

Computational models and M_0

Opening: what a computational model specifies. Then reversible deterministic computation.

States, rules, and outputs

A computational model specifies its states, allowed operations, and readout.

Our progression begins with reversible classical states, adds probability and amplitudes, and ends with the complete quantum circuit model and Qiskit.

The Turing Machine (Turing, 1936)*

Turing models computation as:

- a machine with **finite “m-configurations” (states)**
- a **tape divided into squares**
- a **scanned square** (what the machine “sees”)
- a finite set of **symbols**

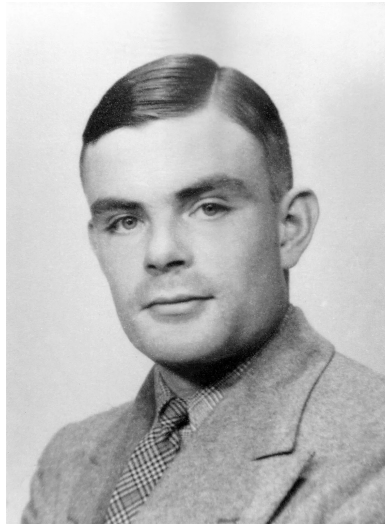
Operation

At each step, behaviour is determined by:

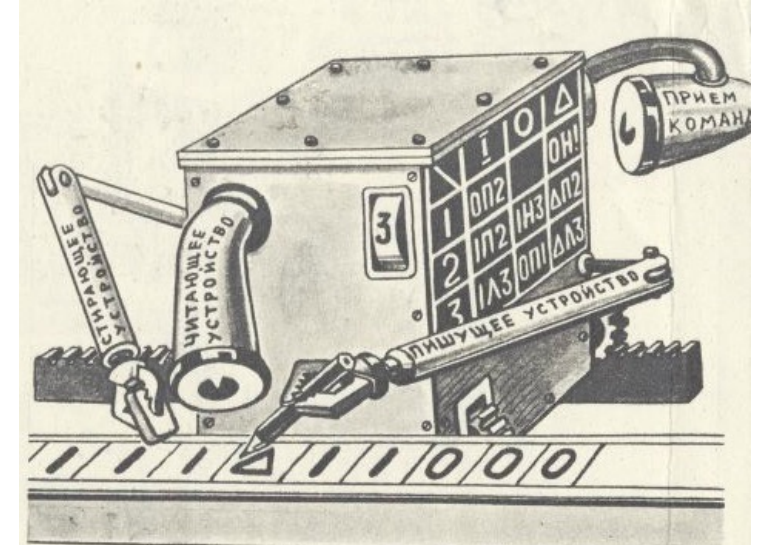
the **configuration** = (state, scanned symbol)

Then the machine may:

- write or erase a symbol
- move left or right
- change its internal state



Alan Turing
graduate student **King's College,**
Cambridge



“We may compare a man in the process of computing... to a machine...”

“The possible behaviour of the machine... is determined by the configuration.” - Turing

*In many ways, you understand this better than most people in 1936, because you already think in terms of step-by-step procedural programs.

The Universal Turing Machine (Turing, 1936)

