Lecture 4:

Quantum Computers

Gates, circuits and programming

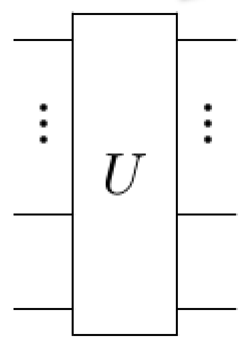
Quantum gates

Quantum gates

- The same way
  classical gates manipulate only a few bits at a time,
  quantum gates manipulate only a few qubits at a time
  - Usually represented as **unitary matrices** we already saw
- Circuit representation

...boxes and different symbols depict operations on qubits

Wires depict qubits



...inheritence of classical computing –
it is better to think of qubits as particles
and gates as physical processes applied to those particles

Pauli-X gate

- Acts on a single qubit

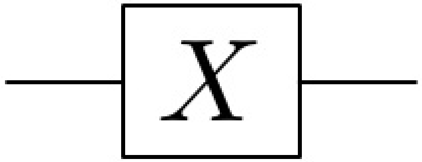
Dirac notation

$$|0\rangle \rightarrow |1\rangle, \quad |1\rangle \rightarrow |0\rangle$$

Matrix representation

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Circuit representation



  - Acting on pure states becomes a **classical *NOT*** gate

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow X\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \cdot 1 + 1 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow X\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \cdot 0 + 1 \cdot 1 \\ 1 \cdot 0 + 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Dirac notation…

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

…is obviously more convenient for calculus

Pauli-X gate

– Acting on a general qubit state

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$X|\psi\rangle = \alpha|1\rangle + \beta|0\rangle = \beta|0\rangle + \alpha|1\rangle$$

– It is its own inverse

$$XX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

Hadamard gate

- Acts on a single qubit
  - Corresponding to the Hadamard transform we already saw

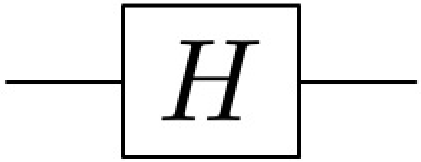
Dirac notation

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Unitary matrix

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Circuit representation



...obviously, no classical equivalent

  - One of the most important gates for quantum computing

Hadamard gate

- An interesting example

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

$$|\alpha_0|^2 = \frac{1}{2} \quad |\alpha_1|^2 = \frac{1}{2}$$

Acting on pure states…

…gives a balanced superposition…

…both states, if measured,
give either 0 or 1 with equal probability

Hadamard gate

– Applying another Hadamard gate
  - *to the first result*

$$H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = |0\rangle$$

  - *to the second result*

$$H\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) - \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$H\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = |1\rangle$$

Hadamard gate

Both states give equal probabilities when measured…

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \qquad\qquad \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

…but when Hadamard transformation is applied
it produces two different states

- The example gives an answer to the question asked before – why state of the system
  has to be specified with complex amplitudes
  and cannot be specified with probabilities only

Pauli-Y gate

- Acts on a single qubit

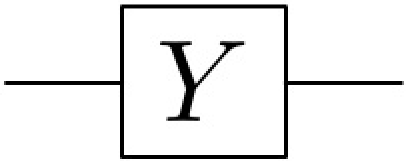
Dirac notation

$$|0\rangle \rightarrow i|1\rangle, \quad |1\rangle \rightarrow -i|0\rangle$$

Matrix representation

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Circuit representation



...another gate with no classical equivalent

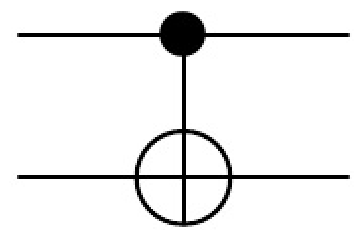
CNOT gate

- ***Controlled NOT*** gate
- Acts on two qubits

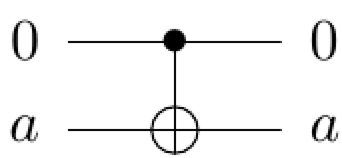
Matrix representation

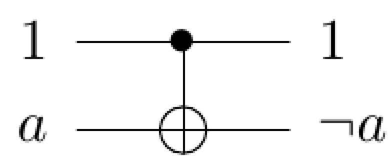
$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Circuit representation



