Quantum Programming Language

Samuel Rupprechter

University of Innsbruck

Supervisor: Julian Nagele

Innsbruck, June 2015

Outline

Qubits and Superpositions

Qubits and Superpositions

- Two different states $|0\rangle$ and $|1\rangle$
- $|\psi\rangle = v_0|0\rangle + v_1|1\rangle$

$$v_0|0\rangle + v_1|1\rangle \rightarrow \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}, \text{ so that } |0\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Qubits and Superpositions

Qubits and Superpositions

- Two different states $|0\rangle$ and $|1\rangle$
- $|\psi\rangle = v_0|0\rangle + v_1|1\rangle$

$$v_0|0\rangle + v_1|1\rangle \rightarrow \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}, \text{ so that } |0\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Symmetric superposition ($v_0 = v_1 = \frac{1}{\sqrt{2}}$).

Qubits and Superpositions

Qubits and Superpositions

- Two different states $|0\rangle$ and $|1\rangle$
- $|\psi\rangle = v_0|0\rangle + v_1|1\rangle$

$$v_0|0\rangle + v_1|1\rangle \rightarrow \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}, \text{ so that } |0\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Symmetric superposition ($v_0 = v_1 = \frac{1}{\sqrt{2}}$).
- Built-in randomness!

Parallelism and Entanglement

Parallelism

- Qubits can be in all the states at the same time
- Quantum register containing n quantum bits $\rightarrow$ contains all numbers from 0 to $2^n - 1$ at the same time

Parallelism and Entanglement

Parallelism

- Qubits can be in all the states at the same time
- Quantum register containing n quantum bits $\rightarrow$ contains all numbers from 0 to $2^n - 1$ at the same time

Entanglement

- Two qubits can be entangled
- Works regardless of physical distance
- Basis for communication

Computational Power

Computational Power

- Can solve some problems in NP in polynomial time (prime factorization)

Computational Power

Computational Power

- Can solve some problems in NP in polynomial time (prime factorization)
- It is not proven that quantum computers are “more powerful” than classical computers

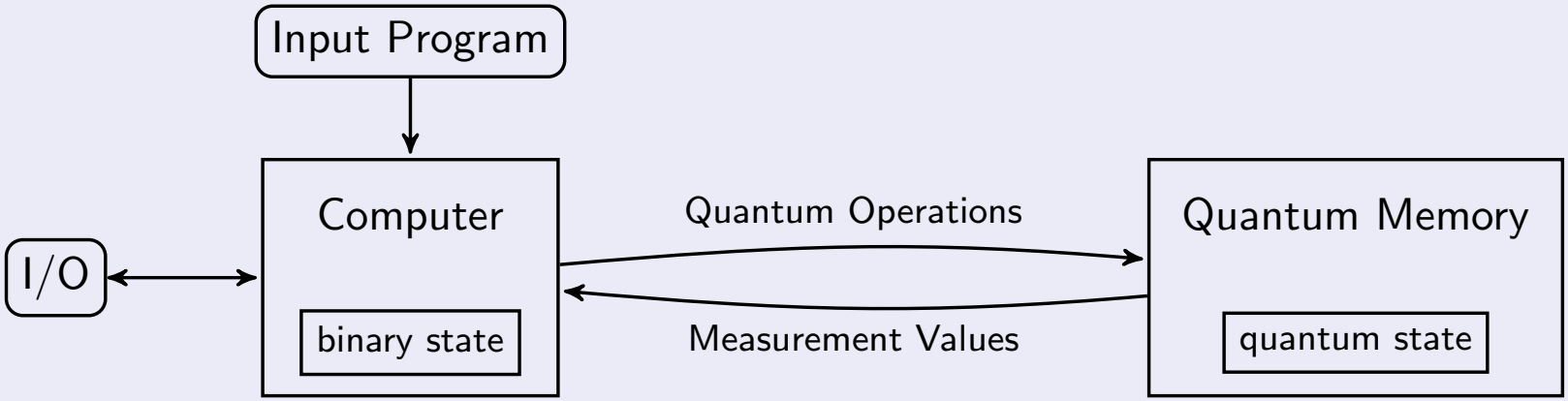
Computational Power

Computational Power

- Can solve some problems in NP in polynomial time (prime factorization)
- It is not proven that quantum computers are “more powerful” than classical computers
- Slower for conventional tasks