Computation = machine

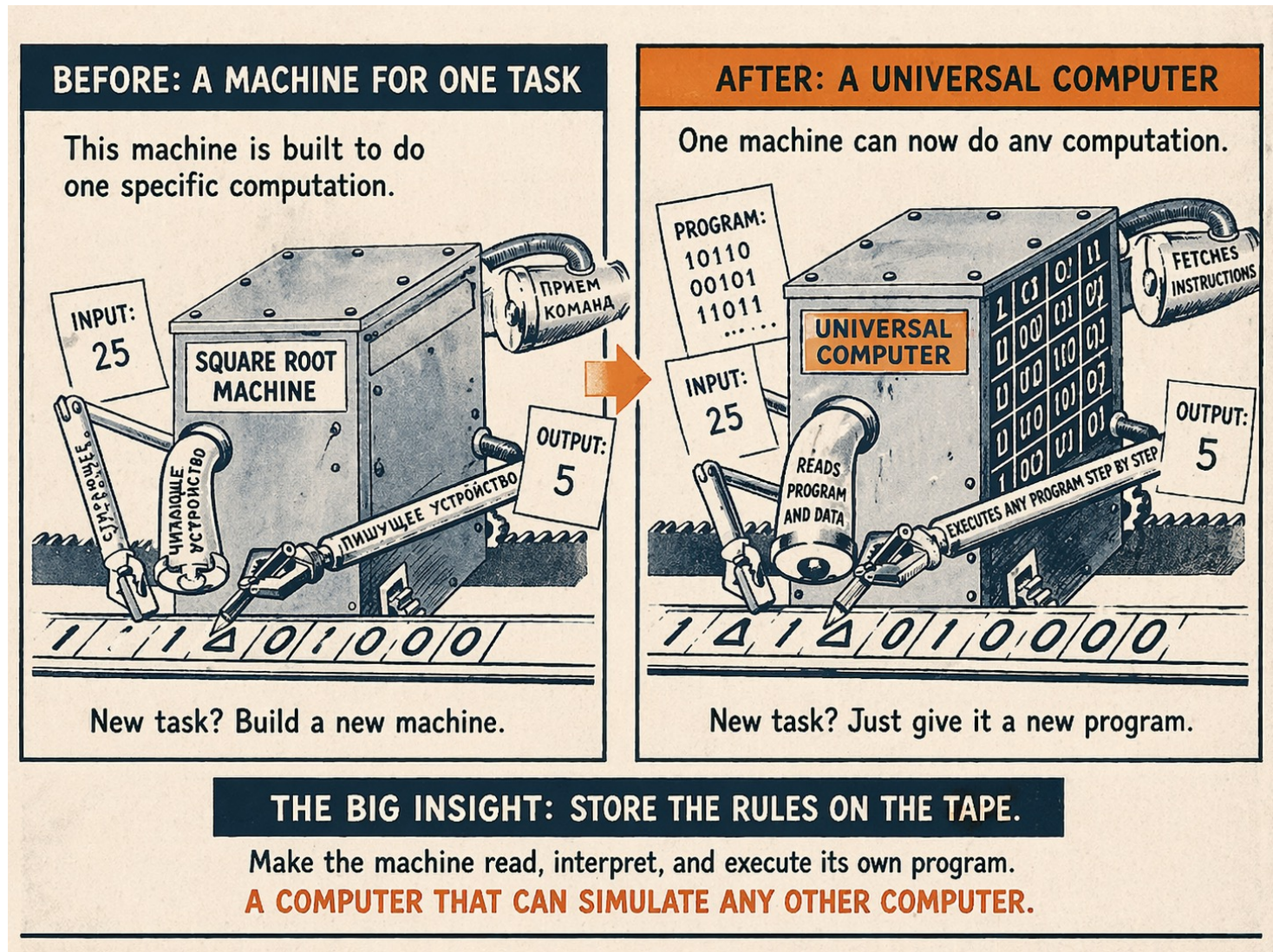
- one machine
- one task
- change the task → build a new machine

Computation = program

one machine
many tasks
change the task → change the program

This gives us two things

- 1) A definition of what we mean to compute
- 2) Motivation to build computers rather than single task machines.