  - Classical gate operation





CNOT gate

– Example of acting on a superposition

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$|00\rangle \to |00\rangle, \quad |01\rangle \to |01\rangle, \quad |10\rangle \to |11\rangle, \quad |11\rangle \to |10\rangle$$

$$\Downarrow$$

$$CNOT|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

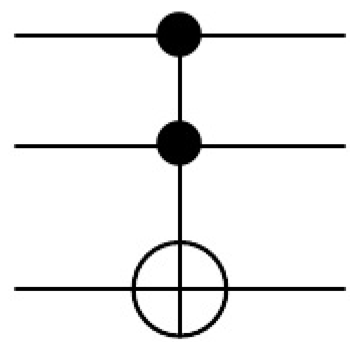
Toffoli gate

- Also called ***Controlled Controlled NOT***
- Acts on three qubits

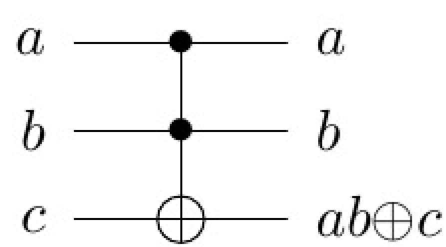
Matrix representation

$$TOFFOLI = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Circuit representation



  - Classical gate operation



Quantum circuits

Universal set of quantum gates

- There is more than one universal set of gates for classical computing
- What about **quantum computing**, is there a **universal set of gates** to which any quantum operation possible can be reduced to?

Universal set of quantum gates

- No, but any unitary transformation can be **approximated to arbitrary accuracy** using a universal gate set
  - For example *(H, S, T, CNOT)*

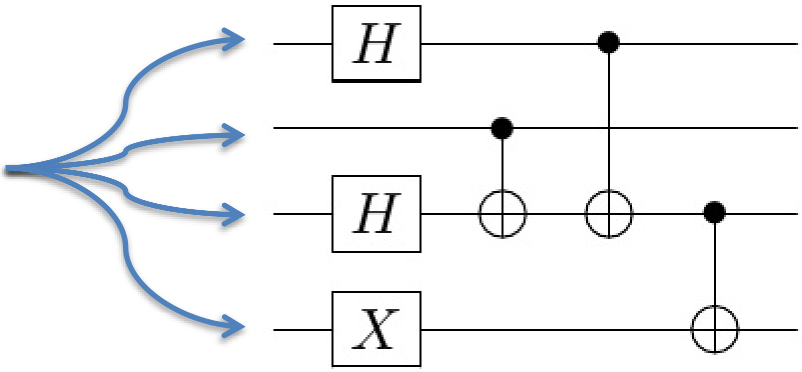
Hadamard gate *Phase gate* *π/8 gate* *CNOT gate*

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Quantum circuits

- The same way
  classical gates can be arranged to form a classical circuit,
  quantum gates can be arranged to form a **quantum circuit**

Unlike classical circuits,
the same number of wires
is going throughout the circuit



...as said before,
inheritence of classical computing –
usually it does not reflect the actual implementation

- Quantum circuit is the most commonly used model
  to describe a **quantum algorithm**

