CMSC657 Final Report: Verifications of Quantum Programs: from ZX-Calculus to more

Yiyun Liu, Yuxiang Peng, James Parker

December 13, 2019

1 Introduction

This report presents a survey on the theory and tools to verify quantum programs. The modern development of quantum programs faces several major issues. First, due to the principles of quantum mechanics, some properties traditional programming languages relying on no longer holds. New theory of language design for quantum programs is necessary. Second, when we measure a quantum state, the state changes. This makes the debugging of quantum programs hard, and the essence of formal verification comes in the way. Thus it is pivotal for us to develop methods to verify quantum programs.

One of the most successful approaches is the ZX-Calculus, a graphical language which can be used for representing linear mappings for qubits. We mainly present the verifications under the context of ZX-Calculus in this report.

ZX-Calculus was motivated by the idea that the Hilbert space formalism is too low-level as a language for quantum computation[6]. Equational reasoning can be tedious and the algebraic representation cannot be easily translated to what is going on under the hood. One interesting property about ZX-Calculus is that, while it is possible to faithfully embed every quantum circuit as a ZX-Calculus diagram[16], not every diagram corresponds to a quantum circuit. Some of the matrices represented are not unitary. This limitation was briefly mentioned in [22]. However, it turns out that this property can also be useful. Real world quantum computing devices suffer from decoherence issues, thus making it necessary to find ways to perform error correction. This can be done by encoding one qubit of information using surface code. One example we found consists of a lattice of multiple qubits. In order to operate on the encoded bits, one needs to perform transversal operations. While we do not fully know why transversal operations are undesirable, there are papers that explore how to perform universal operations without transversal operations. One technique is to use lattice surgery[14], which can be easily encoded as simple ZX-Calculus diagrams[4]. The surgery operations are not unitary, but ZX-Calculus has the flexibility to encode those operations as well. The otherwise complex subject becomes much easier to understand.

We are writing a survey about the papers we have read about ZX-Calculus and other developments of quantum program verifications. ZX-Calculus has many applications, but we are putting more emphasis on its connection with category theory and quantum error correction. We have read papers that give background information on linear logic and formal quantum programming languages. In the future, we will then seek connections between the topics covered in class and ZX-Calculus by interpreting the former in the language of diagrams.

2 Related Work

In this decade, the community pays more and more attention on the programming language design and the analysis, and people tend to get full understanding of the properties of quantum programming language. The origin of quantum programming language dates back to the late 20th century, when Deustch [9] proposed the quantum Turing machine. Where there is the theory immitating Turing machine, there would be one extending -calculus. Selinger and Valiron [25] defined quantum -calculus and endow it with strong type system. These theoretical models guides the design of real language implementation.

When it comes out of theoretical models, there are lots of implementations of quantum programming language. Following the quantum -calculus, Green *et al.* [11] built up the Quipper language embedded in Haskel. It follows functional programming style, and has strong static type checking. Using boxing techniques, Quipper constructs scalable circuits for quantum algorithms. Besides, another quantum programming language Scaffold, developed by Javadi-Abhari *et al.* [15], follows C-style. Scaffold language provides a full stack funcionality, from the language itself to the compiler behind. Microsoft also provides the community with Q, a quantum programming language with quantum control flow. As for formal verifications, Paykin *et al.* [20] proposed an verification language QWIRE for quantum circuits. Besides, Hietala *et al.* presents SQIR [13] as an simplified but practical version of QWIRE. Methods of static analysis of quantum programs are also developed in quantum programming field. Ying [26] discusses the properties of quantum while-language.

3 ZX-Calculus

3.1 Zxcalculus.com's Tutorial

In the tutorial, some basic concepts and rewrite rules are provided. We briefly summary ideas in ZX calculus here.