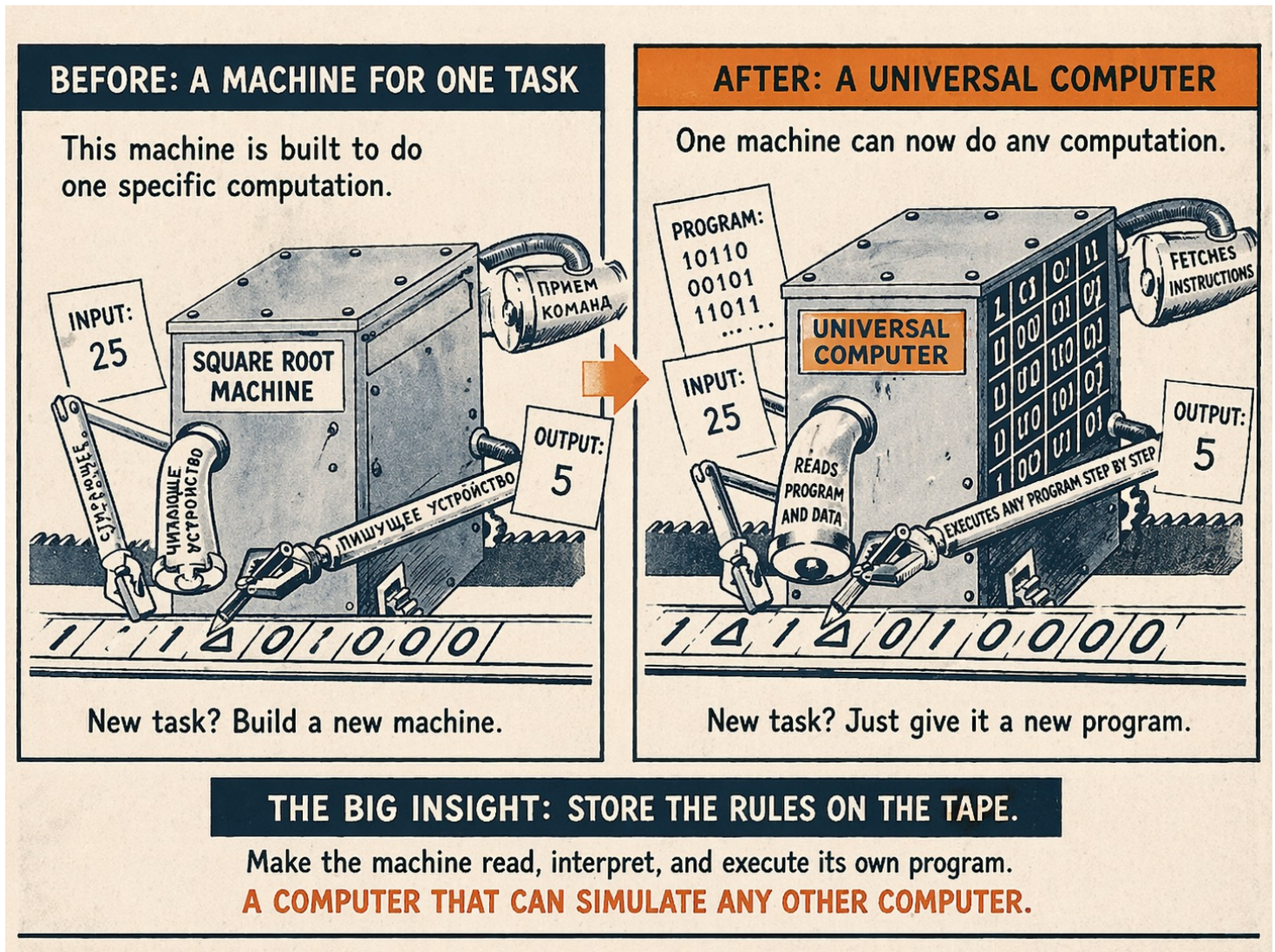


The Universal Turing Machine (Turing, 1936)

This played the same role in computer science as the atom in physics, the cell in biology, and DNA in genetics.

It defined the fundamental unit of computation.

It allowed us to describe and classify computations rigorously.



The Universal Turing Machine (Turing, 1936)

CS500 – Theory of Computation

Once we know what can be computed, the next question is:

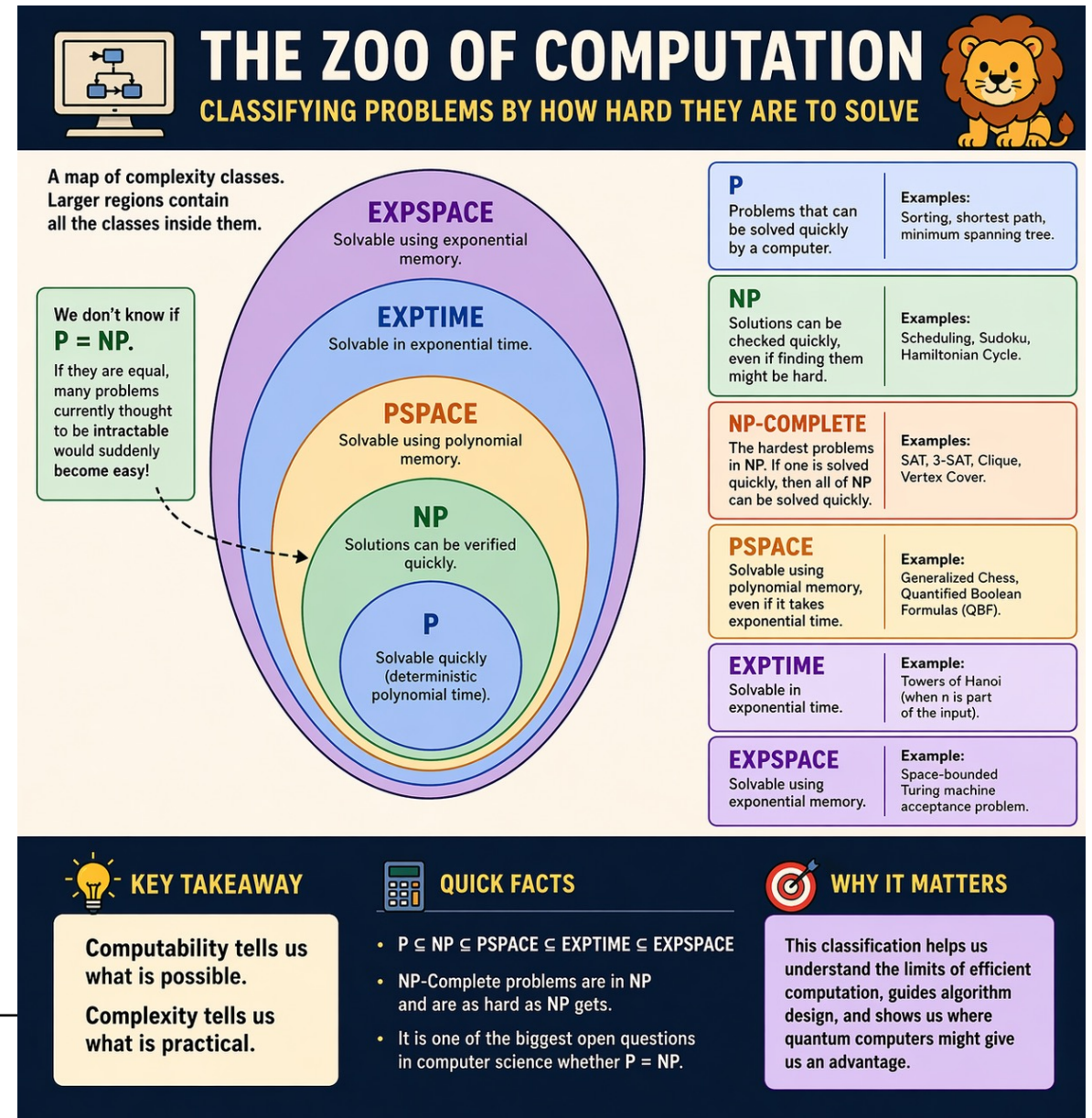
how efficiently can it be computed?

Not all solvable problems are practical.

Some can be solved quickly, others require enormous resources.

To understand this, computer scientists classify problems based on the **time and resources required to solve them**.

These classes (P, NP, EXP, ...) form a “zoo” of computation that helps us distinguish what is feasible from what is not.



WHAT IS A SYSTEM?

Greek σύστημα (systema), meaning:
“a whole composed of parts”

A system is:

- a set of possible states
- a rule for how states change

Examples:

- a physical system: position, velocity, etc.
 - Planets orbiting a sun: solar system
 - a bit: {0, 1}
 - a register: {00, 01, 10, 11}
-

