Introduction to Quantum Programming and Semantics

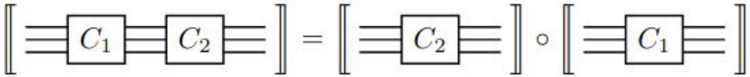
Lecture 3: Composition and tensor product

Chris Heunen

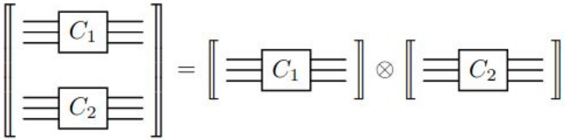
University of Edinburgh

Overview

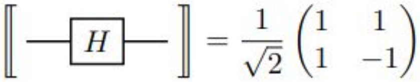
- OpenQASM



- Tensor product



- Deutsch-Jozsa: running example



OpenQASM

OpenQASM

- IBM, but de facto standard

  Braket, Quipper, Tket, many others

- Meant as intermediate representation

  source/target for machines, not humans

- Circuit description

  meant to be executed on classical computer

Features

- Abstraction:
    - Precise location of qubit doesn't matter ("routing")
    - Descriptive names for variables
    - Compile-time constants

- Typing
    - qubit, bit, int, angle
    - registers, e.g. qubit[4], int[16]

- Structure
    - conditionals
    - loops
    - subroutines

Types and gates

- Variables are declared and initialised

```
bit b;
qubit q;
reset q;
```

- Single built-in quantum gate `U`

$$U(a,b,c) = \begin{pmatrix} \cos(a/2) & -e^{ic}\sin(a/2) \\ e^{ib}\sin(a/2) & e^{i(b+c)}\cos(a/2) \end{pmatrix}$$

Subroutines

```
gate X a {
    U ( pi , 0 , pi ) a;
}

gate H a {
    U ( pi/2 , 0 , pi ) a;
}
```

Composition

- Statements separated by semicolons

- Whitespace ignored

- Semantic structure: composition
  - sequential composition of quantum circuits
  - multiplication of unitary matrices
  - sequential composition of string diagrams

Tensor product

Registers of multiple qubits

- Size fixed at declaration
- Interpreted as *tensor product*

single qubit: $|\psi\rangle \in \mathbb{C}^2$

n qubits: $\underbrace{(\mathbb{C}^2) \otimes \cdots \otimes (\mathbb{C}^2)}_{n \text{ times}}$

0 qubits: $(\mathbb{C}^{2^0}) = \mathbb{C}^1 = \mathbb{C}$

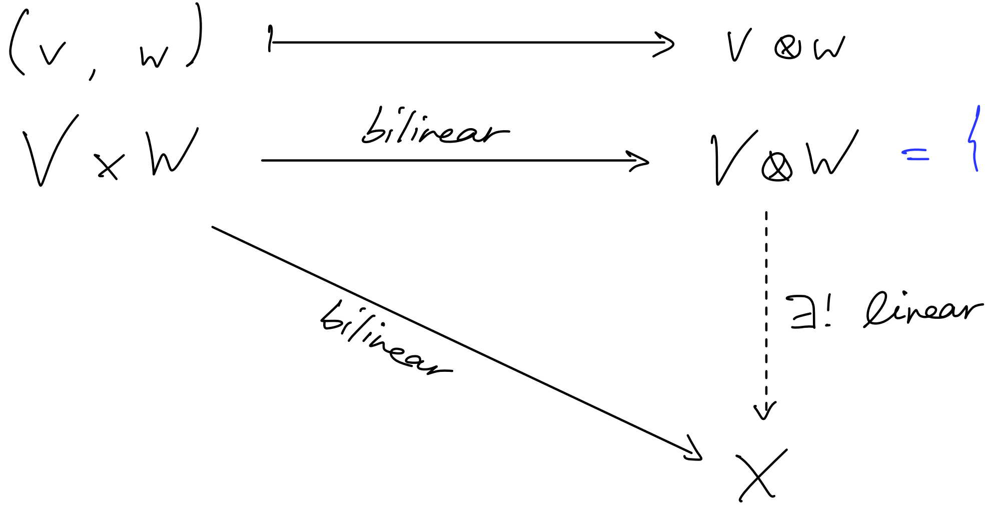
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = f : \mathbb{C}^2 \longrightarrow \mathbb{C}^2$$

$$\begin{pmatrix} e & h \\ k & l \end{pmatrix} = g : \mathbb{C}^2 \longrightarrow \mathbb{C}^2$$

$$f \otimes g : \mathbb{C}^2 \otimes \mathbb{C}^2 \longrightarrow \mathbb{C}^2 \otimes \mathbb{C}^2$$

$$= \begin{pmatrix} ag & bg \\ cg & dg \end{pmatrix} = \begin{pmatrix} ae & ah & be & bh \\ ak & al & bk & bl \\ ce & ch & de & dh \\ ck & cl & dk & dl \end{pmatrix}$$

