Categories and Quantum Informatics

Week 1: Introduction, Categories

Chris Heunen

THE UNIVERSITY *of* EDINBURGH
informatics

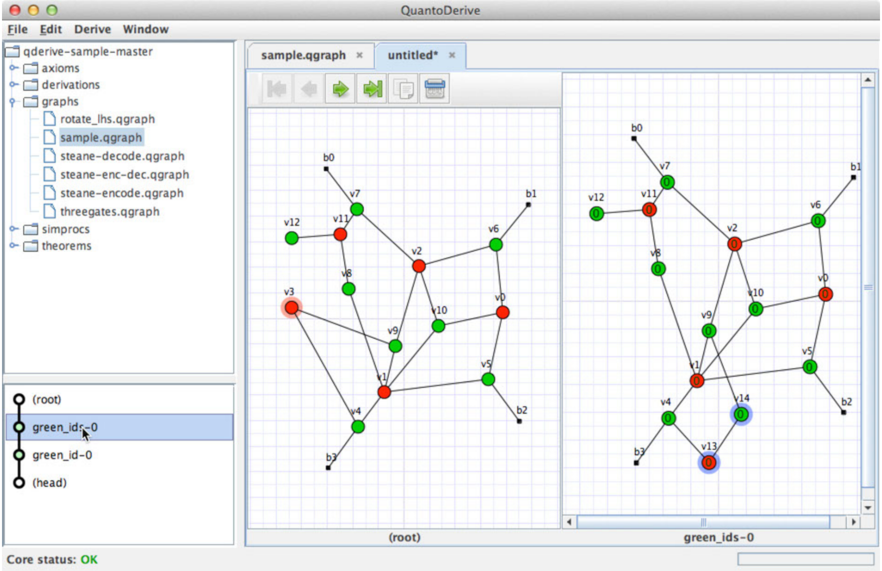
Practicalities

http://www.inf.ed.ac.uk/teaching/courses/cqi

- Lectures: Tuesdays and Thursdays 2-3pm
- Guest lectures:
  -  Andru Gheorghiu: January 23
  -  Pau Enrique Moliner: March 6
  -  Martti Karvonen: March 20
- No lectures: January 25, March 8, March 22.
- Tutorials: Thursday 12-1pm or Friday 2-3pm, weeks 3-9
- Experimental: The Higher Education Academy

Lab

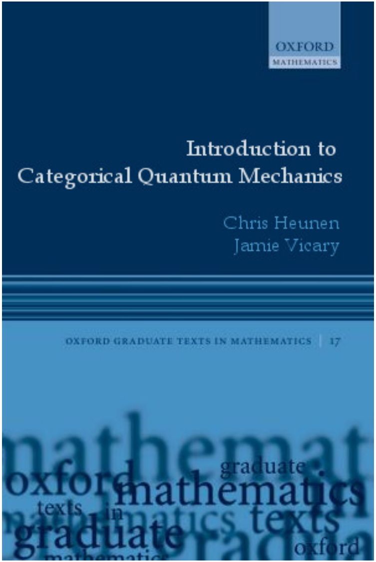
Wednesday March 14 (week 8) 10-11am



Tests

- Tutorials (0%): exercise sheets
- Coursework (30%): week 4
- Written exam (70%): April-May diet

Lecture notes



- Lecture notes on website
- Stripped down version of book
- Please report mistakes and typos

Semantics

Are these two programs the same?

$$P = (\texttt{if 1 = 1 then F else F})$$
$$Q = (\texttt{if 1 = 1 then F else G})$$

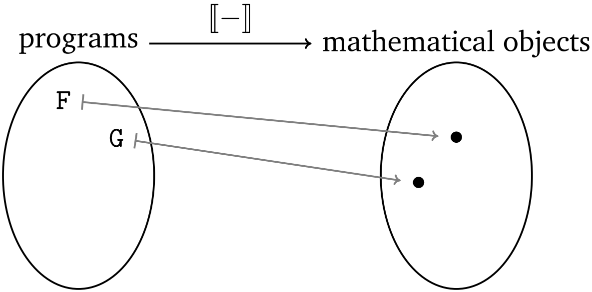
Semantics

Are these two programs the same?

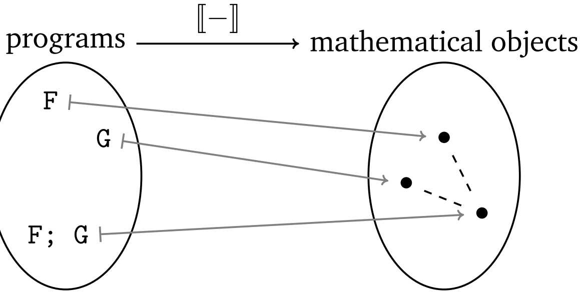
$$P = (\texttt{if 1 = 1 then F else F})$$
$$Q = (\texttt{if 1 = 1 then F else G})$$

- Different syntax
- Different operationally
- But denote same algorithm $[\![P]\!] = [\![Q]\!] = [\![\texttt{F}]\!]$