DETERMINISTIC SYSTEMS

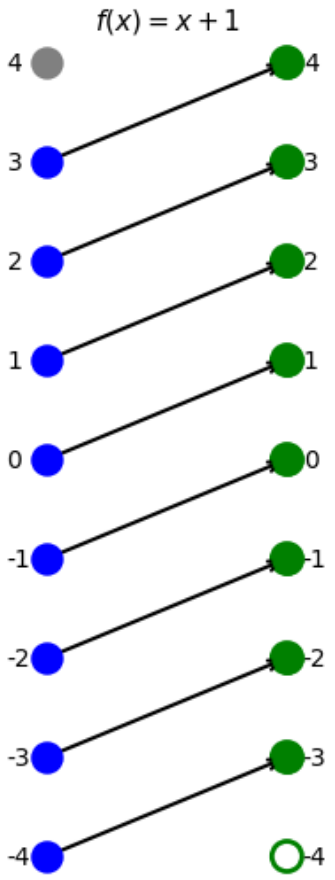
A system is deterministic if:

the next state is completely determined by the current state

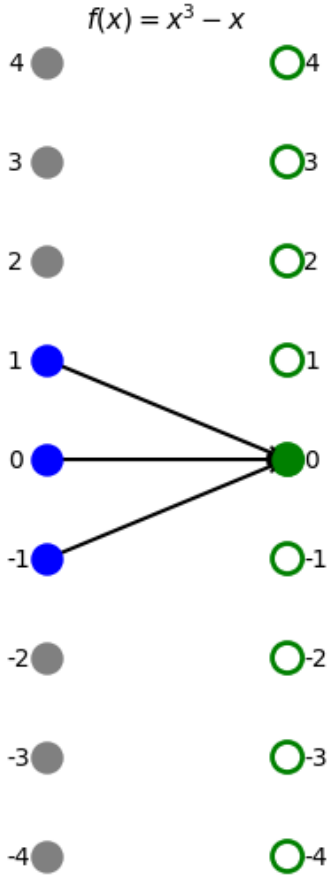
Example:

- NOT gate
- addition
- classical programs (assuming no true random number generator)

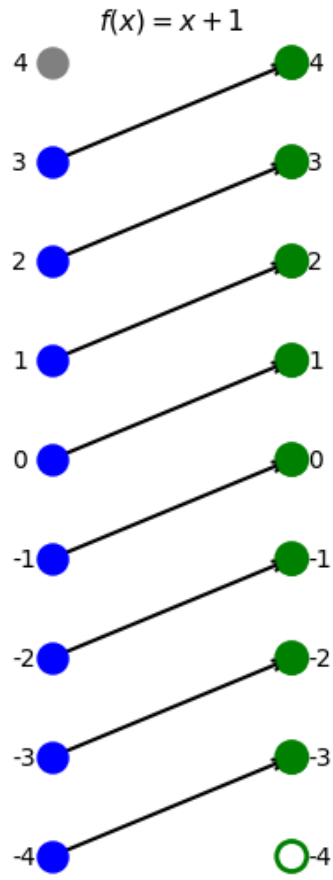
Injective (not surjective)



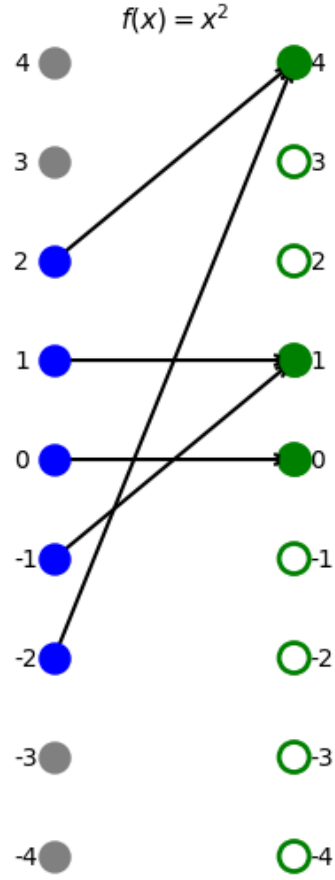
Surjective (not injective)



Bijective on $\mathbb{Z}$



Neither



●

Domain point with visible output

●

Domain point with off-screen output

○

Displayed codomain

●

Visible image

INFORMATION AND REVERSIBILITY

We can define systems that follow transition rules where *information can be lost*

Imagine a binary universe whose laws are AND and OR.

When I say universe, I mean it literally. A universe is states plus rules. Physics tells us the rules of the universe we live in.

