RWTH AACHEN UNIVERSITY
Chair of Computer Science 2
Software Modeling and Verification

Master Thesis

Compilation of Quantum Programs with Control Flow Primitives in Superposition

Sascha Thiemann
Matr.-No.: 406187
Study Program: Computer Science M.Sc.
August 18, 2024

Supervisors: apl. Prof. Dr. Thomas Noll
Chair for Software Modeling and Verification
RWTH Aachen University

Prof. Dr. rer. nat. Dominique Unruh
Chair for Quantum Information Systems
RWTH Aachen University

Contents

1 Introduction

2 Background

... Background

2.1 Quantum Computing

While computers are prevalent and important in today's society, there are many relevant problems which classical computers can currently and perhaps will never realistically be able to solve. Quantum Computing is gaining more momentum as the technology that could solve at least some of these problems. For example, Quantum algorithms like Shor's algorithm [Shor97] could provide a significant improvement for prime factorization given sufficient technology. Therefore, it is estimated to be a valuable market with many of the largest technology companies as well as governments investing billions in the research and development of quantum technology [RDB*22]. While there already exist detailed theoretical foundations [van20, Ying11, YYF12] and advanced algorithms for QC [ACR*10, BGB*18, LoCh19, Shor97], the technology of quantum computers is said to be on the level of classical computers in the 1950s [CFM17]. In the following section, we take a look at the basic concepts of a quantum computer and the core principles it relies on.

Classical Computers are based on simple operations, like `and`, `or`, and `not`, on bits. These bits can either have a value of 0 or 1. Similarly, at their core, quantum computers apply simple operations, like `controlled not`, and `Hadamard`, on quantum bits (qubits). In contrast to classical bits, quantum computers use the unique properties of quantum mechanics to enable qubits to have not just one value of either 0 or 1 but a combination of both. The phenomenon, where a particle or qubit exists in multiple states at the same time, is called *superposition*. Additionally, quantum computers also use the idea of *entanglement* to their advantage where the value of a qubit is dependent on another qubit. The combination of superposition and entanglement enable quantum computers to solve specific problems more efficiently than classical computers [RDB*22], e.g. prime factorization [Shor97].

This is just a colloquially explaination and not technically correct

Repeating info from paragraph above?

Models for Quantum Computers can be divided into three main categories, the *analog model*, the *measurement-based model*, and the *gate-based model*. The analog model uses smooth operations to evolve a quantum system over time such that the resulting system encodes the desired result with high probability. It is not clear whether this model allows for universal quantum computation or quantum speedup [DiCh20]. Instead of smoothly evolving a system, the measurement-based model starts with a fixed quantum state, the cluster-state. The computation is accomplished by measuring qubits of the system, possibly depending on the results of previous measurements. The concept of gate teleportation is used such that the measurements realize quantum gates. The result is a bit-string of the measurement results [DiCh20, Niel06]. Lastly, the gate-based model uses a digitized, discrete set of qubits that are manipulated by a sequence of operations represented by quantum gates. The result is obtained by measuring the qubits at the end of the computation. Although digital quantum computation is more sensitive to noise than analog computations, the digitization can also be used for quantum error correction [DMN13] and mitigate the increased noise [DiCh20]. Furthermore, because qubits are actively manipulated and not passively evolved, digital quantum computers are more flexible than analog ones [RDB*22]. Therefore, the gate-based model is the most common model and this thesis will mainly focus on it.

explain?

Add section on: no cloning/deleting, implicit measurement theorems?[WoZu82, KuBr00]

2.1.1 Superposition

Is a citation needed for this definition? (if yes use [DiCh20])

The first important property of quantum mechanics used by quantum computers is the idea of superposition. The concept of superposition is most known for its role in the "Schrödinger's cat" thought experiment [Wine13] where the life of a cat in a box is dependent on a particle in superposition, only when "measuring" the state of the cat, i.e. looking into the box, we can know if it is still alive.

