UCSB, Physics 129L, Computational Physics
Lecture notes, Week 3

Zihang Wang (UCSB), zihangwang@ucsb.edu

March 14, 2025

Contents

1 Turing Machine

A Turing machine M can be formally described as a quintuple (I swapped the notation Γ, Σ—if you are wondering why it is different from other sources, feel free to ask):

$$M = (Q, \Gamma, \Sigma, \delta, q_0)$$

where:

- Q: A finite **set of states**, representing the state space of the Turing machine.
- Γ: A finite set of **symbols** that form the input alphabet used directly for computation. This set contains symbols of the first kind (e.g., 0, 1).
- Σ: A finite set of **symbols**, representing the tape alphabet on which the Turing machine operates. This set includes both blank symbols B and all symbols in Γ as a subset. Initially, all tape squares are blank except for the input.

- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R, N\}$: The transition function, which is a deterministic algorithm. It takes two parameters as input: the current Turing machine state $q \in Q$ and the current tape symbol $s \in \Sigma$. It produces three outputs: a new Turing machine state q', a modified tape symbol s', and a directional instruction $\mathcal{D} \in \{L, R, N\}$, where: - L: Move left. - R: Move right. - N: Hold position.

  The direction instruction can be loosely understood as the change in the Turing machine's position: left movement, right movement, or no movement. When shifting to the right (left), it is represented by a right (left) notation. Formally, this is expressed as:

  $$\delta(q, s) = (q', s', \mathcal{D}), \tag{1}$$

  where it is defined over finite sets: machine states (Q), tape symbols (Σ), and directions (L, R, N).

  The above transition function can also be written in an alternative form using a quintuple:

  $$qss'\mathcal{D}q'; \tag{2}$$

  Note that semi-colons ";" are used to separate different transition rules. Using this representation allows us to write all transition rules on a single tape, which is critical for constructing a universal Turing machine.

- $q_0 \in Q$: The initial state of the Turing machine.

1.1 Notations

For example, let's consider a simple algorithm that replaces all s_2 by s_1. The **Turing machine** operates at an initial state q_0 on a tape symbol $B \in \Sigma$, (q_0, B),

$$M = (Q = \{q_0, q_1, q_2, q_{halt}\}, \Gamma = \{s_1, s_2\}, \Sigma = \{B, s_1, s_2\}, \delta, q_0). \tag{3}$$

The transition function is given by $\delta(q_0, B) = (q_1, s_1, R)$, which can be understood as follows:

1. The Turing machine, in state q_0, reads the symbol $B \in \Sigma$ on the tape.
2. The Turing machine changes its state to $q_1 \in Q$ and writes the symbol $s_1 \in \Sigma$ onto the tape, replacing B.
3. The Turing machine moves its head to the right (R), resulting in a new configuration where the machine is in state q_1 with its head positioned over the next symbol to the right.

1.1.1 Transition Table (single blank symbol)

The above description can be expressed via the following table (there may be a better way):

Current State	Read	Write	Move	Next State
q_0	B	B	R	q_1
q_1	s_1	s_1	R	q_1
q_1	s_2	s_1	L	q_2
q_2	s_1	s_1	L	q_2
q_2	B	B	R	q_1
q_1	B	B	R	q_{halt}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

(4)

Alternatively, it can be expressed as a one-line expression, as proposed by Turing:

$$q_1 s_1 s_1 R q_1; q_1 s_2 s_1 L q_2; q_2 s_1 s_1 R q_2; q_2 B B R q_1; \tag{5}$$

The term q_{halt} above represents the terminal state of the Turing machine. q_{halt} can either be an accept halt or a reject halt. In modern computation, it can be understood as the status of the "standard error," where 0 indicates that the program completes successfully, and 1 indicates failure. Therefore, the "detection" of errors is directly encoded in the Turing machine.

