QLang: Qubit Language
(Reference Manual)

Christopher Campbell Clément Canonne Sankalpa Khadka Winnie Narang
Jonathan Wong

October 26, 2014

Contents

1 Introduction

2 Lexical conventions

There are five kinds of tokens: identifiers, keywords, constants, expression operators, and other separators. There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

Rephrase: that's plagiarism

2.1 Character set

`QLang` supports a subset of ASCII; that is, allowed characters are `a-zA-Z0-9@#,-_;:()[]{}<>=+/|*`, as well as tabulations `\t`, spaces, and line returns `\n` and \r.

2.2 Comments

Comments start with a # sign, which then extends until the next carriage return. Multiline comments are not supported.

2.3 Identifier (names)

An identifier is an arbitrarily long sequence of alphabetic and numeric characters, where `_` is included as "alphabetic". It must start with a lowercase or uppercase letter, i.e. one of `a-zA-Z`.
The language is case-sensitive: `hullabaloo` and `hullABaLoo` are considered as different.

2.4 Keywords

The following identifiers as reserved for keywords, and no one shall use them because it's forbidden and uncool.

```
pi e
int float comp rvect cvect mat
true false
if elif else
def for from to by while break
or and xor
not re im norm isunit trans det adj conj sin cos tan
```

2.5 Constants

There are three sorts of constants in the language, namely *integer*, *complex* and *identifier* constants. The first are comprised of any sequence of integers of the form `0|([1-9][0-9]*)` (recall that integers are non-negative), and have type `int`. The second are of type `com` and have the form R`|`R`+`R`i|`R`i` where R consists of a (i) sign, (ii) an integer part followed by (iii) a point, (iv) a decimal part, then (v) either a `e` or a `E` followed by an exponent part, possibly signed. (i) and (v) are optional, and either (ii) or (iv) can be missing as well. In more detail, R is

defined as `[+-]{0,1}(((`A`.`B`*|.`B`+)([eE][+-]?B+)?)|`A`[eE][+-]?B+)` and A =`0|([1-9]`B`*)`, B =`0|[1-9]` (that is, R matches a real number such as `2.78e5`, `1.5E-1` or `10.25`). check this paragraph.
Finally, the identifier constants are a subset of the reserved keywords, and include:

`e` the base of natural logarithm $e = \sum_{k=0}^{\infty} \frac{1}{k!}$. Equivalent to `exp(1)`; has type `com`.

`Pi` the constant π. Has type `com`.

`true` represents the Boolean value `true`. Stored internally as `int` 1.

`false` represents the Boolean value `false`. Stored internally as `int` 0.

3 Syntax notation

4 What's in a Name?

5 Objects and lvalues

6 Conversions

7 Expressions

7.1 Operator Precedence

The remaining major subsections of this section describe the groups of expression operators, while the minor subsections describe the individual operators within a group.

7.2 Primary Expressions

7.2.1 identifier

Identifiers are primary expressions. All identifiers have an associated type that is given to them upon declaration (e.g. float ident; declares an identifier named ident that is of type float).

7.2.2 constant

Constants are primary expressions. As discussed above, constants are integers, floats, complex numbers, qubits, and matrices. All constants have an associated type that are predetermined by the QLang language (e.g. integers are of type int).

7.2.3 (expression)

Parenthesized expressions are primary expressions. The type and value of a parenthesized expression is the same as the type and value of the expression without parenthesis. Parentheses allow expressions to be evaluated in a desired precedence. Parenthesized expressions are evaluated relative to each other starting with the expression that is nested the most deeply and ending with the expression that is nested the least deeply (i.e. the shallowest).

7.2.4 primary-expression[expression]

Primary expressions followed by expressions in brackets are primary expressions. Such primary expressions represent indexing into an array, where the primary expression is the array and the expression in brackets is the index. Because indicies must be of type int, the expression in brackets must evaluate to an int. The evaluation of the overall primary expression should give the value associated with the given index of the array.

7.2.5 primary-expression(expression-list)

Primary expressions followed by a parenthesized expression list are primary expressions. The expression list is mandatory and must consist of one or more expressions separated by commas. Such primary expressions can be used in the declaration of functions or function calls. If being used in function declarations, they must be preceded by the correct function declaration syntax and each expression in the expression list must evaluate to a type followed by an identifier. If being used in function calls each expression in the expression list must evaluate to an identifier.