The diagram is a flow from left to right. A wire represents a qubit address in the diagram, and a node represents an operator. There are two kinds of nodes in the diagram. The red one annotated with α represents an operator in the X basis with a phase α, and the green with α represents an operator in the Z

basis with a phase α. Each input wire indicates that this operator takes the corresponding qubit as an input, and each output wire indicates the operators output. The green node with angle α corresponds to an operator transforming the input $|0...0\rangle$ to output $|0..0\rangle$, and transforming the input $|1...1\rangle$ to output $e^{i\alpha}|1...1\rangle$. Formally, its semantics is $|0...0\rangle\langle 0...0| + e^{i\alpha}|1...1\rangle\langle 1...1|$. The red nodes are similar, but in $\{|+\rangle, |-\rangle\}$ basis. Hadamard gate is specially treated in the ZX calculus. It is an operator with only one input and one output, so it can be represented by a property of wire. If a wire has property of Hadamard, the qubit flowing through it changes basis. Hence, the diagram can be decomposed into a sequential flow from left to right, each node represents an operator, and are composed by operator composition.

An important property of ZX calculus is that the operator corresponding to the diagram is intact after a topological transformation of the diagram while keeping the input and output fixed. Thus the diagram can be directly represented by an abstract undirected graph with two sets of special nodes. There are other rewrite rules like the union of nodes in the same color, and changing node color and all wires connected to it at the same time. With these rewrite rules, the equivalence classes in the ZX-calculus extends. The universality of ZX-calculus implies that there are a set of rewrite rule leading to the hmomorphism from the equivalence classes of quantum operators to the equivalence classes of the ZX-calculus. This makes ZX-calculus usable in theory.

3.2 Applications of ZX-Calculus

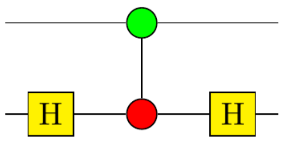
We applied ZX-calculus's diagramatic reasoning technique to two problems from the assignment and the quantum teleportation protocol.

First, let's warm up with a trivial example:

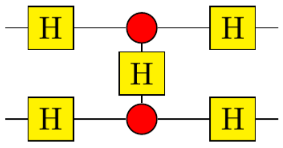
Hadamard and CNOT

A CNOT gate, with two Hadamard gates wrapping around the second wire, is equivalent to the entire circuit flipped. The proof is extremely simple with the axioms of ZX-calculus.

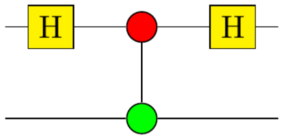
Proof.



$$= \quad \{ \text{ Color Change (Green} \to \text{Red) } \}$$



= { Color Change (Red → Green) }

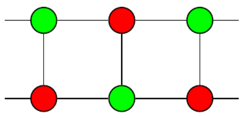


∎

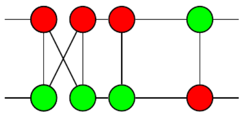
SWAP

This proof shows that two CNOT gates with one flipped CNOT in between is equivalent to the SWAP gate.

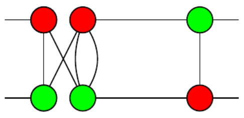
Proof.



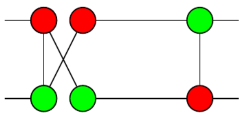
= { Bialgebra Rule }



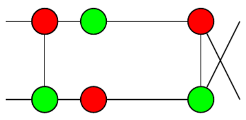
= { Spider Fusion * 2 }



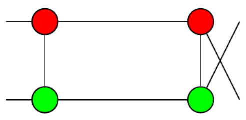
= { Spider Fighting }



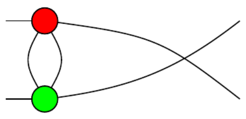
= { Only Connectivity Matters }



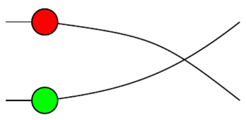
= { Identity Rule }



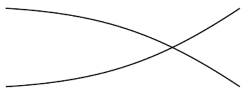
= { Spider Fusion * 2 }



= { Spider Fighting }



= { Identity Rule }



□

ZX-Calculus, just like all other string diagrams, admits a clean calculational proof style.

Quantum Teleportation

We found the correctness proof of the quantum teleportation protocol from [7]. We rephrased the proof to be more explicit about the theorems being applied.