1.1.2 Standard Form

This can be written in the **standard form**. We replace the symbol $s_i = DC\cdots$ by "D" followed by "C" repeated i times (note that this "D" is not the direction $\mathcal{D}$ mentioned previously). Similarly, the machine state $q_i = DA\cdots$ is replaced by "D" followed by "A" repeated i times. For example:

$$q_0 = D, \quad q_1 = DA, \quad q_2 = DAA, \quad B = D, \quad s_1 = DA, \quad s_2 = DAA,$$

and the quintuple has the following expression:

$$\begin{aligned} &q_0 B s_1 R q_1; q_0 s_1 s_1 R q_1; q_1 s_1 s_1 R q_1; q_1 s_2 s_1 L q_2; \\ =&DDDCRDA; DDCDCRDA; DDCDCRDA; DADCDCRDA; DADCCDCRDAA; \end{aligned} \tag{6}$$

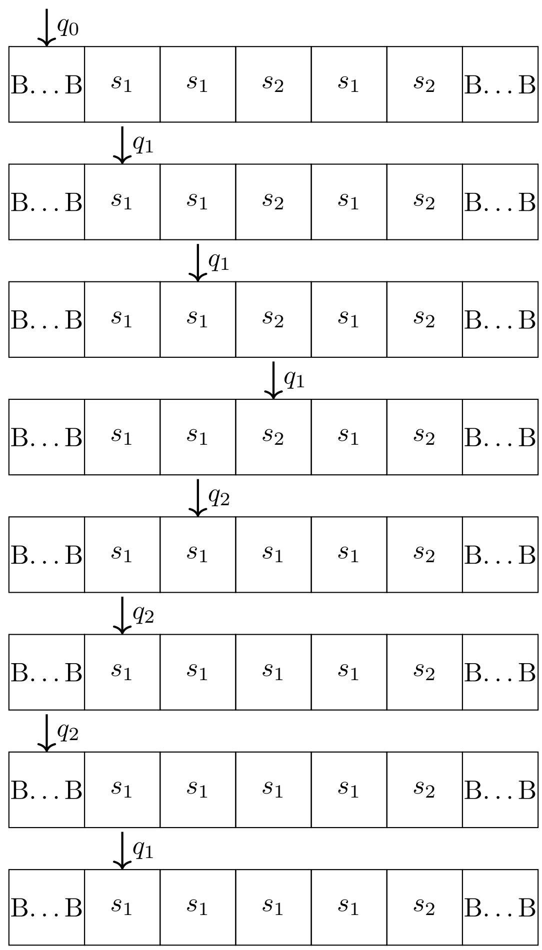
In other words, any Turing machine instruction can be written as combinations of the following 7 letters:

$$A, C, D, L, R, N, ; \tag{7}$$

If we replace "A" with "1," "C" with "2," "D" with "3," "L" with "4," "R" with "5," "N" with "6," and ";" with "7," we obtain a description of the Turing machine in numeral form. These are called **description numbers**.

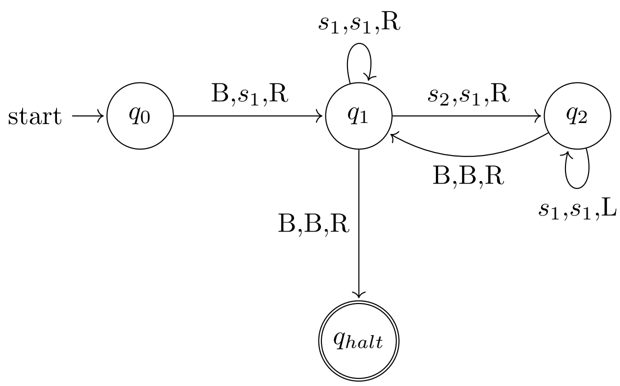
1.1.3 Examples with Tapes

Let's look at the Turing machine in action: The Turing machine moves its head position, and read and write on the tape depending the above algorithm. The following shows the steps,



1.1.4 Transition Diagram

The above expression has the following diagram,



By the end of the process (reaching q_{halt}), the computation result can be read off from the tape or the halt state, $q_{halt} = q_{accept}, q_{reject}$.

