QCLANG: Quantum Computing Language
QCLANG → OpenQASM

Connor Abbott       Klint Qinami
cwa2112@columbia.edu  kq2129@columbia.edu

Columbia University in the City of New York

April 20th, 2018
QCLANG is a high-level programming language for quantum computation.

QCLANG programs compile to the OpenQASM intermediate representation.

QCLANG supports classical data, which can be read, written, and duplicated as usual, and quantum data, which supports unitary transformations and measurements as primitives.
import Quipper

spos :: Bool -> Circ Qubit
spos b = do q <- qinit b
          r <- hadamard q
          return r

Imperative

- QCL: Quantum computation language. Supports user defined operators and functions, C-like syntax.
- QMASM: Low-level language for quantum annealers like D-Wave.
- Q language: C++ extension. Operators like QHadamard and QNot can be defined using C++ classes.

Functional

- QML: Haskell-like language. Introduces both classical and quantum control operators.
- Quipper: Embedded language within Haskell.
- LIQUi⟩: F# extension and toolsuite with quantum simulators.
- Compiler written in OCaml. Classical data is interpreted directly by the compiler, while quantum data is written out to OpenQASM.
- Both classical and quantum dialects are unified through language semantics.
QCLANG: Language Features

- Interact with qubits similarly to regular bits (pass by value)
  - Uses an affine type constraint to enforce no-cloning
- Familiar syntax (C-like)
Safety Invariants: Affine Type Systems

No-cloning theorem

There is no unitary operator $U$ on $H \otimes H$ such that for all normalized states $|\phi\rangle_A$ and $|e\rangle_B$ in $H$

$$U(|\phi\rangle_A|e\rangle_B) = e^{i\alpha(\phi,e)}|\phi\rangle_A|\phi\rangle_B$$

for some real number $\alpha$ depending on $\phi$ and $e$.

No-cloning in logic

In logic, no-cloning and no-deleting correspond to disallowing two rules of inference: the rule of weakening and the rule of contraction.

**Weakening:** $\Gamma \vdash C \implies \Gamma, A \vdash C$

**Contraction:** $\Gamma, A, A \vdash C \implies \Gamma, A \vdash C$
- Dropping contraction as a structural system rule gives a sub-structural type system.
- Affine type systems allow exchange and weakening, not contraction. Effectively, every variable is used at most once.
- In QClang, implemented at runtime.
```
[~] $ cat tests/fail-qubit1.qc

int main() {
    qubit a;
    qubit b;
    qubit c;
    a = b;
    c = b;
}
```
1 [klint@xps13:QClang] $ ./qclang.native tests/fail-qubit1.qc
2 OPENQASM 2.0;
3 include "qelib1.inc";
4 qreg q[1];
5 creg c[1];
6 h q;
7 qreg n_a_q0[1];
8 qreg n_b_q0[1];
9 qreg n_c_q0[1];
10 Fatal error: exception Failure("qubit b used more than once")
11 [klint@xps13:QClang] $
```c
int main() {
    qubit q1;
    qubit q2;
    tup(qubit, qubit) tuple;

    /* entangle q1 and q2 */
    tuple = CX(hadamard(q1), q2);
    /* unentangle */
    tuple = CX(tuple[0], tuple[1]);
    q1 = hadamard(tuple[0]);
    q2 = hadamard(tuple[1]);
    measure(q1);
    measure(q2);
    return 0;
}
```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[1];
creg c[1];
h q;
qreg n_q1_q0 [1];
qreg n_q2_q0 [1];
qreg n_tuple_0_q0 [1];
qreg n_tuple_1_q0 [1];
h n_q1_q0;
cx n_q1_q0 , n_q2_q0;
cx n_q1_q0 , n_q2_q0;
h n_q1_q0;
h n_q2_q0;
creg n_q1_q0_mb0 [1];
measure n_q1_q0 \rightarrow n_q1_q0_mb0;
creg n_q2_q0_mb1 [1];
measure n_q2_q0 \rightarrow n_q2_q0_mb1;
Examples: python3 run_qasm.py tests/test-cx.out

