Cambridge, US)

Near-term quantum devices have limited physical qubit connectivity, and performing operations between non-adjacent qubits can be very expensive. In this talk, I will discuss ways to extend current circuit optimization methods to take into account these restrictions.

3.12 Verified Quantum Programming in QWIRE: Optimization and Error Correction

Robert Rand (University of Maryland – College Park, US)

Joint work of Robert Rand, Jennifer Paykin, Dong-Ho Lee, Steve Zdancewic, Kesha Hietala, Michael Hicks, Xiaodi Wu
Main reference Jennifer Paykin, Robert Rand, Steve Zdancewic: "QWIRE: a core language for quantum circuits", in Proc. of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, pp. 846–858, ACM, 2017.
URL https://doi.org/10.1145/3009837.3009894

We present QWIRE, a quantum circuit language and formal verification tool. QWIRE possesses a denotational semantics in terms of density matrices and is embedded in the Coq proof assistant, allowing us to check our quantum circuits against their mathematical specifications. In this talk, we look at a variety of proofs of circuit specifications in Coq. We then examine two pressing issues for quantum programming: Verified optimization and error-aware semantics and the challenges of incorporating them in QWIRE.

3.13 Proto-Quipper-M: A Categorically Sound Quantum Circuit Description Language.

Francisco Rios (Dalhousie University – Halifax, CA)

Joint work of Francisco Rios, Peter Selinger
Main reference Francisco Rios, Peter Selinger: "A categorical model for a quantum circuit description language", in Proc. of the 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017., EPTCS, Vol. 266, pp. 164–178, 2017.
URL https://doi.org/10.4204/EPTCS.266.11

Quipper is a practical programming language for describing families of quantum circuits. In this talk, we formalize a small, but useful fragment of Quipper called Proto-Quipper-M. Unlike its parent Quipper, this language is type-safe and has a formal denotational and operational semantics. Proto-Quipper-M is also more general than Quipper, in that it can describe families of morphisms in any symmetric monoidal category, of which quantum circuits are but one example. We design Proto-Quipper-M from the ground up, by first giving a general categorical model of parameters and states. After finding some interesting categorical structures in the model, we then define the programming language to fit the model. We cement the connection between the language and the model by proving type safety, soundness, and adequacy properties.

3.14 Toward the first quantum simulation with quantum speedup

Neil Julien Ross (Dalhousie University – Halifax, CA)

Joint work of Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, Yuan Su

As we approach the development of a quantum computer with tens of well-controlled qubits, it is natural to ask what can be done with such a device. Specifically, we would like to construct an example of a practical problem that is beyond the reach of classical computers, but that requires the fewest possible resources to solve on a quantum computer. We address this problem by considering quantum simulation of spin systems, a task that could be applied to understand phenomena in condensed matter physics such as many-body localization. We synthesize explicit quantum circuits for three leading quantum simulation algorithms, one based on product formulas (PF), one based on implementing the Taylor series as a linear combination of unitaries (TS), and another using the recent quantum signal processing approach (QSP). We employ a wide range of techniques to develop tighter error bounds and optimize gate-level implementations. Surprisingly, even for simulations of small systems, we find that the fourth-order PF algorithm outperforms lower-order PF algorithms and that the TS and QSP algorithms require even fewer gates (although at the cost of requiring more qubits). However, the cost of PF algorithms can be reduced significantly by using empirical error bounds, so that PF algorithms remain competitive in contexts where a rigorous guarantee on the accuracy of the simulation is not essential. Our circuits are smaller by several orders of magnitude than those for the simplest classically-hard instances of problems such as factoring and quantum chemistry, and we hope they will pave the way toward the first practical application of a quantum computer.

3.15 Automatic Synthesis in Quantum Programming Languages

Mathias Soeken (EPFL – Lausanne, CH)

Joint work of Mathias Soeken, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Giovanni De Micheli, Martin Roetteler, Thomas Häner
Main reference Mathias Soeken, Thomas Häner, Martin Roetteler: "Programming quantum computers using design automation", in Proc. of the 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018, pp. 137–146, IEEE, 2018.
URL https://doi.org/10.23919/DATE.2018.8341993

When translating quantum programs into (technology-dependent) quantum circuits, we are often confronted with translating conventional classical logic in forms of permutations, Boolean functions, or logic networks. RevKit is a toolkit that combines several state-of-the-art approaches for synthesis, optimization, and mapping. RevKit is powered by the EPFL logic synthesis libraries, such as tweedledum, alice, mockturtle, and kitty.