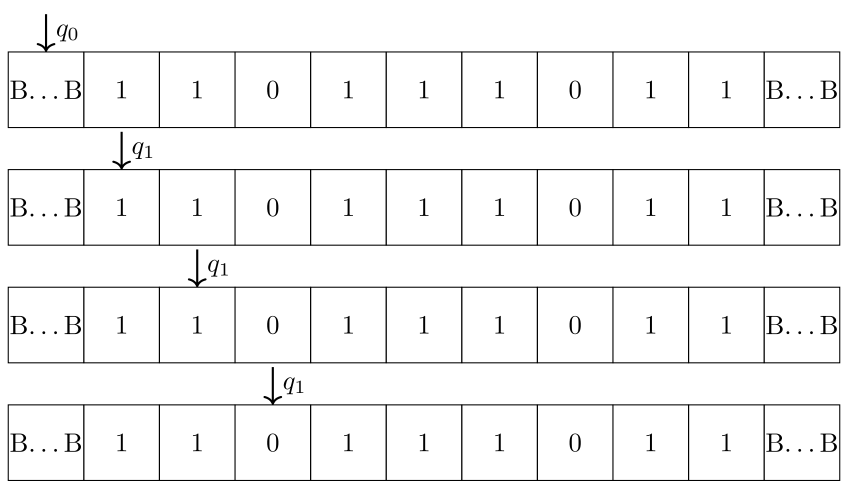
In particular, if $B, s_1, \cdots \in \{0, 1\}$ with n possible states $\{q_i\}$ (state space dimension), the Turing machine is a **2-symbol, n-state Turing machine**.

For example, let's consider the following addition problem, as shown below,

Current State	**Read**	**Write**	**Move**	**Next State**
q_0	B	B	R	q_1
q_1	0	1	R	q_3
q_1	1	1	R	q_1
q_1	B	B	R	q_{halt}
q_3	0	0	L	q_3
q_3	1	1	L	q_3
q_3	B	B	R	q_4
q_4	1	B	R	q_1
q_4	0	0	R	q_4
q_4	B	B	R	q_{halt}

(8)

And a typical binary tape looks like the following,



		q_3								
B…B	1	1	1	1	1	1	0	1	1	B…B

…

q_3										
B…B	1	1	1	1	1	1	0	1	1	B…B

	q_4									
B…B	1	1	1	1	1	1	0	1	1	B…B

							q_1			
B…B	B	1	1	1	1	1	0	1	1	B…B

…

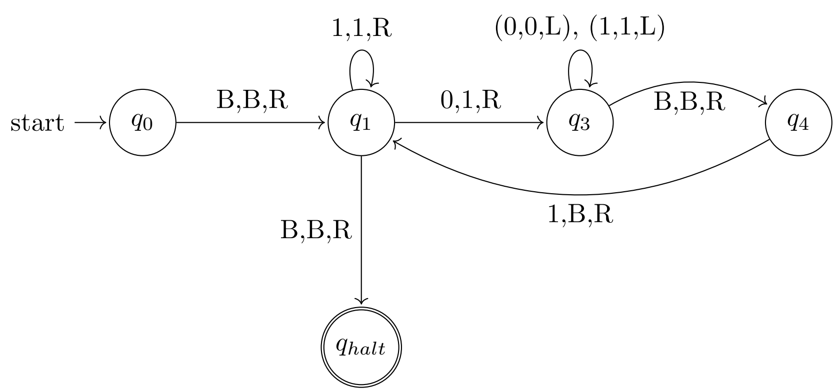
						q_3				
B…B	B	1	1	1	1	1	1	1	1	B…B

…

	q_3									
B…B	B	1	1	1	1	1	1	1	1	B…B

		q_4								
B…B	B	1	1	1	1	1	1	1	1	B…B

			q_1							
B…B	B	B	1	1	1	1	1	1	1	B…B



Loosely speaking, a Turing machine is an algorithm-specific machine under fixed rules, defined within its state space.

The **Church-Turing thesis** proposes that anything computable by an **algorithm** can be computed via a **Turing machine**. For example, a real number is Turing-computable if there exists a **Turing machine** or **algorithm** capable of computing an arbitrarily precise approximation of that number. All algebraic numbers and important constants, such as e and π, that can be determined via root-finding algorithms are **Turing-computable**.

Let's consider the famous **halting problem**, which is **not Turing-computable**: Can we design an algorithm that checks the number of steps a Turing machine takes to halt (with an initial tape)? It turns out that the halting problem is **not Turing-computable** since it is undecidable. In other words, we cannot design a Turing machine that performs this task.