"Kronecker product"

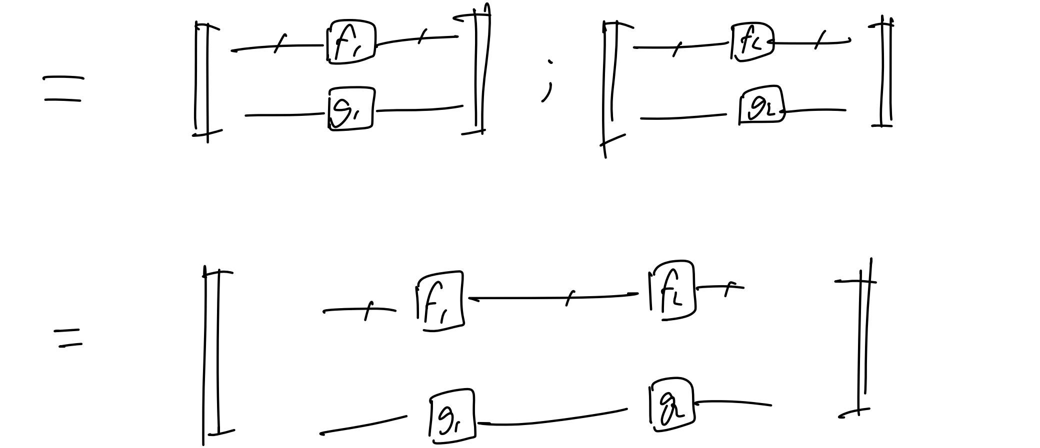


$$V \otimes W = \left\{ \sum_{i=1}^{n} z_i \, v_i \otimes w_i \;\middle|\; z_i \in \mathbb{C},\ v_i \in V,\ w_i \in W \right\}$$

"Tensor products make bilinear maps linear"

Tensor product interacts with composition

- Interchange law



"after"

sequential composition is

associative $h \circ (g \circ f) = (h \circ g) \circ f$

not commutative $g \circ f \neq f \circ g$

parallel composition

associative $h \otimes (g \otimes f) = (h \otimes g) \otimes f$

symmetry $V \otimes W \simeq W \otimes V$

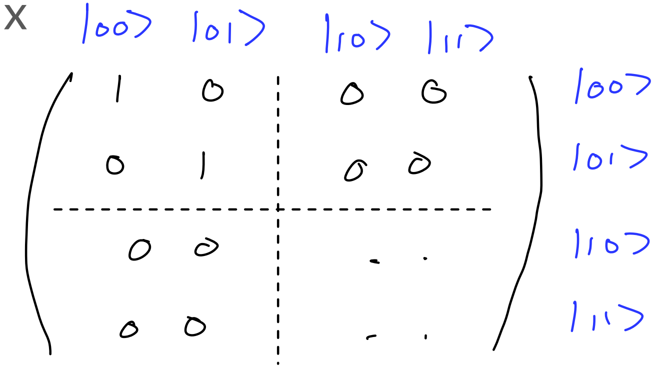
Controlled gates

- Modifier

```
ctrl @ U (a, b, c) q_reg [0], q_reg [1];
```

- Interpreted semantically as matrix

$$1 \oplus U(a, b, c) \quad =$$



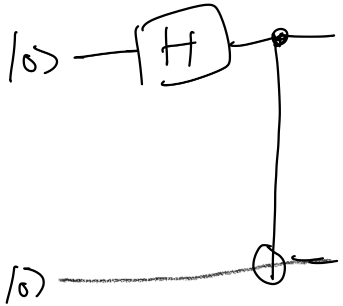
Example

```
gate H a {
   U (pi/2, 0, pi) a;
}

gate CX a, b {
   ctrl @ U (pi, 0, pi) a, b;
}

qubit [2] = q_reg;
reset q_reg;
H q_reg [0];
CX q_reg [0], q_reg [1];
```