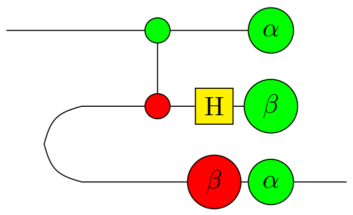
In ZX-calculus, the communication of classical bits and measurements are done at the meta level through variables. Diagrams with no output are referred to as effects. These effects allow us to represent the state after measurement. However, while we can prepare quantum states, we cannot predict post measurement effect. This is the reason why we need the phase variables.

In [8], there is an alternative way of doubling the ZX diagrams so classical communication can be encoded in the diagram itself, without resorting to meta variables. However, most of the papers we found do not use this type of representation.

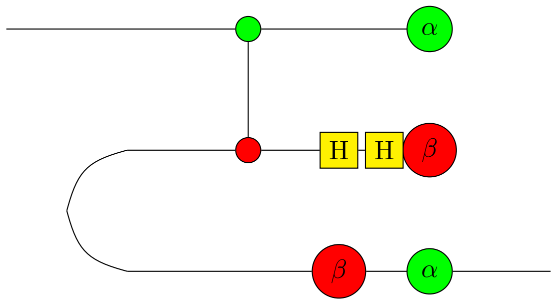
The diagram we start off with has two variables: $\alpha, \beta \in \{0, \pi\}$. They correspond to the four possible outcomes after Alice's measurement. The wire with two outputs and zero input corresponds to an EPR pair, which can be easily verified through the denotational semantics from diagrams to matrices. The top two wires are visible to Alice, while the bottom wire is visible to Bob.

To verify the correctness of the protocol, we need to show this entire diagram reduces to the identity wire.

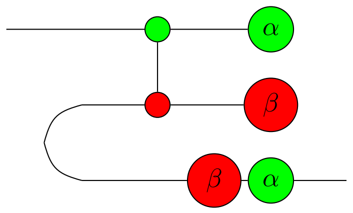
Proof.



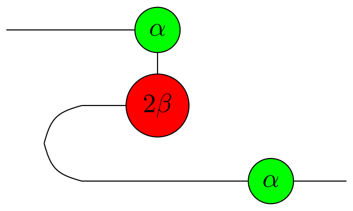
= { Color Change }



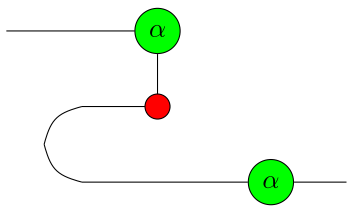
= { $HH = I$ }



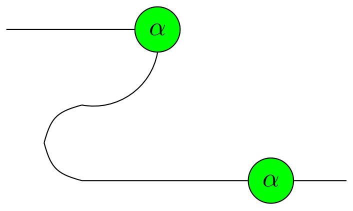
= { Spider Fusion * 3 }



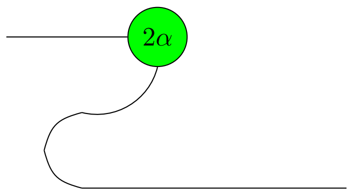
= { $\beta \in \{0, \pi\}$, thus $2\beta = 0$ }



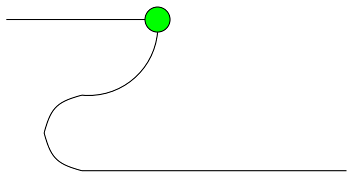
$=$ { Identity Rule }



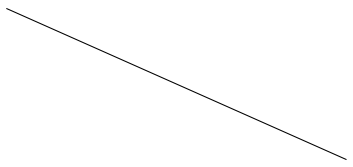
$=$ { Spider Fusion }



$=$ { $\alpha \in \{0, \pi\}$, thus $2\alpha = 0$ }



$=$ { Identity Rule }



□

3.3 PyZX: Large Scale Automated Diagrammatic Reasoning [18]

PyZX is a python library for quantum circuit optimization. PyZX, as its name suggests, is able to represent circuits as ZX-Calculus diagrams. Internally, PyZX differentiates between the circuits and diagrams. However, one can always easily convert back and forth between these two representations.