Similar to the cat being referred to as alive and dead at the same time, qubits in superposition are often informally described as simultaneously having a value of 0 and 1 until their state is measured. However, a qubit in superposition is more formally a linear combination of its basis states. The basis states are the states where the qubit has a value of 0, written $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$, and 1, written $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. Furthermore, the state can be reduced to a simple vector. Therefore, a state ψ in superposition can be written as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\end{pmatrix}.$$

The factors α and β are the amplitudes of the basis states and are complex numbers. The factors must also satisfy the condition $|\alpha|^2 + |\beta|^2 = 1$. This is because of the relation of the amplitudes to the probability to which basis state the state will collapse when measure, described in Sec. 2.1.3 about measurement.

Beside $|0\rangle$ and $|1\rangle$, there exist more relevant short hands for quantum state. For example, $|+\rangle$ and $|-\rangle$ are states in uniform superposition, i.e. both basis state are equally likely, and often used when discussing quantum state und transformations. They are defined as follows:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

2.1.2 Entanglement

Another important quantum mechanical concept is entanglement. Simply said, two qubits are entangled when their values depend on each other. An example would be a quantum system where two qubits are in superposition and equally likely to collapse to either 0 or 1; whichever value one qubit collapses to when measured, the second one will also collapse to the same values. Additionally, changes to one of the qubits can also affect the other one. This happens independent of the locations of the two qubits [RDB*22, HHHH09].

Define Bell β_{00} state

A more formal definition for an entangled state uses the definition of a composite system. Two separate quantum system can be represented as a single system with the tensor product of both systems. For example, the combined state $|\psi\rangle$ of the separate

states $|0\rangle$ and $|1\rangle$ can be represented as:

$$|\psi\rangle = |0\rangle \otimes |1\rangle = |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

When a quantum state cannot be expressed as a tensor product of two states, the state is entangled. The previous example is a case of a maximally entanglement Bell state [DiCh20, MHH19], often denoted β_{00}, and can be expressed as the following:

$$\beta_{00} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The entanglement of states is used by leveraging the effect of the qubits on each other to collaborate to calculate the result. Although this can be simulated on classical computers, it cannot be achieved "natively" because all classical bits are independent of each other. Moreover, quantum algorithms not using entangled states can often be simulated efficiently on classical computers [MHH19]. Therefore, entanglement is at the core of quantum computing but it can also have unintended consequences one needs to be aware of when designing quantum algorithms.

To calculate specific functions or intermediate values, quantum algorithms may need to use additional qubits or registers which state can, in turn, be entangled with the main data of the algorithm. If this entanglement is not resolved in time by, e.g., uncomputing the changes to the qubit or register, it can interfere with future calculations or measurements and cause the results to be invalid. This effect is called *disruptive entanglement* [YVC24].

register were not previously mentioned, add small reference

Uncomputing as a concept was not introduced before

Cannot find literature besides [YVC24] which calls this effect disruptive entanglement, use anyway?

2.1.3 Measurement

For quantum computer to be of any use, we need a way to read out information about its state. However, the information we can obtain from a quantum system is limited by the quantum measurement postulate. The postulate states that the only way, to gain any information from a quantum system, is to measure it. When measuring a quantum state, the state irreversibly collapses to one of its basis states. Furthermore, this is a probabilistic transformation and the original state in superposition cannot be recovered from the result. For a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the measurement collapses the state to $|0\rangle$ with a probability of $|\alpha|^2$. Correspondingly, the state will collapse to $|1\rangle$ with a probability of $|\beta|^2$ when measured [DiCh20].