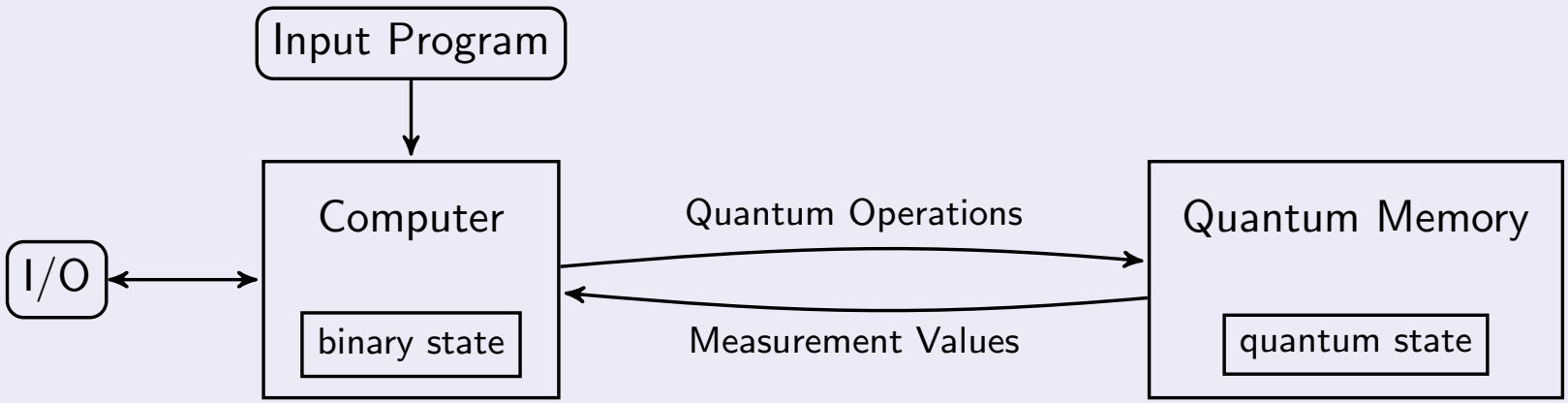
Architecture

Quantum Computer Architecture



Architecture

Quantum Computer Architecture



cQPL

- Compiled to C++
- Run on a quantum simulator

Identifiers and Variables

Data Types

- `bit`
- `int`
- `float`

Variable Declaration

```
new bit b := 0;
new int i := 42;
```

Arithmetic and Logical Expressions

Arithmetic Expressions

- Basic mathematical operators +, -, *, /
- `a + (b - c) * (1/2)`
- Operators "work" on `bits`

Logical Expressions

- Logical operators =, !=, >, <, >=, <=
- `true` has value 0, `false` has value 1

Procedures

Procedures

- Arguments as `name:type` pairs
- Passed by value
- Returns tuple of the classical arguments at the end of its execution
- Called via keyword `call`
- Special built-in procedure `print`

Procedures

Procedures

- Arguments as `name:type` pairs
- Passed by value
- Returns tuple of the classical arguments at the end of its execution
- Called via keyword `call`
- Special built-in procedure `print`

Note

- Tuples can not be assigned to variables!
- Strings can not be assigned to variables!