7.3 Unary Operators

7.3.1 not expression

The result is the negative of the expression. The type of the expression must be int or float. The result has the same type as the expression.

7.3.2 re expression

The result is the real component of the expression. The type of the expression must be comp. The result has the same type as the expression (it is a complex number with 0 imaginary component).

7.3.3 im expression

The result is the imaginary component of the expression. The type of the expression must be comp. The result has the same type as the expression (it is a complex number with 0 real component).

7.3.4 norm expression

The result is the norm of the expression. The type of the expression must be rvect or cvect. The result has type int if the expression is an integer vector, type float if the expression is a float vector, and type comp if the expression is a complex number vector.

7.3.5 isunit expression

The result is a boolean indicating if it is true or false that the expression is a unit vector or unit matrix. The type of the expression must be rvect, cvect or mat.

7.3.6 trans expression

The result is the transpose of the expression. The type of the expression must be rvect, cvect or mat. The result has type cvect if the expression is of type rvect, has type rvect if the expression is of type cvect, and has type mat if the expression is of type mat.

7.3.7 det expression

The result is the determinant of the expression. The type of the expression must be mat. The result has type int if the expression is an integer matrix, type float if the expression was a float matrix, and type comp if the expression was a complex number matrix.

7.3.8 adj expression

The result is the adjucate of the expression. The type of the expression must be mat. The result has the same type as the expression.

7.3.9 conj expression

The result is the complex conjugate of the expression. The type of the expression must be comp or mat. The result has type mat. The result has the same type as the expression.

7.3.10 sin expression

The result is the evaluation of the trigonometric function sine on the expression. The type of the expression must be int, float, or comp. The result has type float if the expression is of type int or float and type comp if the expression is of type comp.

7.3.11 cos expression

The result is the evaluation of the trigonometric function cosine on the expression. The type of the expression must be int, float, or comp. The result has type float if the expression is of type int or float and type comp if the expression is of type comp.

7.3.12 tan expression

The result is the evaluation of the trigonometric function tangent on the expression. The type of the expression must be int, float, or comp. The result has type float if the expression is of type int or float and type comp if the expression is of type comp.

7.4 Binary Operators

7.4.1 expression ˆ expression

The result is the exponentiation of the first expression by the second expression. The types of the expressions must be of type int, float, or comp. If the expressions are of the same type, the result has the same type as the expressions. Otherwise, if at least one expression is a comp, the result is of type comp; if neither expressions are comp, but at least one is float, the result is of type float.

7.4.2 expression * expression

The result is the product of the expressions. The type considerations are the same as they are for *expression^ expression*

7.4.3 expression / expression

The result is the quotient of the expressions, where the first expression is the dividend and the second is the divisor. The type considerations are the same as they are for *expression^ expression.* Integer division is rounded towards 0 and truncated.

7.4.4 expression % expression

The result is the remainder of the division of the expressions, where the first expression is the dividend and the second is the divisor. The sign of the dividend and the divisor are ignored, so the result returned is always the remainder of the absolute value of the dividend divided by the absolute value of the divisor. The type considerations are the same as they are for *expression^ expression.*

7.4.5 expression + expression

The result is the sum of the expressions. The types of the expressions must be of type int, float, comp, or qubit. Otherwise, if at least one expression is a comp, the result is of type comp; if neither expressions are comp, but at least one is float, the result is of type float. Qubits are special and can only be summed with each other.

7.4.6 expression - expression

The result is the difference of the first and second expressions. The type considerations are the same as they are for *expression - expression.*

7.4.7 expression @ expression

The result is the tensor product of the first and second expressions. The The expressions must be of type of mat. The result has the same type as the expression.

7.4.8 expression eq expression

The result is a boolean indicating if it is true or false that the two expression are structurally equivalent. The type of the expressions must be the same.

7.4.9 expression lt expression

The result is a boolean indicating if it is true or false that the first expression is less than the second. The type of the expressions must be int, float, or comp and must be the same.

7.4.10 expression gt expression

The result is a boolean indicating if it is true or false that the first expression is greater than the second. The type of the expressions must be int, float, or comp and must be the same.