In the presentation, we showcase several applications of RevKit in modern quantum programming flows. The examples include integrations with Qiskit, ProjectQ, pyQuil, Q#, and Cirq.

3.16 Representing quantum control

Benoit Valiron (Centrale Supelec – Orsay, FR)

Joint work of Amr Sabry, Benoît Valiron, Juliana Kaizer Vizzotto
Main reference Amr Sabry, Benoît Valiron, Juliana Kaizer Vizzotto: "From Symmetric Pattern-Matching to Quantum Control", in Proc. of the 21st International Conference on the Foundations of Software Science and Computation Structures, FOSSACS 2018, Thessaloniki, Greece, April 14–20, 2018, Lecture Notes in Computer Science, Vol. 10803, pp. 348–364, Springer, 2018.
URL https://doi.org/10.1007/978-3-319-89366-2_19

One perspective on quantum algorithms is that they are classical algorithms having access to a special kind of memory with exotic properties. This perspective suggests that, even in the case of quantum algorithms, the control flow notions of sequencing, conditionals, loops, and recursion are entirely classical. There is however, another notion of control flow, that is itself quantum. The notion of quantum conditional expression is reasonably well-understood: the execution of the two expressions becomes itself a superposition of executions. The quantum counterpart of loops and recursion is however not believed to be meaningful in its most general form. In this talk (based on [1]), we discuss how, under the right circumstances, a reasonable notion of quantum loops and recursion is possible. To this aim, we first propose a classical, typed, reversible language with lists and fixpoints based on Theseus [2]. We then extend this language to the closed quantum domain (without measurements) by allowing linear combinations of terms and restricting fixpoints to structurally recursive fixpoints whose termination proofs match the proofs of convergence of sequences in infinite-dimensional Hilbert spaces. We additionally give an operational semantics for the quantum language in the spirit of algebraic lambda-calculi. This permits to reconciliate several approaches, such as the quantum tests of QML [3], Ying's quantum loops [4] and linear algebraic approaches [5]

References

1 Sabry, A., Valiron, B., and Vizzotto, J. K. (2018). From Symmetric Pattern-Matching to Quantum Control. In International Conference on Foundations of Software Science and Computation Structures (pp. 348–364). Springer.
2 James, R. P., and Sabry, A. (2014). Theseus: a high-level language for reversible computing. Unpublished manuscript.
3 Altenkirch, T., and Grattage, J. (2005). A Functional Quantum Programming Language. In P. Panangaden, Proceedings of the 20th Symposium on Logic in Computer Science (LICS'05) (pp. 249–258). Chicago, Illinois, US. IEEE Computer Society Press.
4 Ying, M. (2016). Foundations of quantum programming. Morgan Kaufmann.
5 Arrighi, P., and Dowek, G. (2008). Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In A. Voronkov, Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA'08) (pp. 17–31). Hagenberg, Austria: Springer.

3.17 Error-aware compilation for the IBM 20-qubit machine

Rodney Van Meter (Keio University – Fujisawa, JP)

Joint work of Yulu Pan, Shin Nishio, Takahiko Satoh, Rodney Van Meter

Roughly, our first project [1] is to characterize qubits, operation fidelity, and path fidelity for moving qubits (or doing long-distance gates), and the second [2] is to take that information and place qubits on processor and plan their movement.

While various projects have worked on compiling programs to meet the constraints of small quantum processors, we believe this is the first work to focus on the inhomogeneity in actual gate errors due to minor differences between qubits as the key metric for placing variable qubits on the physical qubits on the processor.

The error rates for qubits and gates are measured using random benchmarking. For the placement phase, we are using beam search combined with a modified form of Dijkstra's shortest path first, and using the product of gate fidelities as our prediction for success probability on complex circuits. We use the Cuccaro adder circuit as our application for testing the compilation, and KL-divergence as a measure of the quality of circuits. We compared the existing QISkit compiler with QOPTER.