Next, we want to ask: what is the maximum number of steps an n-state Turing machine can take before it halts? This is the famous **Busy Beaver problem**, which involves finding the n-state Turing machine that performs the maximum "work" on a given tape.

The value is called the **Busy Beaver function** (BB), and it can be accessed via an enumerative search algorithm, where we scan all possible n-state Turing machines. The Busy Beaver function grows much faster than **any** Turing-computable function. Here are the known values (we are only able to calculate it up to 6):

$$\begin{aligned} BB(1) &= 1, \\ BB(2) &= 6, \\ BB(3) &= 21, \\ BB(4) &= 107, \\ BB(5) &= 47,176,870. \end{aligned} \tag{9}$$

For larger values where exact results are unknown, we have lower bounds:

$$\begin{aligned} BB(6) &> 10^{865}, \\ BB(7) &> 10^{10^{10^{10^{18,705,352}}}}. \end{aligned} \tag{10}$$

This is so far the fastest-growing function ever discovered.

As a side note, lambda calculus is an alternative approach to the Turing machine, and they emphasize different aspects of computation: lambda calculus focuses on function abstraction and symbolic manipulation, whereas Turing machines provide a step-by-step mechanical process for computation.

The **lambda function**, denoted by the symbol λ, represents an **anonymous function** that takes an input variable and maps it to an output that follows certain rules. Given a function $\lambda x.M$ and an argument N, the result of applying the function is the expression M with x substituted by N, written as $(\lambda x.M)\,N$.

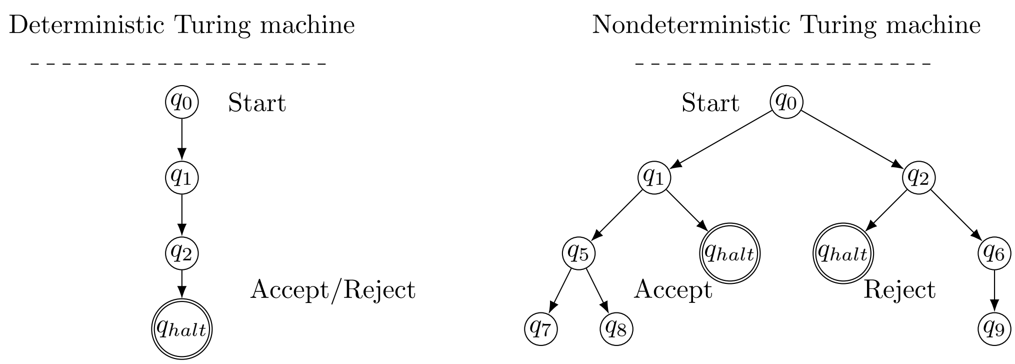
As a side note, lambda calculus is an alternative approach to the Turing machine, and they emphasize different aspects of computation: lambda calculus focuses on function abstraction and symbolic manipulation, whereas Turing machines provide a step-by-step mechanical process for computation.

Lambda function, denoted by the symbol λ, represents an **anonymous function** that takes an input variable and maps to an output that follows certain rules. Given a function $\lambda x.M$ and an argument N, the result of applying the function is the expression M with x substituted by N, written as $(\lambda x.M)\, N$.

2 Nondeterministic Turing machine

The above examples are deterministic Turing machines since, from a given initial condition, the trajectory is fully deterministic, following the transition functions.

A **nondeterministic Turing machine** is a theoretical model of computation that extends the concept of a deterministic Turing machine. Unlike a deterministic Turing machine, which follows a single, deterministic path of steps based on its current state and tape symbol, a **nondeterministic Turing machine** can "branch" into several possible states and form a **computation tree** that **explores all computational paths simultaneously**. The difference between a nondeterministic Turing machine and a deterministic Turing machine is illustrated in the diagram below,



3 Universal Turing machine

We should note that a Turing machine is designed for executing a single algorithm, and it is single-purposed. Can we make a Turing machine that can simulate other Turing machines? This is called the **universal Turing machine**, and it is able to read a tape that contains the following information,

- Description of another Turing machine M (its states, symbols, and transition rules).

- An input string w that the described machine M would process.

The UTM uses these inputs to simulate M's behavior on w, producing the same result as M would. To achieve this, we must put both the building instruction for M and the input w on the tape.