COMPLETED
{"1 0 0": 48, '0 0 0': 52}
Examples: Deutsch-Jozsa Quantum Circuit

\[ |0\rangle \xrightarrow{n} H^{\otimes n} \xrightarrow{} U_f \xrightarrow{} H^{\otimes n} \xrightarrow{} \]

\[ |1\rangle \xrightarrow{} H \xrightarrow{} \]

Figure: D-J Circuit
qubit oracle(qubit q) {
    return !q;
}

int main() {
    /* QClang implementation of Deutsch-Josza Algorithm */
    int i;
    int strlen;
    qubit[] test_bits;
    bit[] measure_bits;
    qubit answer;

    /* Create Qubit array for oracle query */
    strlen = 10;
    test_bits = new qubit[](strlen);
    measure_bits = new bit[](strlen);
/* Create superposition state */
answer = true;
answer = hadamard(answer);
for (i = 0; i < strlen; i = i + 1) {
    test_bits[i] = hadamard(test_bits[i]);
}

/* Query oracle */
answer = oracle(answer);

/* Apply hadamard again */
for (i = 0; i < strlen; i = i + 1) {
    test_bits[i] = hadamard(test_bits[i]);
}

/* Measure */
for (i = 0; i < strlen; i = i + 1) {
    measure_bits[i] = measure(test_bits[i]);
}
OPENQASM 2.0;
#include "qelib1.inc";
qreg q[1];
creg c[1];
h q;
qreg n_answer_q0[1];
qreg temp0.q1[1];
qreg temp1.q2[1];
qreg temp2.q3[1];
qreg temp3.q4[1];
qreg temp4.q5[1];
qreg temp5.q6[1];
qreg temp6.q7[1];
qreg temp7.q8[1];
qreg temp8.q9[1];
qreg temp9.q10[1];
creg temp10.b11[1];
creg temp11.b12[1];
creg temp12.b13[1];
creg temp13.b14[1];
creg temp14.b15[1];
creg temp15.b16[1];
creg temp16.b17[1];
creg temp17.b18[1];
creg temp18.b19[1];
creg temp19.b20[1];
h temp9.q10;
h temp8.q9;
h temp7.q8;
h temp6.q7;
h temp5.q6;
h temp4.q5;
h temp3.q4;
h temp2.q3;
h temp1.q2;
h temp0.q1;
```
37  x n_answer_q0 [0];
38  h n_answer_q0;
39  x n_answer_q0 [0];
40  h temp9_q10;
41  h temp8_q9;
42  h temp7_q8;
43  h temp6_q7;
44  h temp5_q6;
45  h temp4_q5;
46  h temp3_q4;
47  h temp2_q3;
48  h temp1_q2;
49  h temp0_q1;
50  creg temp9_q10_mb21 [1];
51  measure temp9_q10 -> temp9_q10_mb21;
52  creg temp8_q9_mb22 [1];
53  measure temp8_q9 -> temp8_q9_mb22;
54  creg temp7_q8_mb23 [1];
55  measure temp7_q8 -> temp7_q8_mb23;
56  creg temp6_q7_mb24 [1];
57  measure temp6_q7 -> temp6_q7_mb24;
58  creg temp5_q6_mb25 [1];
59  measure temp5_q6 -> temp5_q6_mb25;
60  creg temp4_q5_mb26 [1];
61  measure temp4_q5 -> temp4_q5_mb26;
62  creg temp3_q4_mb27 [1];
63  measure temp3_q4 -> temp3_q4_mb27;
64  creg temp2_q3_mb28 [1];
65  measure temp2_q3 -> temp2_q3_mb28;
66  creg temp1_q2_mb29 [1];
67  measure temp1_q2 -> temp1_q2_mb29;
68  creg temp0_q1_mb30 [1];
69  measure temp0_q1 -> temp0_q1_mb30;
```
Examples: python3 run_qasm.py tests/Deutsch-Jozsa.out

COMPLETED

{
'0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0': 100
}
To do

- Vectorize measure, hadamard, and everything else
- Barrier
- Exact pi constant
- If statements with qubits
- Unify quantum device measurement outputs with QClang code
- Drop affine type constraint?