Our work, while not yet complete, suggests that the single number of gate fidelity is not an adequate measure of success rate. We matched the QISkit success probability while using minimal computational resources in the compilation phase for a 5-qubit adder circuit, but the actual success probability is still low, and the estimated success probability actually far lower. The relative ranking of choice of circuit shows an intermediate level of correlation with the ordering of success probability in circuits. Thus, this complex problem is still ripe for new approaches and hard work, and our future work includes continuing development of these tools.

References

1 Shin Nishio, Yulu Pan, Takahiko Satoh, Rodney Van Meter. "High Fidelity Qubit Mapping for IBM Q," *Proc. 2nd International Workshop on Quantum Compilation*, 2018

2 Yulu Pan, Shin Nishio, Takahiko Satoh, Rodney Van Meter, Hideharu Amano. "QOPTER –Quantum program OPTimizER– ," *Proc. 2nd International Workshop on Quantum Compilation*, 2018

3.18 Data-structures and Methods for the Design of Quantum Computations

Robert Wille (Johannes Kepler Universität Linz, AT)

Joint work of Alwin Zulehner, Robert Wille
Main reference Alwin Zulehner, Robert Wille: "Advanced Simulation of Quantum Computations", TCAD, 2018.
URL https://doi.org/10.1109/TCAD.2018.2834427

In the past decades, the Computer-Aided Design (CAD) community was frequently faced with tremendously complex challenges that often required the efficient consideration of problems of exponential (or even greater) size. In order to tackle these, researchers and engineers developed sophisticated CAD methods employing, e.g., decision diagrams or sophisticated reasoning engines. In contrast, many design problems in the quantum domain are still addressed in a rather straight-forward fashion, e.g., by exponential array-based descriptions or enumerative search algorithms. This talk illustrates how established concepts from the conventional design of circuits and systems can be applied to improve the design of quantum computations. By this, the talk will "bridge" the CAD and the quantum communities by showing how the combination of expertise from both domains eventually yields efficient design methods for quantum computation. The application of those data-structures and methods is exemplarily demonstrated by means of simulation of quantum computation.

3.19 Logic level circuit optimization for topological quantum computation

Shigeru Yamashita (Ritsumeikan University – Shiga, JP)

The TQC (Topological Quantum Computing) model has been receiving a lot of attention because it has proven to be one of the most promising fault-tolerant quantum computation models. In the TQC conceptual model, we arrange physical measurement sequences corresponding to computational steps of quantum computation in a three-dimensional space. While some transformation rules for this arranged three-dimensional space have been known, there was no known systematic way to use the rules to optimize the arranged space. This talk proposes an efficient systematic way to use the known transformation rules by considering the arranged space as a set of loops.

3.20 Reasoning about Parallel Quantum Programs

Mingsheng Ying (University of Technology – Sydney, AU)

We initiate the study of parallel quantum programming by defining the operational and denotational semantics of parallel quantum programs. The technical contributions include: (1) finding a series of useful proof rules for reasoning about correctness of parallel quantum programs; and (2) proving a strong soundness theorem of these proof rules, showing that partial correctness is well maintained at each step of transitions in the operational semantics of a parallel quantum program. This is achieved by partially overcoming the following conceptual challenges that are never present in classical parallel programming: (i) the intertwining of nondeterminism caused by quantum measurements and introduced by parallelism; (ii) entanglement between component quantum programs; and (iii) combining quantum predicates in the overlap of state Hilbert spaces of component quantum programs with shared variables. It seems that a full solution to these challenges and developing a (relatively) complete proof system for parallel quantum programs are still far beyond the current reach.

3.21 Recursive types for linear/non-linear quantum programming

Vladimir Zamdzhiev (LORIA – Nancy, FR)

Linear/non-linear lambda calculi provide a natural framework for quantum programming. By making a distinction between intuitionistic (non-linear / classical) and linear types, we may model classical data and quantum data. The latter cannot be copied or deleted, which is conveniently ensured by the linearity of the type system, whereas the former may be freely copied and discarded, which is also conveniently allowed by the non-linear part of the type system.