Procedure Example

Procedure Example

```
proc p: arg1:int, arg2:int {
    arg1 := arg1 + arg2;
    arg2 := 0;
} in {
    new int var1 := 1;
    new int var2 := 2;
    (var1, var2) := call p(var1, var2);
};
```

Control Flow Example

Control Flow

- `while`
- `if-then-else`

Control Flow Example

```
new int i := 0;
new int d := 0;
while (i < 100) do {
    if (d >= 3) then { print i; d := 0; };
    if (d < 3) then { d := d + 1; };
    i := i + 1;
};
```

Quantum Variables

Quantum Variables

- `qbit`, `qint`
- Declaration and assignment stays the same

Quantum Variables

Quantum Variables

- `qbit`, `qint`
- Declaration and assignment stays the same

Quantum Variable Declaration

```
new qbit b := 0;
new qint i := 42;
```

Quantum Variables

Quantum Variables

- `qbit`, `qint`
- Declaration and assignment stays the same
- No cloning!

Quantum Variable Declaration

```
new qbit b := 0;
new qint i := 42;
```

Quantum Gates

Can change the states of qubits

Quantum Gates

Can change the states of qubits

Predefined Gates

- Negation

Quantum Gates

Can change the states of qubits

Predefined Gates

- Negation
- Controlled Negation
  - `CNot` : $|x, y\rangle \rightarrow |x \oplus y, y\rangle$
  - Used to entangle qubits

Quantum Gates

Can change the states of qubits

Predefined Gates

- Negation
- Controlled Negation
  - `CNot` : $|x, y\rangle \rightarrow |x \oplus y, y\rangle$
  - Used to entangle qubits
- Hadamard-Transform
  - Brings a qubit into an even superposition
  - `H` : $|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

Application of Gates

Application of Gates

```
new qbit b1 := 0;
new qbit b2 := 1;
b1 *= Not;
b1 *= H;
b1, b2 *= CNot;
b1 *= [[1, 0, 0i, 1]];
```

Dump

Dump

Displays the probability distribution of quantum variables

Dump

Dump

Displays the probability distribution of quantum variables

Example

```
new qbit q := 0;
print "Before H: "; dump q;
q *= H;
print "After H: "; dump q;
```

Dump

Dump

Displays the probability distribution of quantum variables

Example

```
new qbit q := 0;
print "Before H: "; dump q;
q *= H;
print "After H: "; dump q;
```

Output

```
Before H: 1 |0>
After H: 0.5 |0>, 0.5 |1>
```

Measure

Measure

Measures a quantum variable and will return either $|0\rangle$ or $|1\rangle$

Measure

Measure

Measures a quantum variable and will return either $|0\rangle$ or $|1\rangle$

Application of Gates

```
measure q then { print "q is |0>"; }
else { print "q is |1>"; };
```

Measure

Measure

Measures a quantum variable and will return either $|0\rangle$ or $|1\rangle$

Application of Gates

```
measure q then { print "q is |0>"; }
else { print "q is |1>"; };
```

Output

```
q is |0>
```

or

```
q is |1>
```

Generating Random Numbers

Coin Toss Example

```
new qbit q := 0;
q *= H;
measure q then {
    print "Tossed head";
} else {
    print "Tossed tail";
};
```

Modules

Communication Example

```
module A {
    new qbit a := 0; new qbit b := 0;
    b *= Not; b *= H; a, b *= CNot;
    send b to B;
    measure a then { print "a = |1>"; }
              else { print "a = |0>"; };
};
module B {
    receive b:qbit from A;
    measure b then { print "b = |1>"; }
              else { print "b = |0>"; };
};
```

Conclusion

Conclusion

- cQPL is the only quantum programming language so far that respects physics and offers communication

Conclusion

Conclusion

- cQPL is the only quantum programming language so far that respects physics and offers communication
- Quantum computers will work and might outperform classical computers someday

Conclusion

Conclusion

- cQPL is the only quantum programming language so far that respects physics and offers communication
- Quantum computers will work and might outperform classical computers someday
- Most current encryption will be broken (prime factorization through parallelism)