7.4.11 expression leq expression

The result is a boolean indicating if it is true or false that the first expression is less than or equal to the second. The type of the expressions must be int, float, or comp and must be the same.

7.4.12 expression geq expression

The result is a boolean indicating if it is true or false that the first expression is greater than or equal to the second. The type of the expressions must be int, float, or comp and must be the same.

7.4.13 expression or expression

The result is a boolean indicating the logical *or* of the expressions. The type of the expressions must be int or float and must be the same. In the expressions, 0 is considered false and all other values are considered true.

7.4.14 expression and expression

The result is a boolean indicating the logical *or* of the expressions. The type of the expressions must be int or float and must be the same. In the expressions, 0 is considered false and all other values are considered true.

7.4.15 expression xor expression

The result is a boolean indicating the logical *or* of the expressions. The type of the expressions must be int, float and must be the same. In the expressions, 0 is considered false and all other values are considered true.

7.5 Assignment Operators

Assignment operators have left associativity

7.5.1 lvalue = expression

The result is the assignment of the expression to the lvalue. The lvalue must have been previously declared. The type of the expression must be of the same that the lvalue was declared as. Recall, lvalues can be declared as int, float, comp, rvect, cvect, mat, and qubit.

8 Declarations

9 Statements

10 External definitions

11 Scope rules

12 Compiler control lines

13 Implicit declarations

14 Types revisited

15 Constant expressions

16 Examples

We present some examples that illustrates the use of Qlang in solving quantum computing problems.

16.1 Solving Quantum Computation Problem

16.1.1 Problem1

Evaluate the following expressions: a. $(H \otimes S)|00\rangle$ b. $\langle 101|000\rangle$ c. $\langle 01|H \otimes H|01\rangle$

16.1.2 Problem 2

Find the matrix corresponding to the quantum circuit:

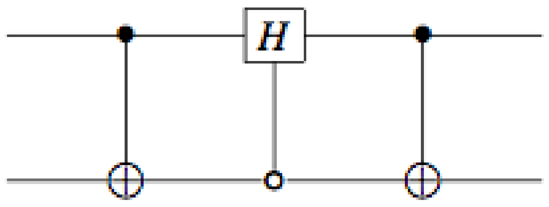

Figure 1: Quantum Circuit implementing series of control gates

```
def circuitMat = findMatrix (){
        qubit mat0=|00>;
        qubit mat1=|01>;
        qubit mat2=|10>;
        qubit mat3=|11>;
CNOT = [1,0,0,0;0,1,0,0;0,0,0,1;0,0,1,0]
HNOT = [1/sqrt(2),0,0,1/sqrt(2);0,1,0,0;1/sqrt(2),0,1,-1/sqrt(2);0,0,0,0]
allGates = CNOT * HNOT * CNOT

circuitMat =[allGates*mat0:allGates*mat1:allGates*mat2:allGates*mat3]

}
```

16.1.3 Problem 3

Consider the circuit and show the probabilities of outcome 0 and 1.

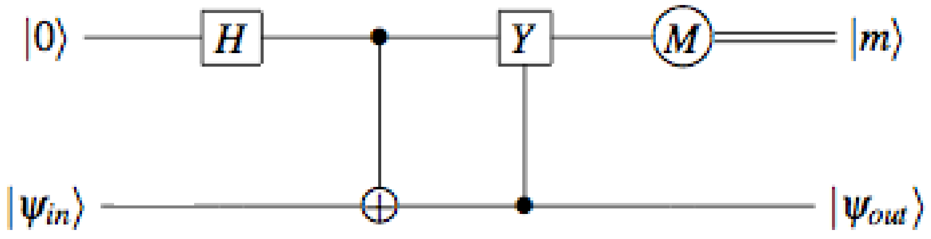

Figure 2: Quantum Circuit

16.2 Deutsch Jozsa Algorithm

```
def outcome = deutschJozsa(qubit in, mat U){

        mat qubitInput;
        qubitInput = in @ |1>;
        input = (H @ H)*input;
        input = U * input;
        input = (H @ I)*input;
        input=(in*Adj(in)@ I)*input;

        if (input == 0){
                outcome = 0;
        }
        else{
                outcome = 1;
        }
}
```