In this talk, we consider the problem of extending such a lambda calculus with recursive types. We design the type system such that we may distinguish between intuitionistic recursive types and linear recursive types. We also describe some work in progress on a conjectured denotational model that soundly models our lambda calculus.

3.22 Quantum Calculi: from theory to language design

Margherita Zorzi (University of Verona, IT)

Joint work of Margherita Zorzi, Andrea Masini, Ugo Dal Lago, Luca Paolini, Luca Roversi
Main reference Margherita Zorzi: "On quantum lambda calculi: a foundational perspective", Mathematical Structures in Computer Science, Vol. 26(7), pp. 1107–1195, 2016.
URL https://doi.org/10.1017/S0960129514000425

In the last 20 years several approaches to quantum programming have been introduced. In this report we will focus on functional calculi and in particular on the QRAM architectural model. We explore the twofold perspective (theoretical and concrete) of the approach and we will list the main problems one has to face in quantum language design.

3.23 Compiling Quantum Circuits to NISQ Devices

Alwin Zulehner (Johannes Kepler Universität Linz, AT)

Joint work of Alwin Zulehner, Alexandru Paler, Robert Wille
Main reference Alwin Zulehner, Alexandru Paler, Robert Wille: "An efficient methodology for mapping quantum circuits to the IBM QX architectures", to appear in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.
URL https://doi.org/10.1109/TCAD.2018.2846658

The Noisy Intermediate-Scale Quantum (NISQ) technology is currently investigated by major players in the field to build the first practically useful quantum computer. IBM QX architectures are the first ones which are already publicly available today. However, in order to use them, the respective quantum circuits have to be compiled for the respectively used target architecture. This demands solutions for automatically and efficiently conducting this compilation process. In this work, we offer solutions to this problem that satisfy all constraints given by the architecture and, at the same time, aim to keep the overhead in terms of additionally required quantum gates minimal. Our experimental evaluation shows that the proposed approach significantly outperforms IBM's own solution regarding fidelity of the compiled circuit as well as runtime. Moreover, to emphasize development, IBM launched a challenge with the goal to optimize such compilers for a certain set of random quantum circuits. Since these circuits represent a worst case scenario for our approach, we developed a correspondingly adjusted version. It has been declared the winner of this so-called QISKit developer challenge, since it yields compiled circuits with at least 10% better costs than the other submissions, while generating them at least 6 times faster (according to IBM). Implementations of the proposed methodologies are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

4 Working groups

4.1 Tools for Quantum Optimization

Matthew Amy (University of Waterloo, CA)

In this session, the group compiled a list of available tools for optimization of quantum circuits. We came up with the following list of tools. While the list is probably highly incomplete, we hope that it is a useful starting point.

- **Feynman,** by Matthew Amy. Optimizations: z-rotation optimization, CNOT-count optimization, T-depth optimization. Language: Haskell library, command-line interface. License: BSD-2. Availability: Github.
- **Newsynth/Gridsynth,** by Neil J. Ross and Peter Selinger. Optimizations: single-qubit Z-rotations to Clifford+T. Language: Haskell library, command-line interface. License: GPL-3. Availability: Hackage.
- **TOpt,** by Earl Campbell. Optimizations: Clifford+T to Clifford+T, T-gate minimization. Language: C++, command-line interface. License: GPL-3. Availability: Github.
- **IonQ's tool,** by IonQ. Optimizations: Clifford+z-rotations+Toffoli to Clifford+z-rotations. Language: Fortran. License: Proprietary.

- **RevKit,** by Mathias Soeken. Optimizations: Look-up table hierarchical reversible synthesis (LHRS). CNOT minimization. Produces Clifford+T. Language: C++, command-line interface. Python bindings. License: MIT.
- **pQCS,** by Olivia Di Matteo and Michele Mosca. Optimizations: Multi-qubit unitary to Clifford+T. Availability: from https://qsoft.iqc.uwaterloo.ca/. License: For research only.
- **IBM QX mapping SU(4),** by Alwin Zulehner and Robert Wille. Optimizations: $SU(4)$ to CNOT+SWAP+1-qubit gates. Availability: Github. License: Non-commercial use only.