The state of the universe evolves according to the ANDs and ORs. But information is lost with each state change, because the different input states yield the same output state (i.e. they are not *injective*)

In other words, we can't run time in reverse in the AND/OR world.

$$\text{And}(0,0) = 0 \wedge 0 = 0$$

$$\text{Or}(0,0) = 0 \vee 0 = 0$$

$$\text{And}(1,0) = 1 \wedge 0 = 0$$

$$\text{Or}(1,0) = 1 \vee 0 = 1$$

$$\text{And}(0,1) = 0 \wedge 1 = 0$$

$$\text{Or}(0,1) = 0 \vee 1 = 1$$

$$\text{And}(1,1) = 1 \wedge 1 = 1$$

$$\text{Or}(1,1) = 1 \vee 1 = 1$$

Going forward in time is well defined but going back in time is ambiguous.

A TIME-REVERSABLE BINARY UNIVERSE: $\neg$

Imagine a binary universe where time is reversible.

This means: Each state change has a unique predecessor and a unique successor i.e. injective and surjective (**bijection**) map.

So, the evolution rule must be: one-to-one and onto (a bijection)

Example: NOT

$$\neg 1 \rightarrow 0 \quad \neg 0 \rightarrow 1$$

Check reversibility:

$$\neg 0 \rightarrow 1 \quad \neg 1 \rightarrow 0$$

No information is lost

Going back and forward in time are well defined.

A TIME-REVERSABLE BINARY UNIVERSE: CNOT

Example: XOR with a control bit (CNOT)

Define a 2-bit universe with state (x, y) , and evolution rule:

$$(x, y) \mapsto (x, x \oplus y)$$

This is the classic **CNOT** rule.

$$\text{CNOT}(0,0) \rightarrow (0,0)$$

$$\text{CNOT}(0,1) \rightarrow (0,1)$$

$$\text{CNOT}(1,0) \rightarrow (1,1)$$

$$\text{CNOT}(1,1) \rightarrow (1,0)$$

Flips one bit only when
another bit is on.

CNOT is bijective so every operation is reversible.

Going back and forward in time are well defined. We will see CNOT again in quantum computing.

REPRESENTING STATES AS VECTORS

We **encode** system states as vectors.

For a bit:

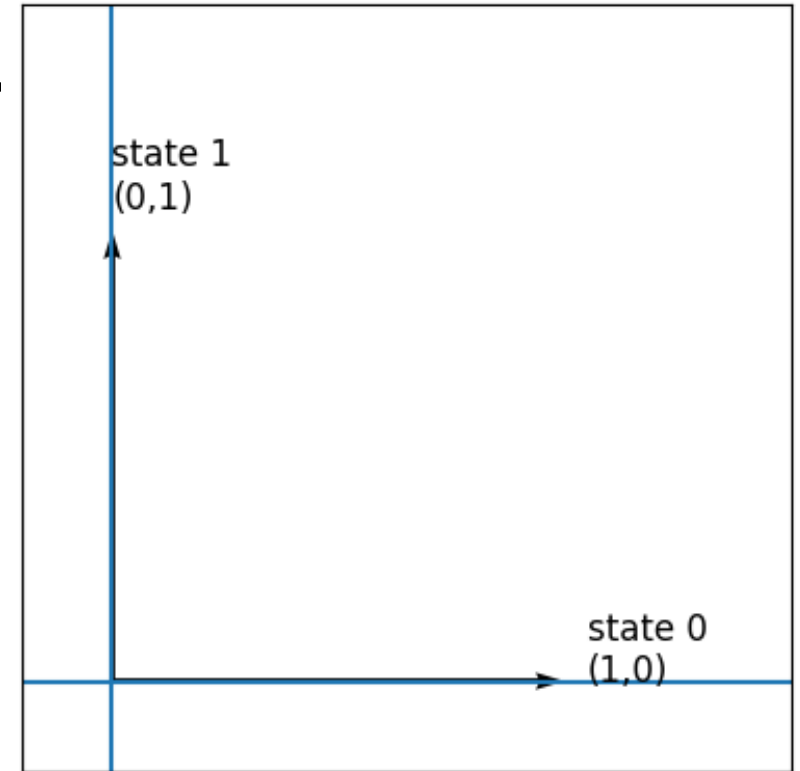
$$0 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad 1 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

This is a **basis representation** from Lecture 5.