Measurement can be represented a a measurement basis set $\{M_i\}_i$ which requires the following condition:

$$\sum_i M_i^\dagger M_i = I.$$

The probability that outcome i is obtained when measuring a state $|\psi\rangle$ is equivalent to $|M_i |\psi\rangle|^2$. After the measurement of outcome i, the state $|\psi'\rangle$ will be equivalent to

$$|\psi'\rangle = \frac{M_i |\psi\rangle}{|M_i |\psi\rangle|} = \frac{M_i |\psi\rangle}{\sqrt{\Pr[\text{observe } i]}}.$$

In contrast to all other transformations, measurements are neither unitary nor reversible and, therefore, are able to "destroy" information on the quantum state before the measurement [DiCh20].

already mentioned above, only mention once

2.1.4 Quantum Gates

In gate-based quantum computer, the transformations applied to the quantum data are represented by *quantum gates*. Similar to quantum states, which can be represented by linear combinations of basis states, or vectors, quantum gates can be formulated as linear transformations of these combinations, or a matrix. Because the result of such a transformation also needs to be a valid quantum state, the transformation needs to be norm-preserving, or *unitary* [DiCh20]. The most relevant and often used unitary gates are depicted in Tab. 2.1.

Add depications for gates in circuits?

A matrix U is unitary if it has an inverse matrix which is equal to its conjugate transpose $U^\dagger$, i.e. the following must hold:

Add troffoli gate, large but important for universality?

$$UU^\dagger = I.$$

Therefore, all transformations applied to quantum states in a gate-based quantum computer must be reversible by definition. This limitation does not apply to classical computers where non-reversible transformations, e.g. mapping an arbitrary bit to a specific value, are easily implementable.

To design a useful quantum computer or language, the set of gates should be *universal*. A set of gates is universal if any gate can be simulated by a combination of the gates from the set with arbitrary accuracy [BrBr01]. An example for a universal set of gates is the combination of the Traffoli gate together with the Hadamard gate [DiCh20].

very short section, expand on universality?

2.1.5 Relevant Algorithms

Although quantum computers have impressive technical abilities, they cannot function without a specially designed algorithm. This algorithm needs to exploit the special quantum properties of qubits to achieve *quantum advantage*, i.e. a better complexity than any classical algorithm. One of the first algorithms to show its quantum advantage was the Deutsch–Josza algorithm [DeJo92]. Deutsch et al. define a problem that can be solved in exponential time on classical computer and present a quantum algorithm which can solve the problem in polynomial time. The Bernstein-Vazirani algorithm [BeVa93] is another example with shown quantum advantage, resulting in a polynomial speed up. However, currently, there does not exist a use case for either of the algorithms and, therefore, they are only of limited theoretical interest [DiCh20].

write better introduction

	Gates	Matrix	Ket-notation
Pauli gates	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\lvert 0\rangle \mapsto \lvert 1\rangle$ $\lvert 1\rangle \mapsto \lvert 0\rangle$
	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\lvert 0\rangle \mapsto \lvert i\rangle$ $\lvert 1\rangle \mapsto -\lvert i\rangle$
	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\lvert 0\rangle \mapsto \lvert 0\rangle$ $\lvert 1\rangle \mapsto -\lvert 1\rangle$
Hadamard gate	H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\lvert 0\rangle \mapsto \lvert +\rangle$ $\lvert 1\rangle \mapsto \lvert -\rangle$
Phase gate	$P(\lambda)$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$	$\lvert 0\rangle \mapsto \lvert 0\rangle$ $\lvert 1\rangle \mapsto e^{i\lambda} \cdot \lvert 1\rangle$
Controlled-NOT gate	CX	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\lvert 00\rangle \mapsto \lvert 00\rangle$ $\lvert 01\rangle \mapsto \lvert 01\rangle$ $\lvert 10\rangle \mapsto \lvert 11\rangle$ $\lvert 11\rangle \mapsto \lvert 10\rangle$
Traffoli gate	CCX	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$	$\lvert 000\rangle \mapsto \lvert 000\rangle$ $\lvert 001\rangle \mapsto \lvert 001\rangle$ $\lvert 010\rangle \mapsto \lvert 010\rangle$ $\lvert 011\rangle \mapsto \lvert 011\rangle$ $\lvert 100\rangle \mapsto \lvert 100\rangle$ $\lvert 101\rangle \mapsto \lvert 101\rangle$ $\lvert 110\rangle \mapsto \lvert 111\rangle$ $\lvert 111\rangle \mapsto \lvert 110\rangle$

Table 2.1: List of relevant quantum gates in matrix representation as as functions in ket-notation.