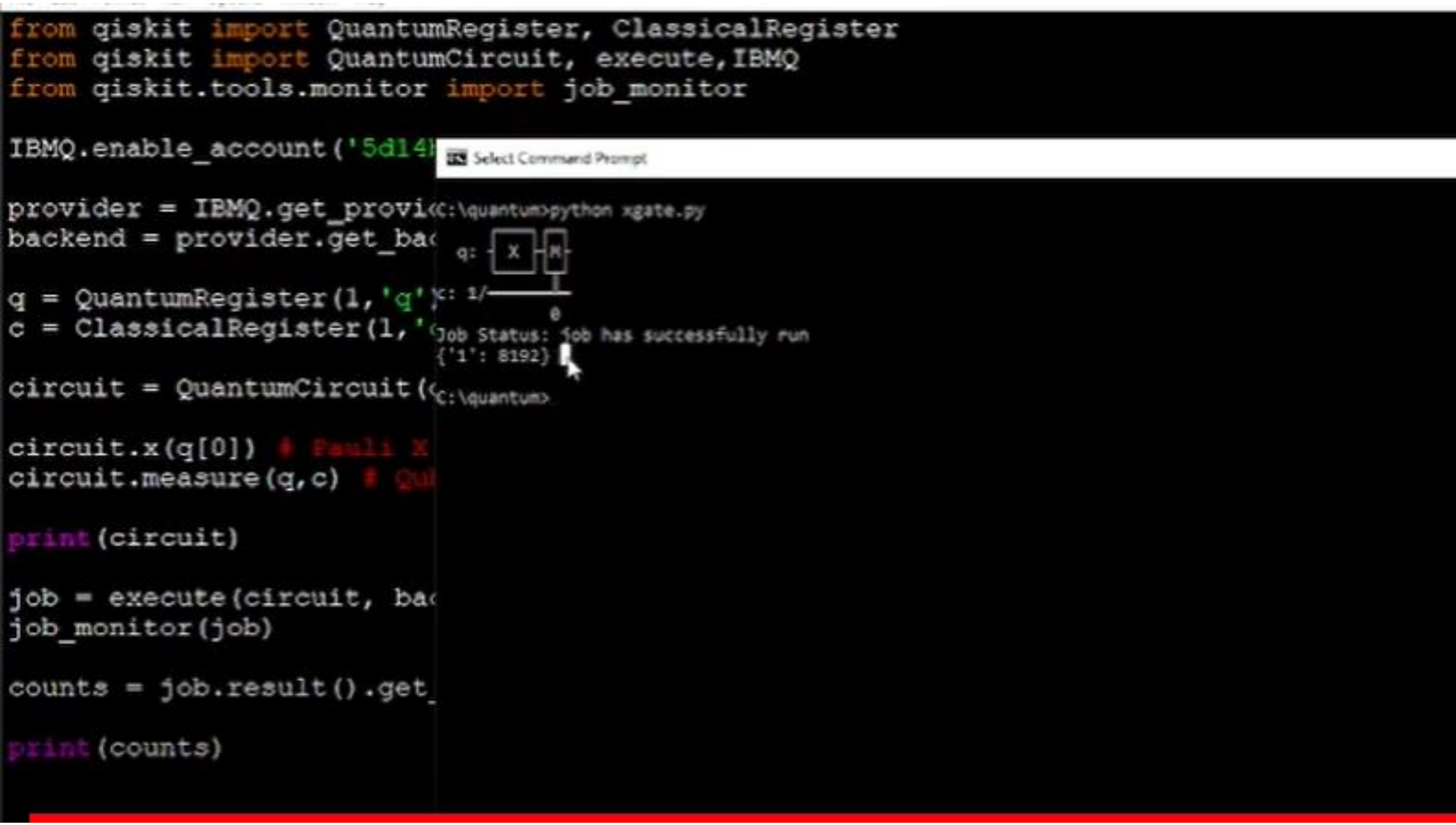
Quantum programming

Quantum programming

- There is already a number of programming languages adapted for quantum computing
  - but there is no actual quantum computer for algorithms to be executed on
- The purpose of quantum programming languages is to provide a tool for researchers, not a tool for programmers
- QCL is an example of such language