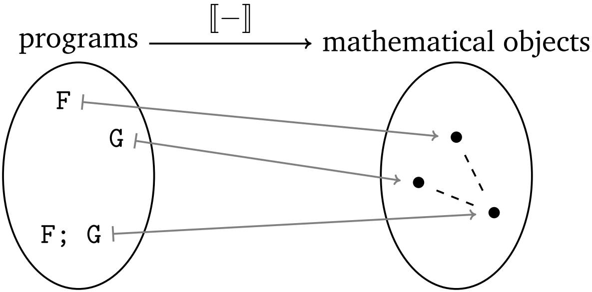
Denotational semantics



Denotational semantics

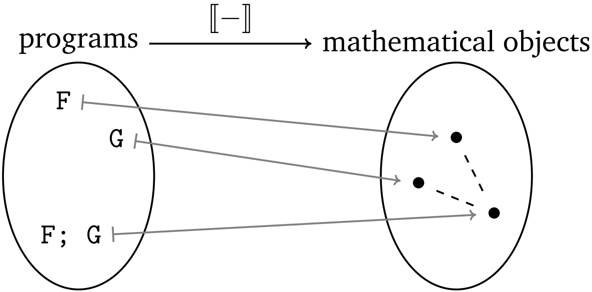


Denotational semantics



- *Operational*: remember implementation details (efficiency)
- *Denotational*: see what program does conceptually (correctness)

Denotational semantics



- *Operational*: remember implementation details (efficiency)
- *Denotational*: see what program does conceptually (correctness)

Motivation:

- Ground programmer's unspoken intuitions
- Justify/refute/suggest program transformations
- Understand programming through mathematics

Quantum technology

intel Newsroom Top News Sections News By Category All News Search Newsroom..

News Byte

January 8, 2018

Share this Article

Contact Intel PR

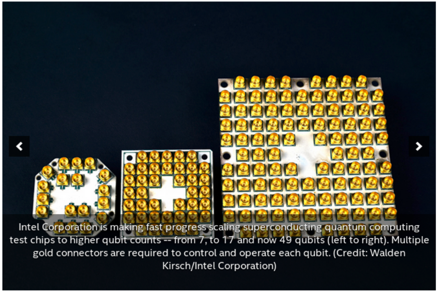
2018 CES: INTEL ADVANCES QUANTUM AND NEUROMORPHIC COMPUTING RESEARCH

Today at the 2018 Consumer Electronics Show in Las Vegas, Intel announced two major milestones in its efforts to research and develop future computing technologies including quantum and neuromorphic computing, which have the potential to help industries, research institutions and society solve problems that currently overwhelm today's classical computers.

During his keynote address, Intel CEO Brian Krzanich announced the successful design, fabrication and delivery of a 49-qubit superconducting quantum test chip. The keynote also noted the promise of neuromorphic computing.

Press Kits: Quantum Computing | 2018 CES

The digitization of nearly everything is creating an explosion of both structured and unstructured data as well as the desire to collect, analyze and act on it. This is driving exponential demand for compute performance and spurring Intel's research into these new, specialized architectures.



Intel Corporation is making fast progress scaling superconducting quantum computing test chips to higher qubit counts -- from 7, to 17 and now 49 qubits (left to right). Multiple gold connectors are required to control and operate each qubit. (Credit: Walden Kirsch/Intel Corporation)

Quantum Computing

The Future is Quantum

November 10, 2017 | Written by: Dario Gil

Categorized: IBM Research | Quantum Computing

Share this post:

Some of the most important technical advances of the 20th century were enabled by decades of fundamental scientific exploration, whose initial purpose was simply to extend human understanding. When Einstein discovered relativity, he had no idea that one day it would be an important part of modern navigation systems. Such is the story of quantum science.

We have come a long way since the earliest days of quantum information theory, when IBM Fellow Charlie Bennett and the other quantum information science pioneers created the foundations that have given rise to a thriving scientific community. Today, this same community has made enough progress that the earliest real systems, which are able to implement theoretical predictions, are being built before our eyes.

For the rest of this incredible story, visit: IBM Research Blog

Dario Gil
Vice President of AI and IBM Q, IBM Research

IBM

Marketplace

IBM Q

IBM Q Network Learn Exp

IBM Q is an industry-first initiative to build commercially available universal quantum computers for business and science.

Watch video (04:46)

Get started

Partner with IBM Q

Learn about the IBM Q Network, a worldwide community of forward-thinking companies, academic institutions, and research labs working with IBM to advance quantum computing.

Try quantum

Explore educational resources, tutorials, and experiment with quantum devices through the IBM Q Experience.

Develop with QISKit

Write and run quantum algorithms on a r computer with QISKit, an open source Py software developer kit.