An algorithm with more potential for practical use is Shor's algorithm [Shor97]. It presents an efficient quantum implementation for the discrete logarithm, i.e. find r for a given a, x, p such that $a^r = x \mod p$. The algorithm is of special interest because Shor also provides a reduction of prime factorization to order finding; order finding is a special case of the discrete logarithm where $x = 1$. Modern cryptography is often based on the complexity of factoring large prime numbers , e.g. the commonly used RSA cryptosystem [RSA78]. Therefore, an advanced quantum computer could brake these systems with Shor's algorithm [MVZJ18]. Not only does this prospect provide a practical use-case for Quantum Computer but it also creates the research field of *post-quantum cryptography* [BeLa17].

poly time

discrete log is also used in modern cryptography

Another relevant algorithm or transformation is the quantum Fourier transform (QFT) [Copp02]. Beside being used as a subroutine in Shor's algorithm, it is also relevant for other algorithm, e.g. addition of quantum registers [Drap00]. Similar to the discrete Fourier transform [Wino78] which operates on vectors, the QFT_{2^n} operates on the quantum equivalent of vector, quantum registers, of size n. Registers of size n consist of n qubits. From the register, the QFT extracts periodic features which are then used by the algorithms using the QFT.

bad formulation

2.1.6 Circuit optimization

- Subsection about possible circuit optimizations
- What are different techniques
- How are they related to the specific quantum hardware
- Peephole opt. of circuits, e.g. null gates ...

2.2 Quantum Control Flow

The idea of quantum control flow was first used by Altenkirch et al. [AlGr05] when defining a functional programming language with quantum control flow elements. The language uses an if-statement in superposition, `if`$^\circ$, which is used to, e.g., defined the Hadamard gate as a function *had* instead of a matrix. The *had* function takes a qubit as an input. If the qubit is true, i.e. the value is one, the function returns a uniform superposition of true and false, where true has a negative sign. Correspondingly, for a false input, a uniform superposition with both signs positive is returned.

$$
\begin{aligned}
had &: Q \to Q \\
had &: x \mapsto \texttt{if}^\circ x \\
&\qquad \texttt{then } \{false \mid -true\} \\
&\qquad \texttt{else } \{false \mid true\}
\end{aligned}
$$

Quantum control flow can be divided into *quantum branching* and *iteration* [YVC24]. In the following, we will discuss both branching and iteration in superposition as well as the limitations of quantum control flow.

2.2.1 Branching

Based on the work presented by Altenkirch et al. [AlGr05], the concept of quantum control flow, more specifically quantum branching, was expanded on and formally defined by Ying et al. [YYF12]. They introduce two different types of quantum branching, quantum guarded commands, and quantum choices as a special case of guarded commands. The definition of quantum guarded commands is based on Dijkstra's guarded commands [Dijk75]. Guarded commands concern the nondeterministic executing of functions based on Boolean expressions, where the nondeterminism derives from the possible overlapping of the guards. In contrast, quantum branching allows for execution of functions based on a value in superposition. The functions are executed such that the result may be a superposition of the results of the individual functions [YVC24]. Quantum branching is, e.g., used in simulation algorithms like [BGB*18], and [LoCh19].

The formal definition for classical guarded commands is given by:

$$\square_{i=1}^{n} b_i \rightarrow C_i$$

where C_i is a command guarded by a Boolean expression b_i. The command can only be executed if the expression is true. Similarly, quantum guarded commands map to a set of quantum programs P_i. Further, a set of qubits or quantum registers, which are not used in any guarded program P_i, and a corresponding orthogonal basis $|i\rangle$ is given. The resulting quantum guarded command is of the following form:

can give better/more indepth explantation, example?

$$\square_{i=1}^{n} \bar{q}, |i\rangle \rightarrow P_i.$$

The quantum programs are guarded by the basis states and the control flow results from the superposition of these basis states [YYF12].