4.2 Challenge problems in quantum computation

Earl Campbell (University of Sheffield, GB)

We discussed possible "challenge problems" in the optimisation of quantum circuits. The idea was to come up with very well defined problems with some success quantifer, similar to the successful SAT competitions. The most popular idea was to consider Hamiltonian simulation of small systems of 10–20 qubits and minimise the number of gates required to achieved a precision of 10^{-3}. Suggested Hamiltonians included:

1. The 1D Heisenberg chain;
2. Jellium;
3. Quantum chemistry hamiltonians available in the openFermion packages.

Given a Hamiltonian, one could consider implementing the operator e^{iHt}, but an additional interesting problem is phase estimation, where one implemented a controlled e^{iHt}. Here, t and the number of circuit repetitions ought to be optimised to minimise the Fisher information of the parameter estimated.

4.3 Wine Cellar Discussion on Quantum Programming Languages

Sabine Glesner (TU Berlin, DE)

Joint work of all the participants of the Dagstuhl seminar 18381

This is an abbreviated summary of a discussion in the Dagstuhl wine cellar that took place during the Dagstuhl seminar 18381 on Quantum Programming Languages on Monday, September 20, 2018. The starting point for the discussion was the list of questions raised in Sabine Glesner's talk.

4.3.1 What are programming languages good for, classically?

Programming languages are central in computer science. Already a brief look at the Turing award winners and their research areas reveals that programming languages never go out of date. In the past, programming languages have been helpful to abstract from hardware details, which gives more programming comfort to software developers. Type systems

have been developed to enhance program correctness by allowing programmers to detect misfittings statically before the program is executed. Also, data management is important and programming languages offer a rich variety of structured data mechanisms, e.g., via class hierarchies. There are many programming language paradigms around (e.g., functional, imperative, object-oriented, logical, etc., as well as combinations thereof). It turns out that each of them supports a certain class of problems. A definite achievement in the area of programming languages is program analyses. They analyse programs statically, and are usually conservative. For example, live variables analysis will find variables that are definitely no longer used, but might err on the side of marking a variable as live when it *might* be live.

4.3.2 What are killer applications for quantum computing?

History has shown that new technologies are usually only successful if they are necessary to accomplish something new (a "killer application"). We brainstormed on what the likely killer applications for quantum computing would be. It was suggested that the field of *quantum chemistry* has many well-formulated and ready-made problems for which quantum computing will be very helpful, and that this will be one of the first "real" applications of quantum computing. Solving problems in quantum chemistry has potential real-world applications, such as the discovery of new catalysts to make chemical reactions more efficient (e.g., carbon capture, the production of fertilizers, or the conversion of solar energy into fuel). Also, physical qubits can be used to build up quantum sensors, which may be another early application. The participants generally agreed that cryptanalysis is not likely to be a killer application for quantum computing, both because it requires a relatively large number (compared to quantum chemistry) of fault-tolerant qubits, and also because the world will switch to post-quantum cryptography as soon as current protocols become insecure.

4.3.3 Quantum hardware

We discussed the size of current actual quantum hardware, the difference between physical and logical qubits, and their respective error rates, which are significantly higher for qubits (10^{-3}) compared to classical transistors (10^{-14}).

4.3.4 Quantum programming languages

Quantum programming languages are not necessarily complete stand-alone languages but often libraries or packages built on top of classical languages. OpenFermion is an example for such a package which is itself implemented in Python. The focus in the development of programming languages is often on the translation, which is often targeted to quantum circuits. A major concern when running quantum programs is the appearance and accumulation of errors. The longer quantum hardware runs, the higher the error rate is. Hence, it is important to understand how error correction can be done. It would be very helpful if such analyses could be done automatically. While there are lots of analyses around, still a major amount of uncertainty comes from the inputs. It is also an interesting question what level of error is tolerable.

It would be good to have benchmarks so that optimizations could be developed (see the analogy for SAT/SMT solvers for which also benchmarks exist). We pursued this question further in a separate working group "Challenge problems in quantum computation".

