Quantum Computing

Lecture 1 - State, Information, and Computation

Kyle Bomeisl

December 2024

1 Lesson Philosophy and Roadmap

To subvert the classics you must first master them. Understanding quantum physics means subverting a lot of ideas that feel like common sense. We assume the moon is still there when we aren't looking at it. It seems like a safe assumption that measuring the temperature outside won't suddenly make it much hotter. Something can't be in two places at once. These are all ideas rooted in classical physics. Unfortunately - or to one's great fascination depending on your perspective - quantum computers completely uproot these ideas. They operate in the realm of quantum physics where things aren't there if you aren't observing them, measuring something can drastically change it, and an electron can be anywhere until you measure it. Internalizing and building a physical intuition for such a world can be extremely fun, if a bit disorienting. A satisfying transition from the classical world - the arena of all our experiences with physics since birth - to quantum physics requires a firm footing in the former. Only by first seeking a deep understanding of the fundamentals of classical physics can we fully appreciate the quantum subversion of them. To these ends, we will focus on three fundamental physical notions: state, information, and computation. These will be the three big picture motifs of this course and the conceptual glue between classical and quantum computation. Both fundamental similarities and groundbreaking differences between quantum and classical computers are rooted in these three ideas.

2 State

What is the state of a physical system? How would one describe the state of a computer? These two questions intersect in a computer's hardware. Every aspect of a computer's software state - what the screen looks like, what programs are running, what music is playing in iTunes - is is completely determined by the hardware's physical state. At any frozen slice of time, a computer is a function of the state of a physical system. In digital computers the underlying physical system is digital circuits. Nanoscale digital circuits called flip-flops store the state of each bit as a voltage. Each flip-flop holds one of two voltages: $V \in \{V^-, V^+\}$. Software interprets the low voltage V^- as a 0 and the high

voltage V^+ as a 1. The notes currently rendered on your screen are read from hardware and translated to a series of 0s and 1s. Each character in this sentence is a series of 0s and 1s read from your computer's hardware. Software uses the ASCII convention to convert bit strings like '01000001' to the letter 'A.' Different computers have hardware built out of different physical systems. The physical system determining an abacus's state at any time is the position of its beads on a grid. The state of certain quantum computers can be specified by the polarization of a series of photons. There are even computers that store their state in algae! Any way you build a computer, it is software interpreting the physical state of hardware. A computer's state is inescapably physical.

2.1 Modeling a System

A physical system's state is given by a configuration of its physical parameters. Which parameters are significant? How do we know what variables determine the state of the system and which are irrelevant? To even define a system's state, we need a model of the system. Two models will be used for physical systems: classical physics and quantum physics. Consider the model of Newtonian physics. Take the simplest example: a one dimensional point particle. The state of this physical system at any point in time is fully determined by the value of the particle's position and velocity: $S_i = \{x_i, \dot{x_i}\}$. A change in state requires one or both of these physical variables to change: $S_{i+1} = \{x_{i+1}, \dot{x_{i+1}}\}$. In a physical system, nature executes its own program to evolve the state: F=ma - Newton's second law. This evolves the point particle from one state to another over time. Can this simple physical system model the hardware of a real computer? Consider a simple example. An abacus's hardware consists of beads on a rod that are constrained to move in one dimension: $State = \{x_1, x_2, x_3, ...x_n\}$ (where x_i is the position of the ith particle along the x axis). This model makes an isomorphism between the state of the abacus and the 1D particle obvious. Each bead is congruent to a particle in 1D space. As an isolated system, the state of the point particles (and the abacus's beads) will evolve as a system in force equilibrium according to Newton's laws - nothing will happen. This is a good thing. A computer's state should not change unless explicitly ordered to. This is the only circumstance in which the state of the abacus should change. Therefore, any physical model isomorphic to it must be in static equilibrium. Performing a computation on the abacus corresponds to a human being moving the beads to different positions on the rod. This corresponds isomorphically to forces acting on the 1D particles. A more complicated example can be constructed by modeling a digital computer's state. The circuitry of a digital computer's hardware is enormously complicated, and the physical laws governing it even more so. The details of semiconductor physics and how the flip-flop circuits produce high voltage V^+ or low voltage V^- are not considered by the computer's software layer. Only a series of these voltages $\{V^-, V^+\}^n$ is input to the software layer of the computer. The software layer takes this set of voltages as input and abstracts them to a series of bits $\{0, 1\}^n$. Models of computation such as the Turing Machine or Finite State Machine do not consider the hardware layer. They do not concern themselves with the physics of the innards of the computer. Similarly, the semiconductor and digital circuit physics that govern the hardware of the computer are not featured in the

model. While the semiconductor physics of a digital computer's circuits are abstracted away, they still completely determine the state of our model. If we changed the physical laws of digital circuits our computer would stop working the way it does. Similarly, if the laws of physics changed from classical physics to quantum physics computation, state, and information should change accordingly. This is the reason that computer state is inescapably physical. Quantizing the classical computer will be a transparent, logical, and satisfying journey using physical models of computation.