The future is quantum: Microsoft releases free preview of Quantum Development Kit

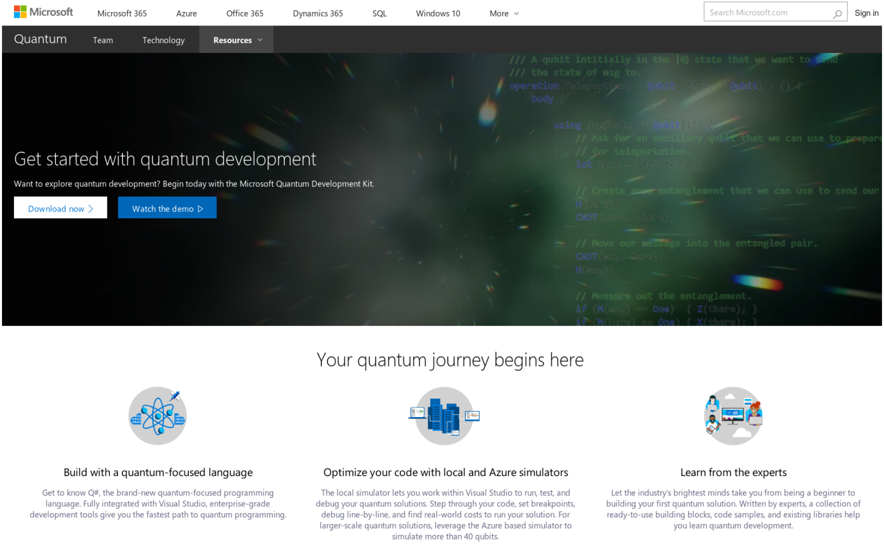
Dec 11, 2017 | Allison Linn

Facebook 172 | Twitter | LinkedIn 328 | Reddit

From left, Charles Marcus, Krysta Svore, Leo Kouwenhoven and Michael Freedman are leading Microsoft's quantum computing efforts. Photo by Brian Smale.

So you want to learn how to program a quantum computer. Now, there's a toolkit for that.

Microsoft is releasing a free preview version of its Quantum Development Kit, which includes the Q# programming language, a quantum computing simulator and other resources for people who want to start writing applications for a quantum computer. The Q# programming language was built from the ground up specifically for quantum computing.


Microsoft
Microsoft 365
Azure
Office 365
Dynamics 365
SQL
Windows 10
More
Search Microsoft.com
Sign in
Quantum
Team
Technology
Resources
Get started with quantum development
Want to explore quantum development? Begin today with the Microsoft Quantum Development Kit.
Download now
Watch the demo
Your quantum journey begins here
Build with a quantum-focused language
Get to know Q#, the brand-new quantum-focused programming language. Fully integrated with Visual Studio, enterprise-grade development tools give you the fastest path to quantum programming.
Optimize your code with local and Azure simulators
The local simulator lets you work within Visual Studio to run, test, and debug your quantum solutions. Step through your code, set breakpoints, debug line-by-line, and find real-world costs to run your solution. For larger-scale quantum solutions, leverage the Azure based simulator to simulate more than 40 qubits.
Learn from the experts
Let the industry's brightest minds take you from being a beginner to building your first quantum solution. Written by experts, a collection of ready-to-use building blocks, code samples, and existing libraries help you learn quantum development.

nature International weekly journal of science

NATURE | NEWS

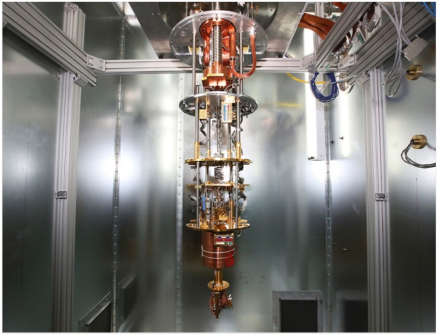
D-Wave upgrade: How scientists are using the world's most controversial quantum computer

Scepticism surrounds the ultimate potential of D-wave machines, but researchers are already finding uses for them.