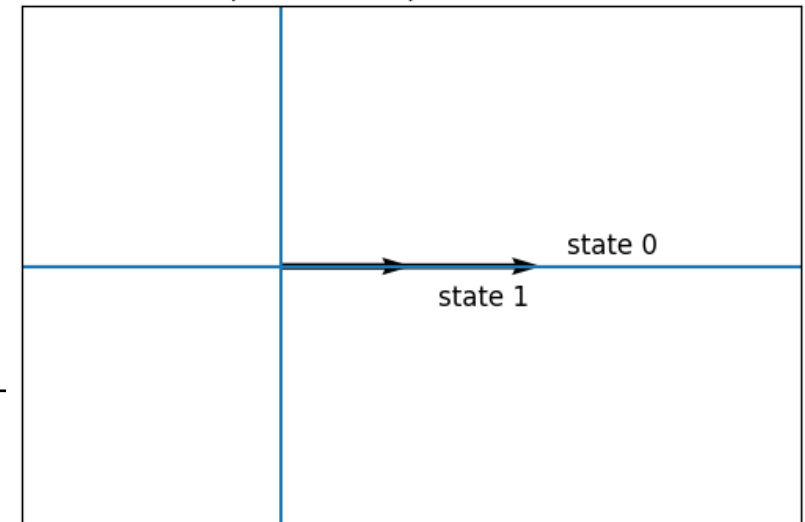
We are putting 0 and 1 into a 2D space purely so we can represent them as vectors and manipulate them with linear algebra and because we are building up to something.

These vectors need to form an orthonormal basis.

Two states $\rightarrow$ two independent directions



1D space: no independent directions



DETERMINISTIC OPERATIONS AS MATRICES

A NOT gate:

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Apply it to state 1

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \times 0 + 1 \times 1 \\ 1 \times 0 + 0 \times 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

So: $1 \rightarrow 0$

Apply it to state 0 (the other flip)

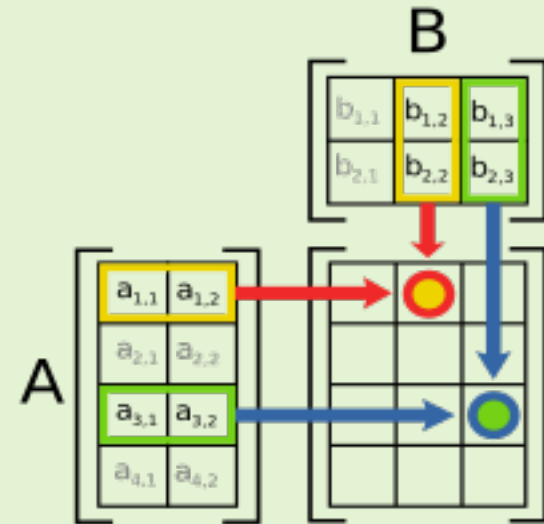
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \times 1 + 0 \times 0 \\ 1 \times 1 + 0 \times 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So: $0 \rightarrow 1$

Matrix Multiplication:

$$A_{2 \times 2} B_{2 \times 2} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

$$A_{2 \times 2} B_{2 \times 1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$



Deterministic and reversible maps

A deterministic transition matrix has one 1 in each column. It may send different inputs to the same output.

A reversible deterministic map is a permutation: it also has one 1 in each row. Reversibility requires retaining enough information to recover the input.

COMPOSING STATE TRANSITIONS

Applying two operations:

$$A(Bx) = (AB)x$$

So:

programs = matrix multiplication

GATES

A classical gate is a function that transforms bits.

Example:

NOT gate

$$\begin{array}{l} 0 \rightarrow 1 \\ 1 \rightarrow 0 \end{array}$$

We can represent this as a matrix acting on a vector.

Represent bits as vectors:

$$0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

NOT gate:

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

Then

$$X \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Ancillas and reversible logic

Develop Toffoli and Fredkin truth tables, clean workspace bits, and uncomputation.

Add a small reversible puzzle with its allowed operations and a proof that the final ancillas reset.

A reversible oracle contract

Define input bits, output bits, and the reversible embedding of a Boolean function.

Test the complete truth table. This oracle contract will support Deutsch and the later query algorithms.