The library provides a basic user interface, which allows us to define circuits manually. It also supports importing files produced by circuit describing languages such as Quipper, QASM.

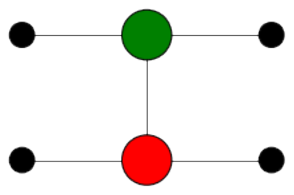
One interesting use case of PyZX is to verify the equivalence of two quantum circuits. Suppose we have two circuits, which correspond to the matrices U and V respectively. If we can show that $UV^\dagger$ reduces to the identity matrix, then we can conclude that $U = V$. Here is an example of how we can show the circuit identity from Problem 2.3, Assignment 2.

First, we define the two circuits:

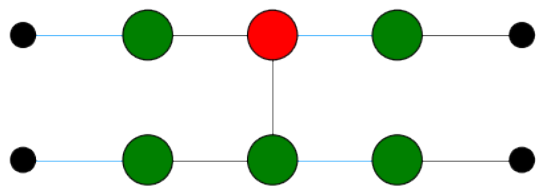
```
import pyzx as zx

cc0 = zx.Circuit(qubit_amount=2)
cc1 = zx.Circuit(qubit_amount=2)
cc0.add_gate("CNOT",0,1)

ccHH = zx.Circuit(qubit_amount=2)
ccHH.add_gate("HAD",0)
ccHH.add_gate("HAD",1)
cc1.add_circuit(ccHH)
cc1.add_gate("CNOT",1,0)
cc1.add_circuit(ccHH)
```

After executing the code, we have the left and right circuits stored in *cc*0 and *cc*1. We can use the *draw* command to show their representation as ZX diagrams:

```
zx.draw(cc0)
```



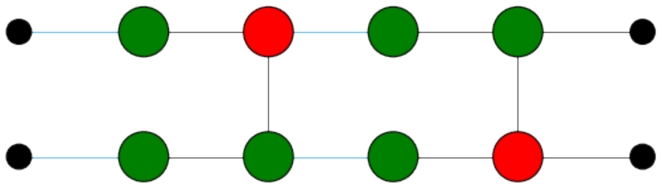
```
zx.draw(cc1)
```



Then we can wire one of the circuit with the adjoint of the other:

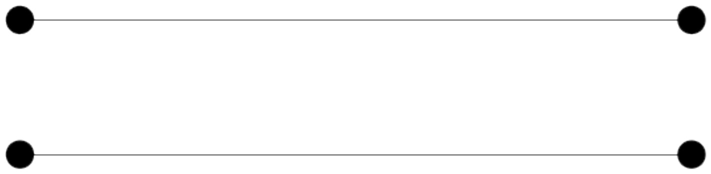
```
cc01 = zx.Circuit(qubit_amount=2)
```

```
cc01.add_circuit(cc1)
cc01.add_circuit(cc0.adjoint())
```



If we simplify the diagram induced by the circuit, we get back two parallel identity wires:

```
g01 = cc01.to_graph()
zx.simplify.full_reduce(g01)
zx.draw(g01)
```



This procedure shows that the two circuits are indeed equivalent.

Just like how a simplification algorithm for algebraic expressions is incapable of proving all algebraic equalities, the optimization algorithm is incomplete in the sense that it cannot verify all circuit equalities. However, based on the benchmark of the authors, the approach works surprisingly well in practice. They applied their equality test on a few optimized circuit and all of them except one were successfully proved as equivalent to the original circuit. The one exception which fails the equality test turns out to be an incorrect optimization.

The authors also included a list of challenges they encountered. One of them is circuit extraction, which involves converting from ZX-Calculus diagrams to quantum circuits. From their experience, the circuit extraction algorithm they use always seem to succeed, but it is unknown when exactly the heuristics would fail. Another open challenge is the application of ZX-Calculus to lattice surgery compilation. [14] shows that lattice surgery can be naturally described by ZX-Calculus, but the authors of this paper suggest there is still work that needs to be done before we can convert between quantum circuits and lattice surgery operations.