2.2.2 Iteration

Quantum iteration can be implemented either as quantum recursion or quantum loops. While some languages implement loops based on the measurement of qubits or registers [Ying11], the concept of quantum iteration requires the body of the loop to be executed in superposition based on the guard in superposition [YYF12].

reference [FYY13]?

While classical iteration takes an operation and repeats it on a classical register for k iterations, quantum iteration is dependent on a value k' in superposition and, correspondingly, return a quantum register in superposition. Moreover, it is a special case of quantum branching and heavily restricted by the limitations of quantum computers [YVC24].

2.2.3 Limitations

While quantum control flow is often based on the corresponding control flow primitives on classical computers, it is restricted by multiple limitations imposed by quantum computers. Therefore, many control flow primitives that are used in classical programs

can either by not used at all or in a limited capacity. There are two main limitations for quantum programs. Firstly, all gate-based quantum computers need to adhere to *reversibility*. Secondly, programs need to follow the *synchronization* principle for them to return any useful results [YVC24].

Reversibility

As introduced in Sec. 2.1.4, any sequence of instructions on gate-based quantum computers, excluding measurements, is required to be reversible by definition, as they are all unitary transformations. Therefore, any quantum control flow is also required to adhere to this principle. A resulting limitation, that is not present on classical computer, is that any guards for guarded commands need to be immutable in the commands themselves. For example, if a qubit's state is flipped when its value is 0, the resulting command will always return value of 1. When a program returns the same result regardless of which statements where executed, the program cannot be reversible. Moreover, control flow, as implemented in classical computers, is also not possible. At a basic level, modern computers use jump and conditional jump instruction to implement branching and loop. However, any classical jump instruction is inherently irreversible. Not only can a jump go to a section of code that is accessible without any jumps, multiple jumps can also lead to the same line of code. Therefore, the a reversed program cannot know which path would be or was taken in original program [YVC24].

Also inherint in definition of quantum branching [YYF12]

A simple solution seems to be offered by the *Landauer Embedding* [Land61]. Fundamentally, the idea of the embedding is to turn a now reversible function into a reversible one by not only returning the output but also the input of the function. In the case a quantum program with, e.g., jump instructions, this would result in an output of the result and a complete history of which path was taken through the program. However, because the quantum data depends on the program history, they become entangled. This leads to disruptive entanglement, as described in Sec. 2.1.2, causing invalid results [YVC24].

Synchronization

As we have previously seen, reversibility alone is not the only limiting factor on quantum control flow. When handling control flow, similar to the classical implementation, with a program counter in superposition, the program counter can become entangled with the data and result in disruptive entanglement leading to an invalid result. To avoid this issue, the program must not only be reversible but also adhere the the principle of *synchronization*. It states that control flow must become independent from the data. Further, because any quantum program needs to be synchronized to return any useful results, while loops dependent on a value in superposition need to be bound by a classical value [YVC24].

2.3 Quantum Control Machine

- Definition of Quantum Control Machine
- How does it solve/handle the limitations of quantum control flow?
- Example program
  - Example for non-reversible program
  - Example for reversible but not synchronous program
  - Example for correct program

```
1      add   res $1
2      add   r1  y
3  l1: rjne  l3  r1  y
4  l2: jz    l4  r1
5      mul   res x
6      radd  r1  $1
7  l3: jmp   l1
8  l4: rjmp  l2
```

Figure 2.1: QCM exponentiation without synchronization

```
1      add   res $1
2      add   r1  max
3  l1: rjne  l3  r1  max
4  l2: jz    l4  r1
5  l5: jg    l7  r1  y
6      mul   res x
7  l6: jmp   l8
8  l7: rjmp  l5
9      nop             ; padding
10 l8: rjle  l6  r1  y
11     radd  r1  $1
12 l3: jmp   l1
13 l4: rjmp  l2
```