Debugging is another important and very difficult issue in quantum programming languages, as there are no checkpoints available. We pursued this question further in a separate working group on "Debugging".

4.3.5 Theoretical Models for Quantum Computing

This point did not receive much discussion, as it was generally agreed that Quantum Turing Machines are considered to be too cumbersome to work with, while quantum circuits are typically the formalism of choice, at least during our discussion.

4.4 Survey of Quantum Languages

Robert Rand (University of Maryland – College Park, US)

The purpose of this session was to compile a list of quantum programming languages and toolkits of interest, and to classify them according to various criteria. We considered the following languages:

Proto-Quipper	F	C	H	Ac	T/L	As	St	V (partial)	O/D/K	—
QWIRE	F	CF	H	Ac	T/L	As	E	V	D/K	—
Quipper	F	CF	H	G	T (partial)	As	E	—	—	In
ProjectQ	I	N	H	G	—	?	E	—	—	In/De (partial)
Q#	I/F	N	H	G	T (partial)	As	St	—	—	—
PyQuil	I	N	H	G	—	—	E	—	—	De
Cirq	I	C	H	S	—	?	E	—	—	—
QISKit	I	CL	H	G	—	—	E	—	—	In/De
Scaffold	I	?	H	G	—	—	St	—	—	In
(Open)QASM	I	CL	A	G	—	—	Ta	—	—	—
Quil	I	N	A	G	—	—	Ta	—	O	—

The letters after each language mean the following:

- Language paradigm: Functional (F), imperative (I).
- Target: Circuit based (C), circuits with feedback from measurements (CF), circuits with limited feedback (CL), or not circuit based (N).
- Abstraction: High-level language (H) or assembly-type language (A).
- Intended audience: General public (G), academic research (Ac), or special-purpose (S).
- Safety: type-safety (T), linearity (L).
- Run-time checks: assertions (As).
- Implementation: embedded language (E), standalone language (St), or target language (Ta).
- Support for verification (V).
- Semantics: operational (O), denotational (D), and/or categorical (K).
- Support for optimization: machine dependent (De) or machine independent (In).

4.5 Software demonstration session

Martin Roetteler (Microsoft Corporation – Redmond, US)

In this session, researchers gave rapid demonstrations of various software tools they have designed. The following is a list of the presentations, which lasted about 10 minutes each.

- Vadym Kliuchnikov: Asserts, unit tests, and Q# tracer tool
- Damian Steiger: ProjectQ: Shor's algorithm, quantum chemistry, rendering
- Frank Fu: Dependent types in Proto-Quipper
- Nader Khammassi: QASM generator
- Chris Granade: Quantum Katas in Q#
- Matt Amy: Feynman tool to optimize and verify quantum circuits
- Andrew Cross: IBM Quantum Experience demo
- Bruno Schmitt: Tweedledee and tweedledum: IRs for RevKit
- Robert Rand: QWIRE proofs in Coq
- Alwin Zulehner: Simulation with QMDDs

4.6 Debugging of Quantum Programs

Rodney Van Meter (Keio University – Fujisawa, JP)

This session focused on identifying appropriate debugging techniques for quantum computing. The issue arises because the most common classical debugging technique, setting break points and examining the program state, cannot be used in the context of quantum computing.

It was suggested that we must draw a distinction between debugging and program verification. In fact, debugging may potentially be used to answer three different questions:

- Specification: did we define the algorithm correctly?
- Code: have we correctly translated the specification to working, bug-free code?
- Runtime: does the simulator or real quantum computer execute the program as expected?

According to the experience of some former members of the IARPA QCS program, typical quantum circuits contain large classical subcircuits (usually oracles, sometimes more than 100× the size of the quantum portion). It may be beneficial to debug these parts separately, as they can be simulated efficiently.

Here is an incomplete list of classical debugging techniques. Some of these may be applicable to quantum programming, although others will not be.

- assertions (invariants in the program)
- interactive debugging (breakpoints)
- time travel debuggers (those that can step backwards in time from a crash)
- inserting print statements (all too common)
- unit testing (e.g., establishing whether executed gate count matches expected gate count)
- static analysis (including type checking)
- dynamic analysis (memory leaks?)

