Quantum Computing 1/27/2022

Lecture 2: From Classical to Quantum Computation

Instructor: Dieter van Melkebeek

In this lecture, we discuss what a computational problem is and introduce the circuit family model of computation which can be used to solve such problems. We go on to show how to track the state of a computation running in a circuit using the concepts of linear algebra and highlight the differences in how we represent states and gates in deterministic, probabilistic, and quantum computation paradigms.

1 Computations

A computational problem can be thought of as a transformation from inputs, x, to outputs, y. This can be represented using a mathematical relation:

$$R = \{(x, y) \in \{0,1\}^* \times \{0,1\}^* : \ y \text{ is a valid output on input } x\}$$

Note that we use a relation and not a function because it is possible for an input to have multiple valid outputs.

The solution to such a problem is a physical process (computation) that transforms the state of the system from an initial state that represents x to a final state that represents y satisfying $(x, y) \in R$.

There are three phases in a computation:

1. Initialization

Preparing the state of the system to represent x. This is quite trivial in classical computations, but can be non-trivial when it comes to quantum computations.

2. Sequence of elementary operations

A sequence of simple steps that alter the state of the system to eventually represent a valid y.

3. Read out

Observing the state of the system and inferring y from it. This step may also be non-trivial in quantum computations.

There are two requirements for such a physical process to be regarded as a computation:

- **Locality** — The elementary operations act locally. That is to say, the components of the system that these operations act on must be physically or geometrically near each other.
- **Finite description** — The sequence of operations must follow from a finite description, e.g. the source code for a program.

2 Deterministic Computation Models

2.1 Turing Machines

Turing machines are a common model for deterministic computation. They consist of a *tape* with symbols on it and a *head* which can read symbols from the tape and overwrite the symbol on the tape. The exact details of how Turing machines work is not required in this course.

Turing machines satisfy both requirements to be considered computational models:

- **Locality** — The elementary operations of a Turing machine are reading the symbol on the tape where the head is, overwriting the symbol on the tape where the head is, and moving the head along the tape. The head can only alter the state of the tape in the position it is present on the tape, thus the operation is local.
- **Finite description** — There are a finite number of control states that affect the behavior of the Turing machine and a finite number of transitions between these states (encoded in the transition function).

Church-Turing Thesis *Every computation can be simulated on a deterministic Turing machine.*

This thesis has been unchallenged to date. It allows you to think of a Turing machine like a program written in any programming language, i.e. it can be thought of as a classical computer.

This means that any computation that a quantum computer can do, a Turing machine/classical computer can also do. Equivalently, if a classical computer cannot perform a computation then, a quantum computer also cannot perform that computation. Quantum computers observe no advantage with respect to the *types* of problems that they can solve.

Strong Church-Turing Thesis *Running time of a simulated Turing machine is bounded by a polynomial in number of steps of simulated computation.*

This thesis is unchallenged to date for deterministic computations. However, it may fail for probabilistic computations (but is conjectured to hold true) and quantum computations (this is the conjecture).

The advantage of quantum computers over classical computers comes from this conjecture, that there are some computations that can be performed more efficiently on quantum computers than on classical computers.

Probabilistic and quantum variants of Turing machines exist as models for those types computations. But quantum Turing machines are cumbersome to deal with and so we'll not be discussing them in the course.

2.2 Circuit Families

Circuits offer an alternative model for deterministic computation, you can think of Boolean logic circuits as an example of this model.

Every circuit operates on inputs of a fixed length, n. So to perform a computational problem, we need a family of circuits, one for every possible input size, $\{C_n\}_{n\in\mathbb{N}}$. C_n is a circuit that operates on an input of size n.

The operations/gates in the circuit are local if they act on elements that are physically near each other. If the elements are not physically near, they can be brought nearer with the help of SWAP gates. This adds a linear overhead in the run time of the circuit but ensures locality.

A finite description however is not guaranteed by a family of circuits. We can't have a unique circuit description for every n if we wish to consider the circuit family a computation. A circuit family is called *"uniform"* if given n, there is a Turing machine that can generate a description for C_n. Since all circuits of a uniform family can be generated with a finite description (the description of the Turing machine), a uniform family of circuits can be considered as a computational model as it meets both requirements.

Note that we need to account for the resources (time and space) required both to the generate the circuit and to run it for an input. The run-time resources usually dominate over the generation-time resources, so when designing algorithms (circuits) the generation-time resources are usually overlooked.

Uniform circuit families are equivalent to Turing machines up to polynomial factors in running time. And so, the string Church-Turing thesis will hold equally well for uniform circuit families as Turing machines.

Probabilistic and quantum variants of uniform circuit families also exist, it is only the type of circuit that will change. The Turing machine that generates the circuits will be deterministic for all cases.