=



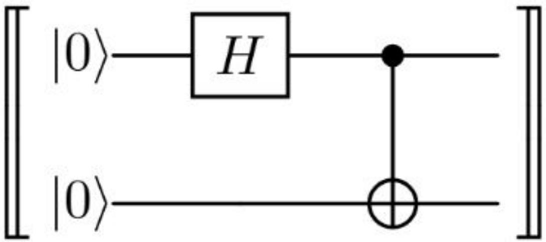
$= \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$

Example

```
gate H a {
    U (pi/2, 0, pi) a;
}

gate CX a, b {
    ctrl @ U (pi, 0, pi) a, b;
}

qubit [2] = q_reg;
reset q_reg;
H q_reg [0];
CX q_reg [0], q_reg [1];
```



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

Deutsch-Jozsa

All-or-nothing oracular promise problems

- Decide on a solution without relying on approximation
- Input is provided as oracle
- Relies on promise about 'global behaviour' of input
- Quantum algorithm faster than best-known classical algorithm
- E.g. Shor, Grover, hidden subgroup

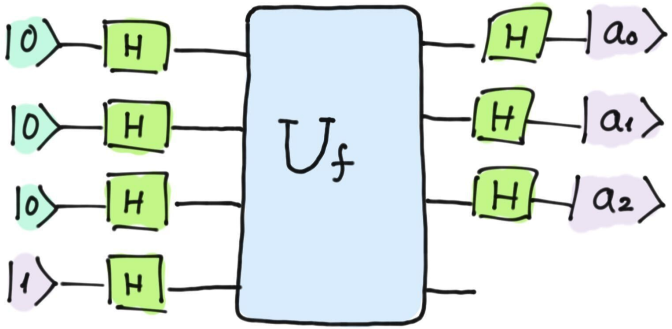
Setting

- Input: 2-colouring $f:\{0,1\}^n \to \{0,1\}$ of bitstrings

- Promise: f is either
  - Constant: any input gets same colour
  - Balanced: 0 on half the inputs, 1 on the other half

- Task: find out which is the case

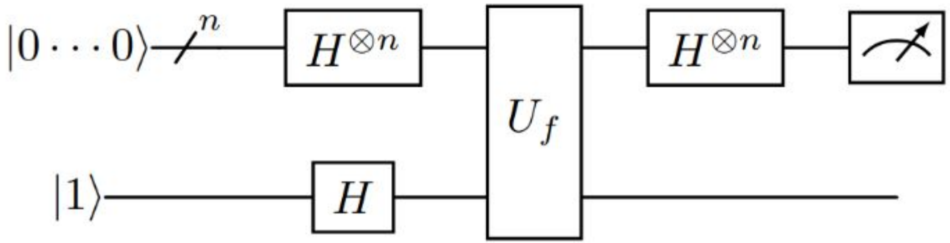
Oracles



- Can ask f for its value on some bitstring
- *Bennett's trick* turns f into unitary gate

$$U_f \left|xy\right\rangle = \left|x\right\rangle \otimes \left|y \text{ XOR } f(x)\right\rangle$$

Deutsch-Jozsa circuit

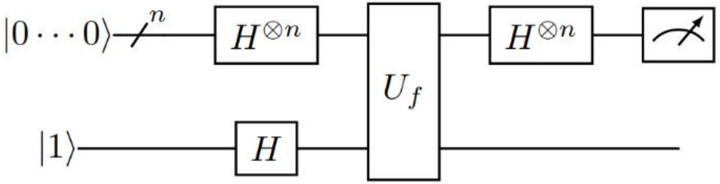


Example oracle

```
gate Oracle x y z {
    CX x z;
    CX y z;
}
```

xyz	x XOR y	z XOR $(x$ XOR $y)$
000	0	0
001	0	1
010	1	1
011	1	0
100	1	1
101	1	0
110	0	0
111	0	1

Implementation



```
// declare three qubits
qreg x;
qreg y;
qreg z;
// set the three qubits to |0>, |0>, and |1>
reset x;
reset y;
reset z;
X z;
// apply Hadamard to all three qubits
H x;
H y;
H z;
// apply the oracle
Oracle x y z;
// apply Hadamard to the first two qubits
H x;
H y;
// measure the first two qubits (will discuss later)
bit a = measure x;
bit b = measure y;
```

Summary:

- OpenQASM low-level de-facto standard circuit description language
- To interpret it, semantics needs both sequential and parallel composition
- Tensor products of matrices
- Deutsch-Jozsa prototypically solves all-or-nothing oracular promise problem