Sample code: Quantum Computing with Qiskit: Pauli X gate

```
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute,IBMQ
from qiskit.tools.monitor import job_monitor

IBMQ.enable_account('bc6b8f9a0b493e27e15c496dd09eab4179db0da6f389b492f62e411197f626be9f55e65cad34ebcb3

provider = IBMQ.get_provider(hub='ibm-q')
backend = provider.get_backend('ibmq_qasm_simulator')

q = QuantumRegister(1,'q')
c = ClassicalRegister(1,'c')

circuit = QuantumCircuit(q,c)]

circuit.x(q[0]) # Pauli X Gate
circuit.measure(q,c) # Qubit Measurment

print(circuit)

job = execute(circuit, backend, shots=8192)
job_monitor(job)

counts = job.result().get_counts()

print(counts)
```


from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute,IBMQ
from qiskit.tools.monitor import job_monitor
IBMQ.enable_account('5d14
provider = IBMQ.get_provi
backend = provider.get_ba
q = QuantumRegister(1,'q'
c = ClassicalRegister(1,'
circuit = QuantumCircuit(
circuit.x(q[0]) # Pauli X
circuit.measure(q,c) # Qu
print(circuit)
job = execute(circuit, ba
job_monitor(job)
counts = job.result().get_
print(counts)
Select Command Prompt
C:\quantum>python xgate.py
q:
X
M
c: 1/
0
Job Status: job has successfully run
{'1': 8192}
C:\quantum>

quantum logic gate: NOT gate

$|0\rangle \to |1\rangle \qquad |1\rangle \to |0\rangle$

$\alpha|0\rangle + \beta|1\rangle \to \alpha|1\rangle + \beta|0\rangle$

$$-\boxed{X}-$$

Matrix representation: $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

$$|\psi\rangle\ -\boxed{X}-\boxed{X}- \ \equiv (-)$$

$$\alpha|0\rangle + \beta|1\rangle \to \alpha|1\rangle + \beta|0\rangle$$

$$\to \alpha|0\rangle + \beta|1\rangle$$

$$|\psi\rangle \to X|\psi\rangle \to XX|\psi\rangle$$

$$XX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

<u>Result</u>: Let M be a matrix. Then

$$\| M|\psi\rangle \| = \| |\psi\rangle \|$$

for all $|\psi\rangle$ iff M is unitary.

Hadamard gate:

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) + \beta\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$= \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = H\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$H|1\rangle = H\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

QCL
Language

Quantum programming

- **QCL** (**Q**uantum **C**omputation **L**anguage)

```
/* Remove "//" if starting interpreter with -n option */
// extern operator H(qureg q);

procedure FlipCoin() {
  qureg q[1]; int x;

  reset;
  H(q);
  measure q, x;
  if x == 1  { print "Heads"; }
  if x == 0  { print "Tails"; }
  reset;
}
```

C-like syntax

allows combining of quantum and classical code

http://tph.tuwien.ac.at/~oemer/qcl.html

QCL

- Comes with its own interpreter and quantum system simulator

Start interpreter…

…with a 4 qubit quantum heap (32 if omitted)

Numeric simulator

```
d:\qcl-0.6.3>qcl -b4 -fb
QCL Quantum Computation Language (4 qubits, seed 1404645265)
[0/4] 1 |0>
qcl> include "test1.qcl"  // interpret file contents
qcl> exit;                // quit interpreter

d:\qcl-0.6.3>
```

Shell environment

…there is no assumption about the quantum computer implementation

QCL

- Example of interpreter interactive use

Initial quantum state

Global quantum register definition

Quantum operator

```
[0/4] 1 |0>
qcl> qureg q[1]; // Allocate one qubit from the quantum heap
qcl> H(q);       // Apply Hadamard transform
[1/4] 0.70711 |0> + 0.70711 |1>
qcl> H(q);
[1/4] 1 |0>
qcl> X(q);       // Pauli-X
[1/4] 1 |1>
qcl> H(q);
[1/4] 0.70711 |0> - 0.70711 |1>
qcl> H(q);
[1/4] 1 |1>
qcl>
```