Quantum circuits are convenient to deal with and we will use them to study quantum algorithms in the course.

3 Linear Algebraic View of Circuit Computations

Recall that computations are transformations from a binary string input to a binary string output. The state of a circuit can be represented using a string of $m \geq n$ components that can be in the state 0 or 1. When the system is initialized, the first n bits are set to match the input string x and the remaining bits are set to 0.

The elementary operations of a circuit (called gates) act on a constant number of components. The geometric proximity of the components is usually ignored during algorithm design. As mentioned previously, SWAP gates can be used to swap neighboring components when the algorithm is implemented.

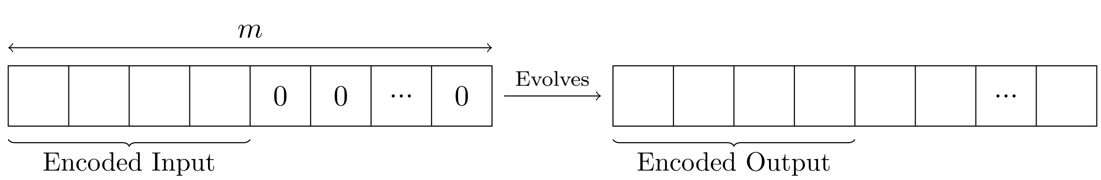

Figure 1: An overview of how the state string evolves in a deterministic computation. First the state is initialized to hold the input, x, with the remaining bits set to 0. Right before readout, the state string of the system will have a substring that encodes the output, y, of the computation.

At any point, the state of the system can be described with an m-bit string s. The gates in a circuit are mappings from one such string to another. Gates only affect a constant-size part of the input string since they follow locality i.e. a gate acting on $k \leq m$ components/bits will map the input to an output state differing in *only* those k bits.

An alternate way of representing the system states is to use 2^m-dimensional vectors. If we do so, the gates can be represented as linear operators acting on these state vectors. We'll see what these state vectors and transition matrices look like for deterministic, probabilistic and quantum computations.

3.1 Deterministic Computation

The state vectors for deterministic computations are binary vectors where exactly one element of the vector is 1 and the rest are 0.

$$|\psi\rangle = \begin{bmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{(2^m-1)} \end{bmatrix}, \ \psi_j \in \{0, 1\}$$

Indexing the elements of the vector starting from 0, allows us to write each index as an m-bit integer. The state vector corresponding to the state string s is the binary vector where $\psi_s = 1$ and all other elements are 0.

Example 1 (State vectors for $m = 2$).

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \ |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \ |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

If the system is in state $|s\rangle$ and a gate with transition matrix T is applied on the system, the resulting state can be calculated as $T|s\rangle$.

Example 2 (Transition matrix for the NAND-gate: $s_2 \leftarrow \overline{s_1 \wedge s_2}$).
The state transitions for the NAND-gate are:

$$|00\rangle \mapsto |01\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle$$

The matrix that captures this mapping is

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Every column of the transition matrix is a valid state vector, i.e. each column has exactly one 1.

Example 3 (Transition matrix for the XOR-gate: $s_2 \leftarrow s_1 \oplus s_2$).
The state transitions for the XOR-gate are:

$$|00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle$$

The matrix that captures this mapping is

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

If every row of a transition matrix, T, also has exactly one 1, this implies that the gate it represents is reversible. That is, there is a gate corresponding to the transition matrix T^{-1}.

Effect when a gate acts on $k < m$ bits

The gate only affects the components on which it acts. For each fixed setting of other components, the gate performs a linear operation defined by T, and the overall effect on the system is also a linear operation which is given by a tensor product of T and identities.

Example 4 (Bit-flip gate).
Considering a 2-bit system where the bit flip gate changes the state of the first bit only. The transition matrix of the bit flip is

$$|0\rangle \mapsto |1\rangle \,, \ |1\rangle \mapsto |1\rangle \,, \ \implies X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Nothing happens to the second bit, so the transition matrix for it is the identity matrix, I. The transition matrix for the whole system is a tensor product of T and I:

$$X \otimes I = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes I = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

And we can verify that this does match the expected mapping for the bit-flip:

$$|00\rangle \mapsto |10\rangle \,, \quad |01\rangle \mapsto |11\rangle \,, \quad |10\rangle \mapsto |00\rangle \,, \quad |11\rangle \mapsto |01\rangle$$

If we consider a bit flip on the second bit instead, we change the order of the tensor product:

$$I \otimes X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes X = \begin{bmatrix} X & 0 \\ 0 & X \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Again, we can verify that the above matrix follows this mapping:

$$|00\rangle \mapsto |01\rangle \,, \quad |01\rangle \mapsto |00\rangle \,, \quad |10\rangle \mapsto |11\rangle \,, \quad |11\rangle \mapsto |10\rangle$$

For more info about how to calculate the tensor products of matrices, you can read up on the "Kronecker product."