Figure 2.2: Synchronized QCM exponentiation

2.4 Quantum Languages

2.4.1 QASM Language

- Give overview of QASM language and concepts

2.5 Compilation

2.5.1 Lexer

2.5.2 Parser

2.5.3 Semantic Analysis

2.5.4 Code Generation

2.5.5 Optimization

- Different optimization techniques
  - Constant folding or constant propagation
  - Peephole optimization

2.5.6 ANTLR

- Give overview of ANTLR and parsing in general

3 Concept

- Expand on concept section from proposal:

The concept for the master thesis is to take the idea of the QCM, specifically the core concept of quantum control flow, and reduce it to its most basic elements and make it realistic for and applicable to NISQ era quantum computers. Concretely, we want to go from jump instructions to basic if-else clause to reduce the complexity of the code and make it easier to read and write. These if-else clauses can easily be implemented as the application of controlled gates. Moreover, because of the synchronization principle and the fact that current quantum computer technology does not support loops depending on measurement, any other loop can be reduced to a for-loop that is unrolled at compile time.

To achieve this goal, we want to define a language "Luie" (short for loop-unrolled if-else) which is partially based on the quantum while language used by Ying [Ying11]. The language is extended by a quantum if clause which takes a quantum register and executes the statements in the clause based on the value of the register. Furthermore, the clause could even be extended to include the evaluation of boolean expression. While the language cannot include while statements based on measurements of registers, as it is the case in the language proposed by Ying, it can include bounded loops which are unrolled at compile time. The language will then be compiled to QASM.

4 Implementation

4.1 Semantic analysis

- What is semantic analysis used for?
- How is it implemented in Luie?
- Different types of semantic analysis
- Errors
  - Types of errors: Critical, warning
  - Different critical errors (Type, undefined, ...)
  - Different warnings (invalid range, ...)

4.2 Code Generation

- How is code generated?
- Important classes and abstractions

4.3 Language Overview

4.3.1 Blocks and Scopes

- Basic structure of Luie
- Consists of blocks and statements
- One main block
- Symbol table that handles scopes
- All blocks have scope

Grammar

- A block consists of arbitrarily many definitions and statements

```
1    grammar Luie;
2
3    parse
4     : block EOF
5     ;
6
7    block
```

```
8       : (definition | statement)*
9       ;
```

4.3.2 Data Types

- Different data types
  - Register
  - Qubits (Registers with size 1)
  - Iterators, in more detail in Sec. 4.3.4

Grammar

- Registers and qubits defined in declaration
- Optional size, otherwise qubit (size 1).
- Identifier is the name of the register/qubit
- Arbitrary string starting with a character or underscore
- Later

```
1     declaration
2       : 'qubit' ('[' size=expression ']')?
            IDENTIFIER ';'
3       ;
4
5     IDENTIFIER
6       : [a-zA-Z_] [a-zA-Z_0-9]*
7       ;
```

Semantic analysis

- Semantic analysis can be differentiated between checking the use of an identifier and checking the type of an identifier.
- *Definedness*: symbol table to check if an identifier is defined
- *Type checking*: symbol table to check if the type of an identifier is correct
  - What are different type
  - Object oriented approach, i.e. because qubit inherits from register it is also a register
  - -> less checks required

Code Generation

- `Definition`s used for code generation, symbols can be transformed into definitions
- Needs unique identifier (language has scopes -> multiple variables with same name possible, while QASM does not)
- Unique identifiers given at generation time because of for loops
  - Explain why ...

4.3.3 Gate Application

Grammar

Semantic Analysis

Code Generation

4.3.4 Control Flow

Grammar

Code Generation

4.3.5 Expressions

Grammar

- Consists of expressions, terms and factors
  - Expressions consist of expression, operator, and term or just a term
  - Term consists of term, operator, and factor or just a factor
  - Factor consists of expression in parentheses, a negated factor, number, identifier or function call
- Inherent order of operations

Evaluation

- All expressions evaluated at compile time
- All expressions inherit from abstract `Expression` class, which has an `Evaluate` method
- Generic return type `T`
- ...