Elizabeth Gibney

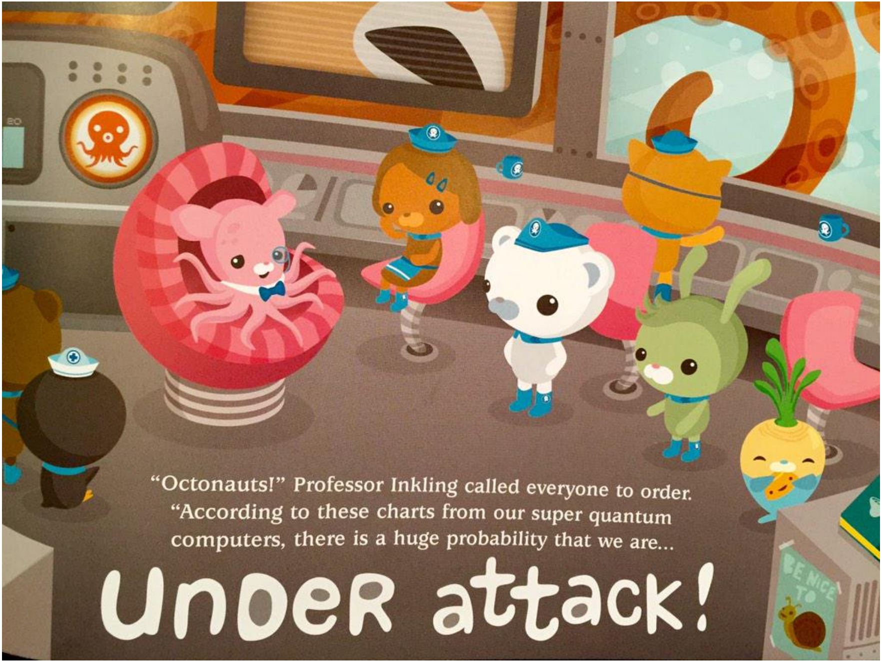
24 January 2017

PDF | Rights & Permissions



Kim Stallknecht/The New York Times/eyevine

D-Wave's latest processor has 2,000 qubits — far surpassing the capacity of previous models.

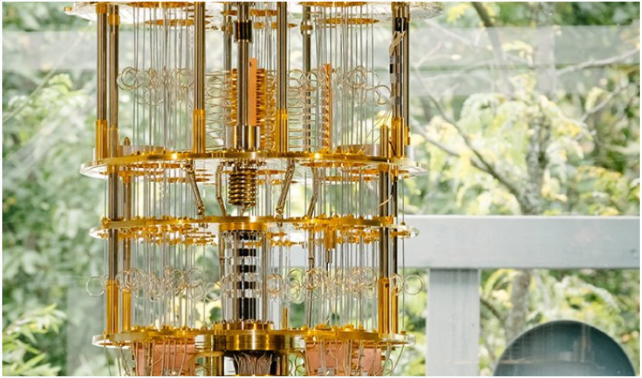


"Octonauts!" Professor Inkling called everyone to order. "According to these charts from our super quantum computers, there is a huge probability that we are...

UNDER attack!

2018 should be the year of quantum supremacy

brian wang | December 26, 2017 5 comments



Tweet Like 69 Share 69 Save 1

IBM, Dwave Systems, Google, Rigetti, Intel and others are computing to develop faster quantum computing systems.

In November, 2017, IBM announced a 50 qubit prototype quantum computer chip.

IBM, Google and Rigetti are working on approximate gate model systems.

Rigetti has a 19 qubit chip.

Programming quantum computers

- What if P, Q executables instead of source code? Black box.
  But can still analyse *information flow*
- Empirical method: know how quantum theory works, but why?
- Cannot copy or delete, how to handle recursion?

Programming quantum computers

- What if P, Q executables instead of source code? Black box.
  But can still analyse *information flow*
- Empirical method: know how quantum theory works, but why?
- Cannot copy or delete, how to handle recursion?

- Investigate semantics to design good programming language
- "Semantics = programming language"

NEWS & TECHNOLOGY 4 January 2017

Physicists can't agree on what the quantum world looks like

Got the maths, not the meaning
Dereje Belachew/Alamy Stock Photo

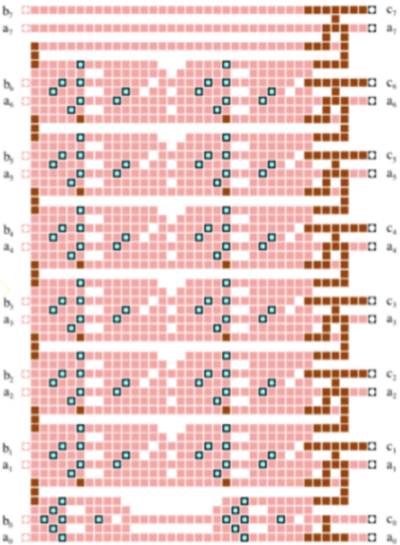
By **Sophia Chen**

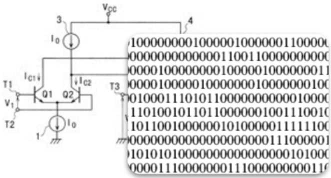
IF YOU find the quantum world confusing you're not alone. A recent survey shows that physicists disagree over the picture of reality that quantum mechanics describes – and that many of them don't even care.

There was no consensus among the 149 survey participants. While 39 per cent supported the so-called Copenhagen interpretation, the conventional picture of quantum mechanics, 25 per cent supported alternatives and 36 per cent had no preference at all. In addition, many weren't sure they understood what certain interpretations described.