Resulting state

Qubits allocated/Quantum heap total

QCL

- Example of initialization and measurement within interpreter

```
[0/3] 1 |0>
qcl> qureg qh[1];
qcl> H(qh);
[1/3] 0.70711 |0> + 0.70711 |1>
qcl> int x;                // integer to receive measurement result
qcl> measure qh, x;        // measure qh, save result to x
[1/3] 1 |1>
qcl> print x;
: 1
qcl> Y(qh);                // Pauli-Y
[1/3] -i |0>
qcl> reset;                // Reinitialize quantum machine
[1/3] 1 |0>
qcl> qureg qc[2];          // Allocate 2 more qubits
qcl> X(qc[0]);
[3/3] 1 |0,01>
qcl> CNot(qc[1], qc[0]); // CNot(target, control);
[3/3] 1 |0,11>
qcl>
```

Reinitializations have no effect on allocations

QCL

- Examples of quantum registers, expressions and references

```
qureg q[16];                // Allocate 16 qubits
qureg qZero=q[0];           // Create reference to qubit 0
X(qZero);                   // Invert qubit 0
                            // q=|0000 0000 0000 0001>
qureg qLowest4=q[0:3];      // Create reference to qubits 0 to 3
X(qLowest4);                // Invert them
                            // q=|0000 0000 0000 1110>
X(q[12\4]);                 // Invert qubits 12 to 12+3
                            // q=|1111 0000 0000 1110>
X(q[12:15] & qZero);        // Qubit concantenation operator, "&"
                            // q=|0000 0000 0000 1111>
int x = #qLowest4;          // Quantum expression length operator, "#"
                            // x = 4
```

Reference definitions have no effect on quantum heap

QCL

- Example of operator definition

```
operator Toffoli(qureg target, qureg control1, qureg control2) {
  if #target != 1 or #control1 != 1 or #control2 != 1
    { exit "Arguments have to be single qubit quantum registers";
  Matrix8x8(
    1, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 1,
    0, 0, 0, 0, 0, 0, 1, 0,
    target & control1 & control2);
}
```

QCL

– Newly defined operator usage

Force interactive use…

…or interpreter will execute file content and exit

```
d:\qcl-0.6.3>qcl -b3 -fb -i toffoli.qcl
QCL Quantum Computation Language (3 qubits, seed 1404657378)
[0/3] 1 |0>
qcl> qureg q[3];
qcl> X(q);
[3/3] 1 |111>
qcl> Toffoli(q[0], q[1], q[2]);  // Toffoli(target, control1, control2);
[3/3] 1 |110>
qcl> !Toffoli(q[0], q[1], q[2]); // inverse transform operator "!"
[3/3] 1 |111>
qcl>
```

Toffoli gate is its own inverse

QCL allows inverse execution

References

- University of California, Berkeley,
  Qubits and Quantum Measurement and *Entanglement,* lecture notes,
  http://www-inst.eecs.berkeley.edu/~cs191/sp12/
- Michael A. Nielsen, Isaac L. Chuang,
  Quantum Computation and Quantum Information,
  Cambridge University Press, Cambridge, UK, 2010.
- Colin P. Williams, *Explorations in Quantum Computing*, Springer, London, 2011.
- Samuel L. Braunstein, *Quantum Computation Tutorial,* electronic document
  University of York, York, UK
- Bernhard Ömer, *A Procedural Formalism for Quantum Computing*, electronic document, Technical University of Vienna, Vienna, Austria, 1998.
- Artur Ekert, Patrick Hayden, Hitoshi Inamori,
  Basic Concepts in Quantum Computation, electronic document,
  Centre for Quantum Computation, University of Oxford, Oxford, UK, 2008.
- Wikipedia, the free encyclopedia, 2014.