Queasy/KetPsi

Maharshi Kadeval
Rajiv Chitale
Rishit D
Vedant Bhandare

Language Specifications for DSL
CS3423: Compilers - 2
Department of Computer Science & Engineering
Indian Institute of Technology, Hyderabad

November 2023

Supervisor Dr. Jyothi Vedurada

Contents

Chapter 1

Aim & Program Flow

1.1 DSL Objectives

This program helps interested users define a Quantum-circuit, composed of:

- Quantum-Registers
- Classical-Registers
- Quantum-Gates
- Measurement Boxes

Users can construct circuits of their choice using predefined gates and also define blocks of gates for further usage. Users can also simulate measurements and accordingly store them in necessary Classical-Registers. Users can also use Classical-Registers to condition the use of gates and control flow of the circuit.

Users have the option to manipulate their data collected in the Clasical-Registers over multiple iterations over the circuit with specified input qubits. Users can compare these values with their assumed distributions and describe error, variance and other parameters. Users can also do basic arithmetic operations on these output-vectors.

1.2 Program

The basic program consists of three sections:

- `init`

- `main` denoted by `\begin` and `\end`
- `output`

Please find below a sample program that illustrates quantum teleportation.

```
\begin_init
    #registers quantum = 3
    #registers classical = 3
    #iters = 1000

    #set classical states -> 0,0,0
    #set quantum states -> {1,0}, {1,0}, {(0, 0), (0, 1)}

    block BellGen (i, j) -> j [
        $ Start with Hadamard $
        H -> i
        $ Apply C-X $
        X : i -> j
    ]

\end_init

\begin_main
    $ Generate Bell State $
    BellGen (1, 2)

    \barrier

    $ Init Transformations $
    X : 0 -> 1
    H -> 0

    $ Measurements $
    measure: 0 -> 0
    measure: 1 -> 1

    $ Classical Controlled Ops $
    1 ? X -> 2
    0 ? Z -> 2

\end_main

\begin_output

    $ Save Value $
    \save "output.csv"

```

```
    $ Print the quantum register $
    $ echo("Final Transferred State: ", q_output[2]) $

    $ Print the frequency list of classical registers $
    echo("Classical Register List: ", c_output)

    $ Compute Fidelity $
    state original = {(0, 0), (0, 1)}

    int a = 1
   $ int a = 2 $
    int i = 0

    for i in (1:2){
        int a = 3
        int j = 0
        $ int j = 1 $
        for j in (4:6){
            int a = 7
            int j = 8
            a = a + 1
        }
        int y = 9
    }

\end_output
```

Listing 1.1: Quantum Teleportation

Chapter 2

Structure

2.1 `init` Section

This section contains three necessary statements that provide the compiler with necessary information about qbits

```
1       #registers quantum = 3
2       #registers classical = 2
3       #iters = 1000
```

to set number of quantum registers, set number of classical registers and set number of iterations, respectively.

Optional statements include initializing states of quantum registers:

```
1    #set quantum states -> (2,3), (0,1), (5,5)
2    #set classical states -> 0, 1
```

This sets the state of first quantum register as $2 + 3i$, second one as i and third one as $5 + 5i$ and sets the state of first classical register as 0 and second one as 1.

This section will also have user defined code block definitions (refer to block definitions section)

```
1    block (i,j,k) -> (k) {
2        $
3            statement calls
4        $
5    }
```

2.2 `main` Section

\begin marks the beginning of the main section and \end marks the end of the main section. This section includes

- calls to pre-defined or user-defined *blocks* (refer to block definitions section)
- \barrier: It's a directive to the compiler to not combine blocks before and after \barrier
- *measure* calls block that reads the value of a register and stores it in another register (both registers provided by the programmer)
- condition-otherwise for conditional statements
- *for*, *for_lex* and *for_zip* loops for repetitive calls to blocks.

2.3 `output` Section

This section contains the features of any general programming language to help interpret the results of the quantum experiment further and better understand the results from repeated runs of the circuit.
This section mostly contains C-like features but syntactic sugar for certain operations and loops. We also support more datatypes for vector and array manipulation.

Chapter 3

Compilation Steps

3.1 Steps to Run

In the ./code folder use the following command,

```
1   make run SRC=t1.txt
```

The variable PHASE in the makefile can be set to II, III or IV according to the phase whose test cases are to be run.

3.2 Compilation Process

The makefile does the following:

- Cleans the ./code/exec directory
- Generates the lexer lex.yy.c
- Generates the parser y.tab.c
- Compiles them into an executable, parser
- Compiles header files required by generated code
- Compiles parser with libraries into a.out
- Runs a.out on testcase given as SRC

The results are produced in ./code/exec

- Target code out.cpp

- Parsing steps in output.parsed
- Tokens in tokens.txt