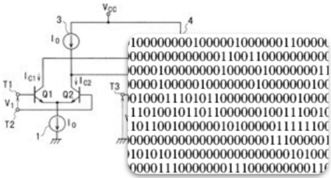
Need for abstraction

8-bit adder, dimension $\sim 2^{1764}$



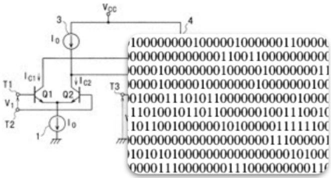


V_{CC}
3
4
I_0
I_{C1}
I_{C2}
T1
Q1
Q2
V_1
T2
T3
1



$$\lambda x.\lambda y.\lambda z.xz(yz)$$

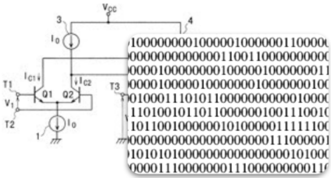
```
fun fact 0 = 1
  | fact x = x * fact (x-1)
```



$$\lambda x.\lambda y.\lambda z.xz(yz)$$

```
fun fact 0 = 1
  | fact x = x * fact (x-1)
```

Hilbert space,
unitary transforms,
self-adjoint
operators....



$$\lambda x.\lambda y.\lambda z.xz(yz)$$

```
fun fact 0 = 1
  | fact x = x * fact (x-1)
```

Hilbert space, unitary transforms, self-adjoint operators....

?

Categorical semantics

Want:

- Compositionality: $[\![\texttt{F; G}]\!] = [\![\texttt{G}]\!] \circ [\![\texttt{F}]\!]$
- Concurrency: $[\![\texttt{F while G}]\!] = [\![\texttt{F}]\!] \otimes [\![\texttt{G}]\!]$
- Recursion: $[\![\texttt{F(X)}]\!] = [\![\texttt{F}]\!]([\![\texttt{X}]\!])$

Categorical semantics

Want:

- Compositionality: $[\![\mathtt{F;\ G}]\!] = [\![\mathtt{G}]\!] \circ [\![\mathtt{F}]\!]$
- Concurrency: $[\![\mathtt{F\ while\ G}]\!] = [\![\mathtt{F}]\!] \otimes [\![\mathtt{G}]\!]$
- Recursion: $[\![\mathtt{F(X)}]\!] = [\![\mathtt{F}]\!]([\![\mathtt{X}]\!])$

Where can $[\![\mathtt{F}]\!]$ live?

- λ-calculus
- partially ordered sets
- categories

Categorical semantics

Want:

- Compositionality: $[\![\texttt{F; G}]\!] = [\![\texttt{G}]\!] \circ [\![\texttt{F}]\!]$
- Concurrency: $[\![\texttt{F while G}]\!] = [\![\texttt{F}]\!] \otimes [\![\texttt{G}]\!]$
- Recursion: $[\![\texttt{F(X)}]\!] = [\![\texttt{F}]\!]([\![\texttt{X}]\!])$

Where can $[\![\texttt{F}]\!]$ live?

- λ-calculus
- partially ordered sets
- categories

Instantiate in different categories:

- Isolate differences between quantum and classical behaviour
- Apply quantum thinking to other settings

Categories

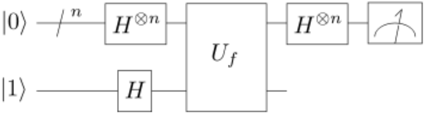
Category theory is a way of thinking more than deep theorems

> *"The essential virtue of category theory is as a discipline for making definitions, the programmers main task in life."*
>
> *– D. E. Rydeheard*

> *"Good general theory does not search for the maximum generality, but for the right generality."*
>
> *– S. Mac Lane*

Monoidal categories

Added benefit: graphical calculus



Correctness proof:

The algorithm begins with the n+1 bit state $|0\rangle^{\otimes n}|1\rangle$. That is, the first n bits are each in the state $|0\rangle$ and the final bit is $|1\rangle$. A Hadamard transformation is applied to each bit to obtain the state

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|0\rangle-|1\rangle).$$

We have the function f implemented as quantum oracle. The oracle maps the state $|x\rangle|y\rangle$ to $|x\rangle|y\oplus f(x)\rangle$, where $\oplus$ is addition modulo 2 (see below for details of implementation). Applying the quantum oracle gives

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|f(x)\rangle-|1\oplus f(x)\rangle).$$

For each x, $f(x)$ is either 0 or 1. A quick check of these two possibilities yields

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle(|0\rangle-|1\rangle).$$

At this point the last qubit may be ignored. We apply a Hadamard transformation to each qubit to obtain

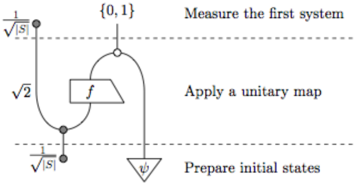
$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle=\frac{1}{2^n}\sum_{y=0}^{2^n-1}\left[\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right]|y\rangle$$

where $x\cdot y=x_0y_0\oplus x_1y_1\oplus\cdots\oplus x_{n-1}y_{n-1}$ is the sum of the bitwise product.
Finally we examine the probability of measuring $|0\rangle^{\otimes n}$,

$$\left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\right|^2$$

which evaluates to 1 if $f(x)$ is constant (constructive interference) and 0 if $f(x)$ is balanced (destructive interference).

vs



Categories

Categories consist of

- objects $A, B, C, \ldots$
- morphisms $A \xrightarrow{f} B$ going between objects

Categories

Categories consist of

- objects $A, B, C, \ldots$
- morphisms $A \xrightarrow{f} B$ going between objects

