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.
July 16, 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

- Introduction with random citation to not cause error [ACR*10]

2 Background

... Background

2.1 Quantum Computing

- Introduction into quantum computing
- From bits to qubits
- Entanglement

2.1.1 Entanglement

- Explain entanglement
- How is entanglement relevant for quantum computing?
- Explain disruptive entanglement

2.1.2 Operators and Gates

- General theoretical bases of operators and gates
- Most important gates and their functions

2.1.3 Measurement

- How are qubits measured?
- What is the effect of measurement on qubits?

2.1.4 Relevant Algorithms

2.2 Quantum Control Flow

- Introduction into quantum control flow
- Branching
- Iteration
- Limitations

2.2.1 Branching

- Explain branching principle
- How is branching relevant for control flow?
  - Example of branching in classical computing
  - Example of branching in quantum computing

2.2.2 Iteration

2.2.3 Limitations

Reversibility

- Explain reversibility principle
- How is reversibility relevant for control flow?
  - Example of reversible and irreversible operations (JMP instructions)
-

Synchronization

- Explain synchronization principle
- Tortoise and hare example

2.2.4 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

```
      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

2.3 Quantum Languages

2.3.1 QASM Language

- Give overview of QASM language and concepts

2.4 Compilation

2.4.1 Lexer

2.4.2 Parser

2.4.3 Semantic Analysis

2.4.4 Code Generation

2.4.5 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

```
grammar Luie;

parse
 : block EOF
 ;

block
 : (definition | statement)*
 ;
```

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

```
declaration
 : 'qubit' ('[' size=expression ']')?
     IDENTIFIER ';'
 ;

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

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

- `Definitions` 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

Evaluation

4.4 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 \$nˆ1/2+o(1)\$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010.

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