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

Master Thesis Proposal

Title tbd

Sascha Thiemann
Matr.-No.: 406187
Study Program: Master Computer Science
June 17, 2024

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

1 Introduction

- Short intro into QC
  - What is QC
  - Why is it important
  - What can it be used for

2 Motivation

With the emergence of quantum computing, many quantum languages were introduced. Most languages focus on a lower level representation of quantum circuits. An example would be the popular Open Quantum Assembly Language (QASM)[CBSG17]. QASM consists mainly of quantum and classical gates that can be manipulated by predefined and composite gates as well as limited (classical) if-statements. There are also languages with a focus on high level interactions, e.g. Tower[ChMi22] which contains data structures in superposition, and Silq [BBGV20] which allows for automatic uncomputing of registers. What all these languages have in common is the restriction to quantum data while using only classical control flow. Although quantum control flow was defined by Ying et al. [YYF12] over 10 years ago, only very few languages have incorporated the principle. One example is the functional programming language proposed by Altenkirch et al. [AlGr05] where `if°` is used to define the Hadamar gate. Only recently was the Quantum Control Machine (QCM) with quantum control flow at its core proposed by Yuan et al. [YVC24].

The QCMs syntax and logic are both heavily influenced by classical assembly languages. The language consists of quantum registers, gate, swap and get-bit operations[1], simple numeric operations on registers, and, finally, jump instructions. The jump instructions range from simple to conditional to indirect and are used to enable quantum control flow. Although the jump instructions are basic on jumps in classical computers, they are heavily limited by two concepts quantum computers based on unitary gates must adhere to, *reversibility* and *synchronization*. [YVC24]

Because quantum computers are based on unitary gates, all there operations need to be unitary and, therefore, reversible as well. This also includes jump instructions which are not reversible in classical computers. To ensure reversibility of jumps, the

Example for non-reversibility?

[1]The gate operations are limited to the Hadamar and NOT gates.

```
     add    res  $1
     add    r1   y
l1:  rjne   l3   r1   y
l2:  jz     l4   r1
     mul    res  x
     radd   r1   $1
l3:  jmp    l1
l4:  rjmp   l2
```

Figure 2.1: QCM exponentiation without synchronization

```
     add    res  $1
     add    r1   max
l1:  rjne   l3   r1   max
l2:  jz     l4   r1
l5:  jg     l7   r1   y
     mul    res  x
l6:  jmp    l8
l7:  rjmp   l5
     nop                  ; padding
l8:  rjle   l6   r1   y
     radd   r1   $1
l3:  jmp    l1
l4:  rjmp   l2
```

Figure 2.2: Synchronized QCM exponentiation

QCM uses a *branch control register* which values controls how much the instruction pointer of the machine advances after an execution. The branch control register can then be manipulated reversibly. The idea of a branch control register can also be found in reversible architectures for classical machines[AGY07, TAG12].

Although such a program counter addresses the issue of reversibility, it can become entangled with data registers when in superposition. This can lead to disruptive entanglement where the output of the program becomes invalid. [YVC24] To prevent any disruptive entanglement of the data and control registers, the QCM adheres to the principle of synchronization. It requires that the control flow is separated from the data at the end of execution. Examples where synchronization comes into play are given in Fig. 2.1 and Fig. 2.2 where x^y and $x^{\mathtt{min}\{y,max\}}$ are calculated respectively. While the first example is completely reversible, it does not adhere to the principle of synchronization. Given the inputs two different inputs, the loop will be executed a different amount of times. This means that after the faster input completed the loop, the program counter of the slower input cannot catch up. To prevent this issue, the second program uses padding which is executed instead of the main loop.

Because of the reversibility of the QCM, any jump instruction in the code needs to have an opposing return jump instruction. Additionally, the synchronization principle requires any loop with n instructions to contain n padding instructions and any loops cannot depend on quantum data for its iterations. Together with the syntax based on classical assembly languages, the language of the QCM is hard to read and write.

3 Concept

- Idea Explain in concept
  - reduce QCM to basics
  - lead to concept section
- Language features: qif-else, bounded loops, (boolean eval)
- Translation to quasm
- overall (more) realistic for NISQ
- Further (compliler optimizations)
- Example grammar

Bibliography

[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.

[AGY07] Holger Bock Axelsen, Robert Glück, and Tetsuo Yokoyama. Reversible machine code and its abstract processor architecture. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science – Theory and Applications*, volume 4649 of *Lecture Notes in Computer Science*, pages 56–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BBGV20] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–300, New York, NY, USA, 2020. ACM.

[CBSG17] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language.

[ChMi22] Charles Yuan and Michael Carbin. Tower: Data structures in quantum superposition, 2022.

[TAG12] Michael Kirkedal Thomsen, Holger Bock Axelsen, and Robert Glück. A reversible processor architecture and its reversible logic design. 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, Alexis de Vos, and Robert Wille, editors, *Reversible Computation*, volume 7165 of *Lecture Notes in Computer Science*, pages 30–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[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.

```
grammar Luie;

parse
 : block EOF
 ;

block
 : (definition | statement)*
 ;

definition
 : 'qubit' IDENTIFIER ';'
 ;

statement
 : GATE IDENTIFIER ';'
 | qifStatement
 ;

qifStatement
 : ifStat elseStat? END
 ;

ifStat
 : IF IDENTIFIER DO block
 ;

elseStat
 : ELSE DO block
 ;

GATE
 : XGATE
 | ZGATE
 | HGATE
 ;

XGATE : 'x';
ZGATE : 'z';
HGATE : 'h';

IF      : 'qif';
ELSE    : 'else';
DO      : 'do';
END      : 'end';

IDENTIFIER
 : [a-zA-Z_] [a-zA-Z_0-9]*
 ;

COMMENT
 : ( '//' ~[\r\n]* | '/*' .*? '*/' ) ->
     skip
```

```
    ;

  SPACE
    : [ \t\r\n\u000C] -> skip
    ;
```