Examples:

- physical systems, physical processes governing them
- data types, algorithms manipulating them
- algebraic/geometric structures, structure-preserving functions
- logical propositions, implications between them

Categories

Categories consist of

- objects $A, B, C, \ldots$
- morphisms $A \xrightarrow{f} B$ going between objects

Examples:

- physical systems, physical processes governing them
- data types, algorithms manipulating them
- algebraic/geometric structures, structure-preserving functions
- logical propositions, implications between them

Ignore all structure of objects, focus relationships between objects
"Morphisms are more important than objects"

Categories

A category $\mathbf{C}$ consists of the following data:

- a collection $\mathrm{Ob}(\mathbf{C})$ of objects
- for every pair of objects A and B, a collection $\mathbf{C}(A,B)$ of morphisms, with $f \in \mathbf{C}(A,B)$ written $A \xrightarrow{f} B$
- for all morphisms $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ a composite $A \xrightarrow{g \circ f} C$
- for every object A an identity morphism $A \xrightarrow{\mathrm{id}_A} A$

such that:

- associativity: $h \circ (g \circ f) = (h \circ g) \circ f$
- identity: $\mathrm{id}_B \circ f = f = f \circ \mathrm{id}_A$

Sets and functions

The category **Set** of sets and functions:

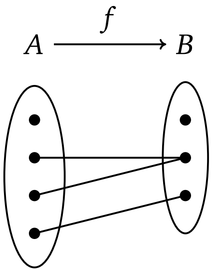
- *objects* are sets $A, B, C, \ldots$
- *morphisms* are functions $f, g, h, \ldots$
- *composition* of $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ is the function $g \circ f : a \mapsto g(f(a))$
- *the identity morphism* on A is the function $\mathrm{id}_A : a \mapsto a$

Sets and functions

The category **Set** of sets and functions:

- *objects* are sets $A, B, C, \ldots$
- *morphisms* are functions $f, g, h, \ldots$
- *composition* of $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ is the function $g \circ f : a \mapsto g(f(a))$
- *the identity morphism* on A is the function $\mathrm{id}_A : a \mapsto a$

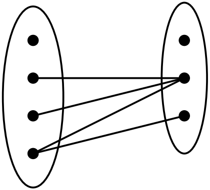
Think of a function $A \xrightarrow{f} B$ dynamically, as indicating how elements of A can evolve into elements of B



Relations

Given sets A and B, a relation $A \xrightarrow{R} B$ is a subset $R \subseteq A \times B$.

$$A \xrightarrow{R} B$$

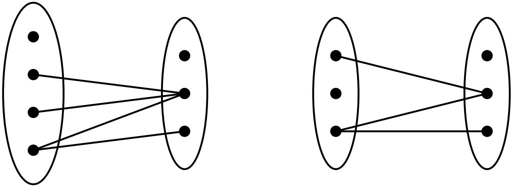


Nondeterministic: an element of A can relate to more than one element of B, or to none.

Composition of relations

Suppose we have a pair of head-to-tail relations:

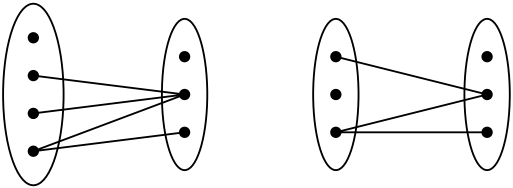
$$A \xrightarrow{R} B \qquad B \xrightarrow{S} C$$



Composition of relations

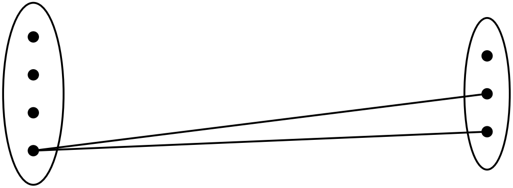
Suppose we have a pair of head-to-tail relations:

$$A \xrightarrow{R} B \qquad\qquad B \xrightarrow{S} C$$



Then our interpretation gives a natural notion of composition:

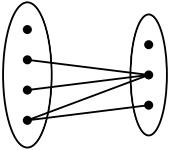
$$A \xrightarrow{S \circ R} C$$



Relations as matrices

We can write relations as (0,1)-valued matrices:

$$A \xrightarrow{R} B$$



$$\leftrightsquigarrow \quad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Composition of relations is then ordinary matrix multiplication, with logical disjunction (OR) and conjunction (AND) for $+$ and $\times$.

Sets and relations

The category **Rel** of sets and relations:

- *objects* are sets $A, B, C, \ldots$;
- *morphisms* are relations $R \subseteq A \times B$, with $(a, b) \in R$ written aRb;
- *composition* $A \xrightarrow{R} B \xrightarrow{S} C$ is $\{(a, c) \in A \times C \mid \exists b \in B \colon aRb, bSc\}$;
- *the identity morphism* on A is $\{(a, a) \in A \times A \mid a \in A\}$.