3.4 The completeness of ZX-Calculus

The construction of completeness of ZX-Calculus involves a series of works. Here completeness indicates that there exists a set of rules such that any ZX diagram can be transformed to another equivalent ZX diagram.

First, in [1], it is showed that ZX-calculus is complete for quantum circuits described by stablizers. The authors established the equivalence of graph states with ZX diagrams, proposed a normal form for these states. A reduction rule set is introduced as well. Then in [2], quantum circuits with single qubit cliford group and T gates are showed complete. At last, in [17], the authors showed a bijection from ZX-calculus diagrams to ZW-calculus diagrams, whose completeness was showed. By the completeness of ZW-calculus, the completeness of ZX-calculus is established finally.

4 Other developments

4.1 A mixed linear and non-linear logic: Proofs, terms and models[5]

Linear logic, unlike ordinary logic, has the notion of resources. This is relevant to quantum computing where a qubit cannot be copied due to the no-cloning theorem. Traditional logic does not capture the idea that resources can be consumed. For example, it is possible to form the conjunction $A \wedge A \wedge A \ldots \wedge A$ where A can occur an arbitrary number of times, as long as there A is available in the context of the judgement.

Prior research shows linear logic can be regarded as a superset of ordinary logic. However, despite its expressive power, linear logic can be verbose to use. Programmers should be able to access the more concise non-linear part of a programming language without having to introduce the extra syntax from the linear part. The major contribution of the paper is a new model which consists of a symmetrical monoidal closed category and a cartesian closed category which are related by adjoint functors. In Section 3, the author gives an example of how to construct a mixed linear and non-linear system from two independent systems.

This new approach makes a mixed linear and non-linear type system more user-friendly. It allows us to perform classical-postprocessing without having to worry about the heavy syntax induced by the linear quantum world.

4.2 QWIRE: A Core Language for Quantum Circuits [20]

Paykin *et al.* introduce QWIRE, a programming language used to implement quantum circuits that are controlled by and interact with classical computations. The main contribution of this work is that QWIRE provides correctness guarantees through its strong type system while being compatible with existing host languages. In addition, they define a denotational semantics for density matrices and prove that the operational semantics of QWIRE are sound with respect to the denotational semantics.

QWIRE has a host language that defines classical computations. The host language looks similar to existing functional, typesafe languages like ML and Haskell. In the host language, terms are introduced to run quantum circuits and output their results. This gives users flexibility since they are able to run different quantum circuits based on the output of previous quantum computations.

QWIRE has a circuit language used to define quantum computations. Wires in the circuit language have types that include unit, bits, qubits, and tuples. Quantum computations are defined as circuits, which apply gates (or unitary matrices), to wires. Furthermore, circuits are compositional. A circuit can be defined as a function that can be reused in the construction of other circuits.

An important requirement for quantum computation is that qubits cannot be used more than once. To enforce this, QWIRE uses a linear type system for the circuit language. Linear types ensure that when a wire is created, it must be used exactly once. This guarantees that one wire cannot be used as input to two different gates. This also ensures that every wire must be used as input to a gate, measured, or explicitly dropped. QWIRE uses Benton's Linear/Non-linear Logic (LNL) in its typing rules to enforce linearity [5]. At a high level, QWIRE has two contexts, one for classical, non-linear variables and one for quantum, linear variables. When linear variables are consumed, they are removed from the linear context so they cannot be used again.

Here is a simple example quantum program written in QWIRE:

```
flip : Bool = run (q  <- gate init0 ();
                   q' <- gate H q;
                   b  <- gate meas q';
                   output b)
```

This flip operation calls `run` which takes a quantum circuit as input and returns the result of the circuit. The circuit begins by initializing `q` with the $|0\rangle$ state. It then applies the hadamard gate, `H` and measures the result. This results in a boolean value with 50% odds of being `True` or `False`.

An open problem for QWIRE is that it does not currently support error correcting codes. We also expect incorporating QWIRE into existing programming languages might prove difficult due to its dependence on linear type systems (which most languages do not provide).