4.4 Optimization

4.4.1 Constant folding

- Inherent in the language
- All variables known at compile time
- Any expression is evaluated at compile time

4.4.2 Peephole optimization

- Not implemented, but planed
- Replace sequences of gates with more efficient ones

4.5 Test Cases

- Different test categories
- How are they implemented?
- What do they test?
- (Continuous integration)

5 Conclusion and Future Work

- Conclusion to thesis
- Future work
  - how could language be extended

Bibliography

[ACR*10] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any and-or formula of size n can be evaluated in time $n^1/2+o(1)$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010.

[AlGr05] T. Altenkirch and J. Grattage. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 249–258. IEEE, 2005.

[BrBr01] Jean-Luc Brylinski and Ranee Brylinski. Universal quantum gates.

[BGB*18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4), 2018.

[BeLa17] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

[BeVa93] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In Rao Kosaraju, David Johnson, and Alok Aggarwal, editors, *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing - STOC '93*, pages 11–20, New York, New York, USA, 1993. ACM Press.

[CFM17] Frederic T. Chong, Diana Franklin, and Margaret Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180–187, 2017.

[Copp02] D. Coppersmith. An approximate fourier transform useful in quantum factoring.

[DiCh20] Yongshan Ding and Frederic T. Chong. *Quantum Computer Systems*. Springer International Publishing, Cham, 2020.

[Dijk75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[DeJo92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[DMN13] Simon J. Devitt, William J. Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on progress in physics. Physical Society (Great Britain)*, 76(7):076001, 2013.

[Drap00] Thomas G. Draper. Addition on a quantum computer.

[FYY13] Yuan Feng, Nengkun Yu, and Mingsheng Ying. Reachability analysis of recursive quantum markov chains. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Krishnendu Chatterjee, and Jirí Sgall, editors, *Mathematical Foundations*

of Computer Science 2013, volume 8087 of *Lecture Notes in Computer Science*, pages 385–396. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[HHHH09] Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of Modern Physics*, 81(2):865–942, 2009.

[KuBr00] Arun Kumar Pati and Samuel L. Braunstein. Impossibility of deleting an unknown quantum state. *Nature*, 404(6774):164–165, 2000.

[Land61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[LoCh19] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019.

[MHH19] Gary J. Mooney, Charles D. Hill, and Lloyd C. L. Hollenberg. Entanglement in a 20-qubit superconducting quantum computer. *Scientific reports*, 9(1):13465, 2019.

[MVZJ18] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. 2018.

[Niel06] Michael A. Nielsen. Cluster-state quantum computation. *Reports on Mathematical Physics*, 57(1):147–161, 2006.

[RDB*22] Roman Rietsche, Christian Dremel, Samuel Bosch, Léa Steinacker, Miriam Meckel, and Jan-Marco Leimeister. Quantum computing. *Electronic Markets*, 32(4):2525–2536, 2022.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Shor97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[van20] John van de Wetering. Zx-calculus for the working quantum computer scientist.

[Wino78] S. Winograd. On computing the discrete fourier transform. *Mathematics of Computation*, 32(141):175–199, 1978.

[Wine13] David J. Wineland. Nobel lecture: Superposition, entanglement, and raising schrödinger's cat. *Reviews of Modern Physics*, 85(3):1103–1114, 2013.

[WoZu82] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[Ying11] Mingsheng Ying. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):1–49, 2011.

[YVC24] Charles Yuan, Agnes Villanyi, and Michael Carbin. Quantum control machine: The limits of control flow in quantum programming. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1–28, 2024.

[YYF12] Mingsheng Ying, Nengkun Yu, and Yuan Feng. Defining quantum control flow.