Sets and relations

The category **Rel** of sets and relations:

- *objects* are sets $A, B, C, \ldots$;
- *morphisms* are relations $R \subseteq A \times B$, with $(a,b) \in R$ written aRb;
- *composition* $A \xrightarrow{R} B \xrightarrow{S} C$ is $\{(a,c) \in A \times C \mid \exists b \in B \colon aRb, bSc\}$;
- *the identity morphism* on A is $\{(a,a) \in A \times A \mid a \in A\}$.

It seems like **Rel** should be a lot like **Set**,
but we will discover it behaves a lot more like **Hilb**.

While **Set** is a setting for classical physics,
and **Hilb** is a setting for quantum physics,
Rel is somewhere in the middle.

Diagrams

Helps to draw diagrams, indicating how morphisms compose

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[r, "g"] \arrow[d, "i"'] & C \\
D \arrow[r, "k"'] & E \arrow[ur, "j"'] &
\end{tikzcd}
$$

Diagram commutes if every path from object to another is equal

Two ways to speak about equality of composite morphisms:
algebraic equations, and commuting diagrams.

Terminology

For morphism $A \xrightarrow{f} B$

- A is its domain
- B is its codomain
- f is endomorphism if $A = B$
- f is isomorphism if $f^{-1} \circ f = \mathrm{id}_A, f \circ f^{-1} = \mathrm{id}_B$ for some $B \xrightarrow{f^{-1}} A$
- A and B are isomorphic ($A \simeq B$) if there is isomorphism $A \to B$

Terminology

For morphism $A \xrightarrow{f} B$

- A is its domain
- B is its codomain
- f is endomorphism if $A = B$
- f is isomorphism if $f^{-1} \circ f = \mathrm{id}_A, f \circ f^{-1} = \mathrm{id}_B$ for some $B \xrightarrow{f^{-1}} A$
- A and B are isomorphic ($A \simeq B$) if there is isomorphism $A \to B$

If a morphism has an inverse, it is unique:

$$g = g \circ \mathrm{id} = g \circ (f \circ g') = (g \circ f) \circ g' = \mathrm{id} \circ g' = g'$$

Terminology

For morphism $A \xrightarrow{f} B$

- A is its domain
- B is its codomain
- f is endomorphism if $A = B$
- f is isomorphism if $f^{-1} \circ f = \mathrm{id}_A, f \circ f^{-1} = \mathrm{id}_B$ for some $B \xrightarrow{f^{-1}} A$
- A and B are isomorphic ($A \simeq B$) if there is isomorphism $A \to B$

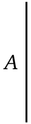
If a morphism has an inverse, it is unique:

$$g = g \circ \mathrm{id} = g \circ (f \circ g') = (g \circ f) \circ g' = \mathrm{id} \circ g' = g'$$

A groupoid is a category where every morphism is an isomorphism

Graphical notation

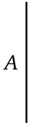
Draw object A as:



It's just a line. Think of it as a picture of the identity morphism $A \xrightarrow{\mathrm{id}_A} A$. Remember: morphisms are more important than objects.

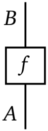
Graphical notation

Draw object A as:



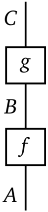
It's just a line. Think of it as a picture of the identity morphism $A \xrightarrow{\mathrm{id}_A} A$. Remember: morphisms are more important than objects.

Draw morphism $A \xrightarrow{f} B$ as:



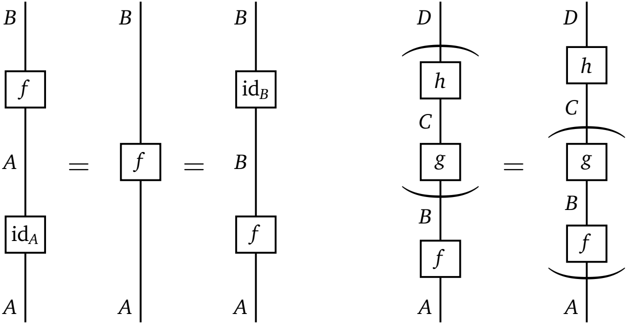
Graphical notation

Draw composition of $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ as:



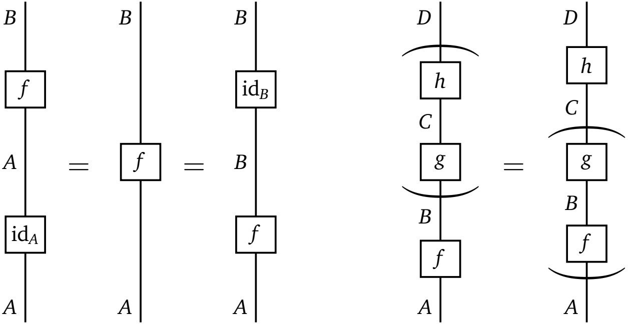
Graphical notation

Identity law and associativity law become:



Graphical notation

Identity law and associativity law become:



This *one-dimensional* representation is familiar; we usually draw it horizontally, and call it algebra. The graphical calculus 'absorbs' the axioms of a category.