4.3 Verified Optimization in a Quantum Intermediate Representation [13]

The QWIRE language constructs a dynamic framework for quantum computing, but is too complicated to describe and show properties of simple circuits. In [13], Hietala *et al.* present SQIR, an intermidiate representation for quantum programs.

It represents the qubits as an fixed one dimensional array, making use of the natural linearity of quantum circuits. When embedded in Coq, SQIR has great expressibility to describe circuits. Besides, it implemented the denotational semantics to matrix level. So the proofs of correctness is relatively easy compared to QWIRE.

Hietala *et al.* also proved the correctness of several small scale circuits used in optimizations. Moreover, they formally showed the correctness of Deutsch-Jozsa algorithm, one of the quantum algorithms with quantum speed up.

Due to the ease in design, SQIR is useful in proving basic algorithms' correctness. In their later work they proved several common circuit optimization tricks based on SQIR.

4.4 A Verified Optimizer for Quantum Circuits [12]

Hietala *et al.* present VOQC, a verified compiler for quantum circuits. Compilers optimize quantum programs to reduce the number of gates and compile them to support different hardware architectures. Unfortunately, such program transformations can include mistakes which change the semantics of programs and introduce bugs. VOQC addresses this problem by leveraging formal methods to mechanically prove that the optimizations it performs are correct.

VOQC compiles programs written in the quantum programming lanugage SQIR. SQIR is embedded in the Coq proof assistant, which enables VOQC to provide correctness guarantees about the optimizations it performs. To reason about quantum program correctness, VOQC offers two denotational semantics. One is a unitary representation of quantum programs, while the other is a density matrix representation. To prove that an optimization is correct, VOQC proves that the denotational semantics of the optimized program are equivalent to the denotational semantics of the original program. To simplify these proofs, VOQC implements a Coq tactic called `gridify` that makes use of various identities for matrix arithmetic.

VOQC is useful in practice. The optimizations it implements are extracted to an OCaml application, which makes it simple for others to run. It can take standard OpenQASM circuits as input, convert them to SQIR, perform optimizations, and output them back as OpenQASM. In addition, it can compile programs to the Tenerife machine architecture, which is required for IBM's quantum hardware. The optimizations VOQC are effective and offer a 17.7% reduction in the number of gates on average. This work was recently submitted for publication and is the forefront of verified quantum compilers. In future work, the authors will use VOQC as part of a fully verified quantum compiler stack.

4.5 Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory [21]

Péchoux *et al.*extend the Quantum Programming Language proposed by Selinger *et al.*[24] (but also named QPL). They introduce inductive datatypes into the system, where these types are used to represent qubits, classical bits, and the data structure based on them. Unlike most of other models, QPL supports the qubit defined in infinite dimensional Hilbert space. This is a covenience for programmers to measure and discard qubits.

The syntax of QPL includes quantities of terms. First, there are **new unit, new qbit, copy** to generate new objects. Second, there are **discard, measure** to decease a qubit. Third, it includes the generating of basic quantum states using its **left, right** syntax. Fourth, it includes basic construction of data structures based on both qubits and classical bits, using pairing clauses. Fifth, it contains typical quantum operators described in the while language form. At last, it includes a first-order recursion in the syntax.

After constructing the syntax and their corresponding semantics, the authors show that the QPL's type system is affine. Thus, they construct a slice category $\mathbf{C}/I$ to interpretate their type system. As a result, a category is induced to represent the QPL language, where the authors name it $\mathbf{W}^*_{\mathrm{NCPSU}}$, and a casual interpretation of types can be described by functors on the slice category. This description is sound and adequate proved in the paper.

4.6 ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity [3]

ZX-calculus is not the only graphical calculus for representing quantum computations. Instead of using Z and X spiders as primitives, one can choose a different set of primitives. ZH-calculus is a graphical calculus which uses Z-spiders and H-boxes as its primitives. H-boxes, just like ordinary spiders, can have an arbitrary amount of finite input and output. In the case where there is exactly one input and one output, an H-box with phase 0 corresponds to a Hadamard gate.