2.2 Classical State

State and how it transforms under computations is easiest to discuss in the language of mathematics. The state of a system will be represented as a vector on which computations act. The natural mathematical representation of a computation is then a matrix. Information, what an observer knows about the state, will be modeled by a vector as well. A state vector of a system with state variables $\{s_i\} = \{s_1, s_2, s_3, ...s_n\}$ can be written as a vector: $\vec{S} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ ... \\ s_n \end{pmatrix}$. For a given state configuration $S = \{s_1 = a_1, s_2 = a_2, s_3 = a_3, ...s_n = a_n\}$, the corresponding state vector is: $\vec{S} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ ... \\ a_n \end{pmatrix}$. A bit's state vector looks like: $\vec{S_b} = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix}$. The state vector of a bit in the 0 state is: $\vec{S_b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and in the 1 state: $\vec{S_b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. A fair coin has the state vector: $\vec{S_c} = \begin{pmatrix} a_{heads} \\ a_{tails} \end{pmatrix}$. The state vector of the heads state is: $\vec{S_b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and for the tails state: $\vec{S_b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Usefully, our mathematical model of the state of these two systems has produced an isomorphism between the two. The state of a fair coin and a bit's state have the same mathematical representation suggesting the isomorphic mapping (coin side) $\rightarrow$ (bit value). Both the fair coin and the bit are known as two-state systems.

3 Computation

State would be simpler if it were static, but dynamic state is much more interesting and realistic. How does state change? A physical system evolves according to the laws of nature and a computer's state changes according to a program. The particular mechanisms of how nature updates the state of the universe continuously to fulfill the laws of nature is unknown. In a computer, it's the CPU. Fascinatingly, all these details can

be abstracted away to reveal that both your laptop and the universe can be considered computers.

3.1 Deterministic Computation

A deterministic computation changes the state in a way that the outcome is known ahead of time. It is a function mapping the input state to the output state. Since states are vectors, computations are matrices. A matrix representing a deterministic computation takes the following form: $C_{ij} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1n} \\ a_{21} & a_{22} & a_{23} & ... & a_{2n} \\ a_{31} & a_{32} & a_{33} & ... & a_{3n} \\ ... & & & & \\ a_{n1} & a_{n2} & a_{n3} & ... & a_{nn} \end{pmatrix}$. Each matrix entry a_{ij} can only be 0 or 1 $a_{ij} \in \{0, 1\}$. Consider a two-state system's state as an example: $\vec{S}_2 = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$. The simplest deterministic computation possible on this state is the identity computation: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$. The identity computation returns the same input. It does not change the state. The next natural example is the NOT Boolean computation: $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix}$. This computation flips the state of the system. A real system modeled by these mathematics is the fair coin. A NOT Boolean computation would change the coin's state from heads to tails or from tails to heads. As a final example, there is the constant computation: $\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_0 \end{pmatrix}$. Any matrix that fulfills the criteria $a_{ij} \in \{0, 1\}$ models a deterministic computation. Vice versa, any deterministic computation can be modeled by such a matrix.

3.2 Probabilistic Computation

Probabilistic computations are modeled by stochastic matrices. A stochastic matrix must fulfill two requirements:
1) All of its entries are positive real numbers
2) The entries of every column sum to 1
The state of a system is known after operating on it with a probabilistic computation. However, this state will be in the form of a probability distribution over the possible results of a measurement. A deterministic computation tells us exactly what the state of the system will be *in addition to* the result of any measurement on the system. This is the difference between state and information. After flipping a fair coin, its state vector will look like: $\vec{S}_c = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$. This represents the information we as an observer have about the coin. When performing strictly deterministic computations, state and information are congruent. The state vector shows exactly what the result of a measurement will be. Probabilistic computations can produce probabilistic state vectors that only tell you the information one has about the system. All we know about the flipped

coin before looking at it is summed up in its state vector: $\vec{S_c} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$. It has a 0.5 probability of being in the state of heads and 0.5 probability of being in the state of tails. A probabilistic state vector can be written as a linear combination of deterministic state vectors: $\vec{S_c} = 1/2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 1/2 \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. In classical physics, the coin really is in one of those states even before we measure it. This will change in quantum physics. Though the mathematical model of state, information, and computation as vectors and matrices will survive unscathed. The "coin flip computation" can be modeled with a "coin flip operator" matrix: $\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix} = 1/2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 1/2 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Probabilistic computations can be written as linear combinations of deterministic computations. The coin flip operator written in the form can be read as a one half probability that the identity operator is applied so the coin remains in its original state and a one half probability that the flip operator is applied changing the state of the coin.