Functors

Morphisms are more important than objects: what about categories themselves? Given categories $\mathbf{C}$ and $\mathbf{D}$, a functor $F\colon \mathbf{C} \to \mathbf{D}$ is:

- for each object $A \in \mathrm{Ob}(\mathbf{C})$, an object $F(A) \in \mathrm{Ob}(\mathbf{D})$
- for each morphism $A \xrightarrow{f} B$ in $\mathbf{C}$, a morphism $F(A) \xrightarrow{F(f)} F(B)$ in $\mathbf{D}$

such that structure is preserved:

- $F(g \circ f) = F(g) \circ F(f)$ for morphisms $A \xrightarrow{f} B \xrightarrow{g} C$ in $\mathbf{C}$
- $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$ for objects A in $\mathbf{C}$

Functors

Morphisms are more important than objects: what about categories themselves? Given categories $\mathbf{C}$ and $\mathbf{D}$, a functor $F \colon \mathbf{C} \to \mathbf{D}$ is:

- for each object $A \in \mathrm{Ob}(\mathbf{C})$, an object $F(A) \in \mathrm{Ob}(\mathbf{D})$
- for each morphism $A \xrightarrow{f} B$ in $\mathbf{C}$, a morphism $F(A) \xrightarrow{F(f)} F(B)$ in $\mathbf{D}$

such that structure is preserved:

- $F(g \circ f) = F(g) \circ F(f)$ for morphisms $A \xrightarrow{f} B \xrightarrow{g} C$ in $\mathbf{C}$
- $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$ for objects A in $\mathbf{C}$

It is:

- full when $f \mapsto F(f)$ are surjections $\mathbf{C}(A, B) \to \mathbf{D}(F(A), F(B))$
- faithful when $f \mapsto F(f)$ are injections $\mathbf{C}(A, B) \to \mathbf{D}(F(A), F(B))$
- essentially surjective on objects each $B \in \mathrm{Ob}(\mathbf{D})$ is isomorphic to $F(A)$ for some $A \in \mathrm{Ob}(\mathbf{C})$
- equivalence when full, faithful, essentially surjective on objects

Natural transformations

Given functors $F, G \colon \mathbf{C} \to \mathbf{D}$, a natural transformation $\zeta \colon F \Longrightarrow G$ assigns to every object A in $\mathbf{C}$ of a morphism $F(A) \xrightarrow{\zeta_A} G(A)$ in $\mathbf{D}$, such that for every morphism $A \xrightarrow{f} B$ in $\mathbf{C}$:

$$\begin{tikzcd} F(A) \arrow[r, "\zeta_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\ F(B) \arrow[r, "\zeta_B"'] & G(B) \end{tikzcd}$$

If every component ζ_A is an isomorphism then ζ is called a *natural isomorphism*, and F and G are said to be *naturally isomorphic*.

Natural transformations

Given functors $F, G\colon \mathbf{C} \to \mathbf{D}$, a natural transformation $\zeta\colon F \Longrightarrow G$ assigns to every object A in $\mathbf{C}$ of a morphism $F(A) \xrightarrow{\zeta_A} G(A)$ in $\mathbf{D}$, such that for every morphism $A \xrightarrow{f} B$ in $\mathbf{C}$:

$$\begin{tikzcd} F(A) \arrow[r, "\zeta_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\ F(B) \arrow[r, "\zeta_B"'] & G(B) \end{tikzcd}$$

If every component ζ_A is an isomorphism then ζ is called a *natural isomorphism*, and F and G are said to be *naturally isomorphic*.

A functor $F\colon \mathbf{C} \to \mathbf{D}$ is an equivalence if and only if there is a functor $G\colon \mathbf{D} \to \mathbf{C}$ and natural isomorphisms $G \circ F \simeq \mathrm{id}_{\mathbf{C}}$ and $F \circ G \simeq \mathrm{id}_{\mathbf{D}}$.

Products

Given objects A and B, a product is:

- an object $A \times B$
- morphisms $A \times B \xrightarrow{p_A} A$ and $A \times B \xrightarrow{p_B} B$

such that any two morphisms $X \xrightarrow{f} A$ and $X \xrightarrow{g} B$ allow a unique morphism $\begin{pmatrix} f \\ g \end{pmatrix} : X \longrightarrow A \times B$ with $p_A \circ \begin{pmatrix} f \\ g \end{pmatrix} = f$ and $p_B \circ \begin{pmatrix} f \\ g \end{pmatrix} = g$

$$\begin{tikzcd}
 & X \arrow[dl, "f"'] \arrow[d, dashed, "\left(\begin{smallmatrix} f \\ g \end{smallmatrix}\right)"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "p_A"] \arrow[r, "p_B"'] & B
\end{tikzcd}$$

Universal property: $A \times B$ is universal way to put A and B together

Summary

- Denotational semantics: structure behind computation
- Categories: objects and (more importantly) morphisms
- Examples: sets and functions, sets and relations
- Isomorphic objects: behave the same
- Functors: ‘morphisms between categories’
- Equivalent categories: behave the same
- Products: combine objects universally