The paper introduces the !-box notation, which can be used as a shorthand for a family of diagram equations. This notation allows the authors to represent the derivations and theorems concisely. Compared to ZX-calculus, ZH-calculus admits a much simpler completeness proof. The completeness proof of ZX-calculus is done indirectly using the completeness ZW-calculus[16]. On the contrary, ZH-calculus's completeness proof is done completely within itself by showing every ZH diagram can be reduced to a normal form.

ZH-calculus also has more succinct representations of some of the quantum gates. The authors noted that the AND gate can be represented with only 2 generators in ZH-calculus, while in ZX-calculus around 25 generators are required.

5 Conclusion and future directions

In this report, we survey the main results about ZX-calculus and other related works in the verification of quantum programs. The ZX-Calculus is useful in verification because its wonderful property related to abstract graph, and is universal and complete. Extensively we introduce several results revealing the relationship of quantum programs with the categorical models.

Several future directions are interesting for research. The current rule set to show completeness is very complicated. A simpler version of a rule set is necessary for the practical usages of ZX-calculus. As an approach to formal verification, its implementation in a proof engine is essential. Another issue of the ZX-Calculus is that measurement is not modeled in it, while measurement should be an pivotal component in quantum algorithms.

References

[1] Miriam Backens. “The ZX-calculus is complete for stabilizer quantum mechanics”. In: *New Journal of Physics* 16.9 (2014), p. 093021.

[2] Miriam Backens. “The ZX-calculus is complete for the single-qubit Clifford+ T group”. In: *arXiv preprint arXiv:1412.8553* (2014).

[3] Miriam Backens and Aleks Kissinger. “ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity”. In: *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 31, 2019), pp. 23–42. ISSN: 2075-2180. DOI: `10.4204/EPTCS.287.2`. arXiv: `1805.02175`. URL: `http://arxiv.org/abs/1805.02175` (visited on 11/22/2019).

[4] Niel de Beaudrap and Dominic Horsman. “The ZX calculus is a language for surface code lattice surgery”. In: *arXiv:1704.08670 [quant-ph]* (Apr. 27, 2017). arXiv: `1704.08670`. URL: `http://arxiv.org/abs/1704.08670` (visited on 09/12/2019).

[5] P. N. Benton. “A mixed linear and non-linear logic: Proofs, terms and models”. In: *Computer Science Logic*. Ed. by Leszek Pacholski and Jerzy Tiuryn. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, pp. 121–135. ISBN: 978-3-540-49404-1.

[6] Bob Coecke. “Quantum picturalism”. In: *Contemporary Physics* 51.1 (Jan. 1, 2010), pp. 59–83. ISSN: 0010-7514. DOI: `10.1080/00107510903257624`. URL: `https://doi.org/10.1080/00107510903257624` (visited on 07/12/2019).

[7] Bob Coecke and Ross Duncan. “Interacting quantum observables: categorical algebra and diagrammatics”. In: *New Journal of Physics* 13.4 (Apr. 2011), p. 043016. ISSN: 1367-2630. DOI: `10.1088/1367-2630/13/4/043016`. URL: `https://doi.org/10.1088%2F1367-2630%2F13%2F4%2F043016` (visited on 07/14/2019).

[8] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes*. Google-Books-ID: I9gcDgAAQBAJ. Cambridge University Press, Mar. 16, 2017. 847 pp. ISBN: 978-1-107-10422-8.

[9] David Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.

[10] Andrew Fagan and Ross Duncan. "Optimising Clifford Circuits with Quantomatic". In: *Electronic Proceedings in Theoretical Computer Science* 287 (Jan. 31, 2019), pp. 85–105. ISSN: 2075-2180. DOI: `10.4204/EPTCS.287.5`. arXiv: `1901.10114`. URL: `http://arxiv.org/abs/1901.10114` (visited on 09/16/2019).

[11] Alexander S Green et al. "Quipper: a scalable quantum programming language". In: *ACM SIGPLAN Notices*. Vol. 48. 6. ACM. 2013, pp. 333–342.

[12] Kesha Hietala et al. *A Verified Optimizer for Quantum Circuits*. 2019. arXiv: `1912.02250 [cs.PL]`.