3.1.1 Summary

- State vectors are vectors with exactly one 1 and remaining 0s.
- Columns of transition matrices also have exactly one 1 and remaining 0s.

3.2 Probabilistic Computation

The elementary operations can depend on the outcome of "coin tosses" in the case of probabilistic computation. As such, the state of the system becomes a random variable. We can write the state as a linear combination or "superposition" of vectors:

$$|\psi\rangle = \sum_{s \in \{0,1\}^m} p_s |s\rangle = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{(2^m-1)} \end{bmatrix}$$

Where $p_s = \Pr[\text{system in state s}]$

The phases of the computation are:

1. **Initialization** — The system is put into the basis state $|x0^{m-n}\rangle$, where x is an n-bit input.
2. **Sequence of probabilistic gates**
3. **Read out** — Observe $|\psi\rangle$, this collapses the superposition into a basis state $|s\rangle$. We will observe the state s with a probability of p_s. The output y is extracted from s.

Because the elements of the vector represent probabilities, $p_s \in [0,1]$ and moreover, the probabilities of each possible observation must add up to 1 so,

$$\sum_s p_s = 1.$$

This sum is called the "1-norm" of $|\psi\rangle$, and is written:

$$\| \, |\psi\rangle \, \|_1 = \sum_s |p_s|$$

Since the computation is probabilistic, there is a chance that at the end of the computation, we will read out the wrong answer, $(x, y) \notin R$. Probabilistic computations are subject to a correctness requirement. For every valid input x,

$$\Pr[(x, y) \in R] \geq 1 - \epsilon,$$

where ϵ is the error bound.

Transition matrices for probabilistic gates

The transition matrices for probabilistic gates observe the following (equivalent) properties:

- The transition matrix is a convex combination of transition matrices of deterministic gates.
- The transition matrix T is stochastic, i.e., T has non-negative entries and satisfies $\sum_j T_{ij} = 1$ for each j (each column sum to 1).
- The linear operators representing these gates transform probability distributions to probability distributions.
- The transformation matrices have non-negative entries and preserve the 1-norm.

Examples of probabilistic gates

All deterministic gates are probabilistic gates.

Coin-toss gate: $\begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$, puts both $|0\rangle$ and $|1\rangle$ in an equal probability superposition $\frac{1}{2}(|0\rangle + |1\rangle)$.

3.3 Quantum Computation

The state can be written as a superposition:

$$|\psi\rangle = \sum_{s \in \{0,1\}^m} \alpha_s |s\rangle$$

with "amplitudes" $\alpha_s \in \mathbb{R}$ (or $\mathbb{C}$), such that $\sum_s |\alpha_s|^2 = 1$.

The phases of the computation are:

1. **Initialization** — The system is put into the basis state $|x0^{m-n}\rangle$, where x is an n-bit input.
2. **Sequence of quantum gates**
3. **Read out** — Measure $|\psi\rangle$, this collapses the superposition into a basis state $|s\rangle$. We will observe the state s with a probability of $|\alpha_s|^2$. The output y is extracted from s.

The measurement probabilities are encoded in the amplitudes of the state vector and the probabilities must sum to 1:

$$\sum_s |\alpha_s|^2 = 1$$

This square root of this sum is called the "2-norm" of $|\psi\rangle$, and is written:

$$\| \, |\psi\rangle \, \|_2 = \sqrt{\sum_s |\alpha_s|^2}$$

Again, this paradigm of computation is probabilistic so quantum computations are also subject to a correctness requirement. For every valid input x,

$$\Pr[(x, y) \in R] \geq 1 - \epsilon,$$

where ϵ is the error bound.

Transition matrices for quantum gates

The transition matrices for quantum gates observe the following (equivalent) properties:

- The transition matrices preserve the 2-norm.
- Transition matrix T is unitary, i.e., $T^*T = I = TT^*$, where T^* is the conjugate transpose of T (also written as $T^\dagger$).

Examples of quantum gates

All reversible deterministic gates are quantum gates.

Hadamard gate, $\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$, this is a quantum coin-toss gate, mapping the basis states to equal probability superpositions (yet both superpositions are distinct due to the negative entry).

Exercise

Consider a system with $m = 1$. Recall the Hadamard gate

$$H = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

Determine the output distributions of each of the following processes:

1. Start in state $|0\rangle$, apply H, apply H again, and observe.
2. Start in state $|0\rangle$, apply H, observe, apply H again, and observe.