- code review
- testing with random instances
- post-mortem (what is a "quantum crash dump"?)

Additional discussion focused on the distinction between regression testing and performance testing, and on the difficulty of discriminating between hardware bugs and software bugs. The group identified the following research questions and directions:

- How applicable are probabilistic techniques? Is probabilistic model checking applicable?
- How do we debug on logical qubits, as opposed to physical qubits?
- Checking if a circuit is the identity is QMA-hard; is checking it up to ε still QMA-hard?
- What classes of properties do we want to check?
- How can we make verification tools useful to programmers?
- What concrete tools can we develop?

4.7 Opportunities for Education and Outreach

Rodney Van Meter (Keio University – Fujisawa, JP)

The discussion centered on new opportunities for public outreach and education that are enabled by the emergence of new quantum tools. In general, the audience for quantum tools can be divided into three categories:

- The general public, e.g., popular science enthusiasts (learners we hope to attract), who just want to learn the key ideas.
- Black box library users, who don't care how it works.
- Algorithmists, who will need to learn how to create new interference patterns.

The first group is the most difficult to reach. It was suggested that people in this group generally have three questions, in this order:

- What does it do?
- When will I have it?
- How does it work?

Physicists tend to answer the questions in exactly the opposite order.

Participants

- Matthew Amy
University of Waterloo, CA
- Sébastien Bardin
CEA LIST, FR
- Xiaoning Bian
Dalhousie University – Halifax, CA
- Earl Campbell
University of Sheffield, GB
- Andrew Cross
IBM TJ Watson Research Center – Yorktown Heights, US
- Olivia Di Matteo
University of Waterloo, CA
- Austin G. Fowler
Google Research – Mountain View, US
- Frank Fu
Dalhousie University – Halifax, CA
- Vlad Gheorghiu
University of Waterloo, CA
- Sabine Glesner
TU Berlin, DE
- Robert Glück
University of Copenhagen, DK
- Christopher Granade
Microsoft Corporation – Redmond, US
- Markus Grassl
Max Planck Institute for the Science of Light, DE
- Thomas Häner
ETH Zürich, CH
- Shih-Han Hung
University of Maryland – College Park, US
- Nader Khammassi
TU Delft, NL
- Vadym Kliuchnikov
Microsoft Corporation – Redmond, US
- Sriram Krishnamoorthy
Pacific Northwest National Lab. – Richland, US
- Andrew John Landahl
Sandia National Labs – Albuquerque, US
- Albertus Johannis Lindenhovius
Tulane University – New Orleans, US
- Michael W. Mislove
Tulane University – New Orleans, US
- Michele Mosca
University of Waterloo, CA
- Beatrice Nash
MIT – Cambridge, US
- Jennifer Paykin
Galois – Portland, US
- Robert Rand
University of Maryland – College Park, US
- Mathys Rennela
CWI – Amsterdam, NL
- Francisco Rios
Dalhousie University – Halifax, CA
- Martin Roetteler
Microsoft Corporation – Redmond, US
- Neil Julien Ross
Dalhousie University – Halifax, CA
- Bruno Schmitt Antunes
EPFL – Lausanne, CH
- Peter Selinger
Dalhousie University – Halifax, CA
- Mathias Soeken
EPFL – Lausanne, CH
- Damian Steiger
ETH Zürich, CH
- Rainer Steinwandt
Florida Atlantic University – Boca Raton, US
- Benoit Valiron
Centrale Supelec – Orsay, FR
- Rodney Van Meter
Keio University – Fujisawa, JP
- Michael Walter
University of Amsterdam, NL
- Robert Wille
Johannes Kepler Universität Linz, AT
- Shigeru Yamashita
Ritsumeikan University – Shiga, JP
- Mingsheng Ying
University of Technology – Sydney, AU
- Vladimir Zamdzhiev
LORIA – Nancy, FR
- Margherita Zorzi
University of Verona, IT
- Alwin Zulehner
Johannes Kepler Universität Linz, AT
- Paolo Zuliani
Newcastle University, GB