[13] Kesha Hietala et al. "Verified Optimization in a Quantum Intermediate Representation". In: *arXiv preprint arXiv:1904.06319* (2019).

[14] Clare Horsman et al. "Surface code quantum computing by lattice surgery". In: *New Journal of Physics* 14.12 (Dec. 7, 2012), p. 123011. ISSN: 1367-2630. DOI: `10.1088/1367-2630/14/12/123011`. arXiv: `1111.4022`. URL: `http://arxiv.org/abs/1111.4022` (visited on 09/16/2019).

[15] Ali Javadi-Abhari. "Towards a Scalable Software Stack for Resource Estimation and Optimization in General-Purpose Quantum Computers". PhD thesis. Princeton University, 2017.

[16] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. "Completeness of the ZX-Calculus". In: *arXiv:1903.06035 [quant-ph]* (Mar. 13, 2019). arXiv: `1903.06035`. URL: `http://arxiv.org/abs/1903.06035` (visited on 07/17/2019).

[17] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. "The ZX-calculus is complete for the single-qubit Clifford+ T group". In: *arXiv preprint arXiv:1903.06035* (2019).

[18] Aleks Kissinger and John van de Wetering. *PyZX: Large Scale Automated Diagrammatic Reasoning*. 2019. arXiv: `1904.04735 [quant-ph]`.

[19] Daniel Marsden. "Category Theory Using String Diagrams". In: (Jan. 28, 2014). URL: `https://arxiv.org/abs/1401.7220v2` (visited on 04/20/2019).

[20] Jennifer Paykin, Robert Rand, and Steve Zdancewic. "QWIRE: A Core Language for Quantum Circuits". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. event-place: Paris, France. New York, NY, USA: ACM, 2017, pp. 846–858. ISBN: 978-1-4503-4660-3. DOI: `10.1145/3009837.3009894`. URL: `http://doi.acm.org/10.1145/3009837.3009894` (visited on 07/12/2019).

[21] Romain Péchoux et al. "Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory". In: (2019).

[22] Robert Rand, Kesha Hietala, and Michael Hicks. “Formal Verification vs. Quantum Uncertainty”. In: (2019). In collab. with Michael Wagner, 11 pages. DOI: `10.4230/lipics.snapl.2019.12`. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10555/` (visited on 08/02/2019).

[23] P. Selinger. “A Survey of Graphical Languages for Monoidal Categories”. In: *New Structures for Physics*. Ed. by Bob Coecke. Vol. 813. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 289–355. ISBN: 978-3-642-12820-2 978-3-642-12821-9. DOI: `10.1007/978-3-642-12821-9_4`. URL: `http://link.springer.com/10.1007/978-3-642-12821-9_4` (visited on 07/04/2019).

[24] Peter Selinger. “Towards a quantum programming language”. In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586.

[25] Peter Selinger, Benoıt Valiron, et al. “Quantum lambda calculus”. In: *Semantic techniques in quantum computation* (2009), pp. 135–172.

[26] Mingsheng Ying. *Foundations of Quantum Programming*. Morgan Kaufmann, 2016.

A Timeline

- **9/28/19** - A mixed linear and non-linear logic: Proofs, terms and models[5]
- **10/5/19** - QWIRE: A Core Language for Quantum Circuits [20]
- **10/12/19** - PyZX: Large Scale Automated Diagrammatic Reasoning [18]
- **10/17/19** - Midterm report due. Summary of previous readings.
- **10/19/19** - Formal Verification vs. Quantum Uncertainty[22]
- **10/26/19** - Quantum Picturalism [6]
- **11/2/19** - A Survey of Graphical Languages for Monoidal Categories - Chapter 2 and 3 [23]
- **11/9/19** - Category Theory Using String Diagrams - Section 1, 2, and 3 [19]
- **11/14/19** - Project slides due.
- **11/16/19** - Completeness of the ZX-Calculus - Chapter 1 and 2 [16]
- **11/21/19** - Group presentations.
- **11/23/19** - Surface code quantum computing by lattice surgery [14]
- **11/30/19** - Optimising Clifford Circuits with Quantomatic[10]
- **12/12/19** - Final report due. Summary of readings.