Turing Machines

Last Updated November 18th, 2025

1 Introduction

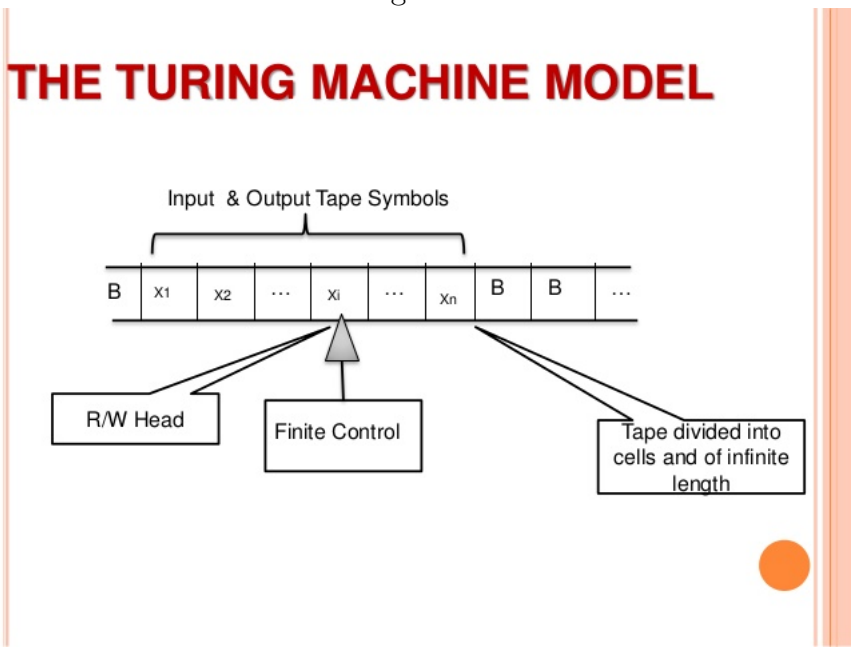
In this lecture we look at the Turing machine model of computation. A Turing machine is a kind of automaton that is more powerful than a pushdown automaton since the former is equipped with an infinite stack memory while the Turing machine has a memory consisting of an infinite array of cells that can be accessed in any order. Turing machines are equivalent in power to other models of computation, including modern programming languages.

Although Turing machines can seem very difficult to work with in practice, they have the following theoretical advantages.

Turing machines can be easily described in the abstract (as a 7-tuple), which makes them preferable to work with when developing a theoretical argument about general computation.

Turing machines highlight the two features that yield general computation: a finite control together with read/write acccess to an infinite array of memory cells that may be accessed in any order.

Figure 1:



1.1 Turing machine definition

A **Turing machine** consists of a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

Q a finite set of states

Σ the finite input alphabet, not including the blank symbol $\sqcup$

Γ the finite **tape alphabet**, where $\Sigma \subset \Gamma$ and $\sqcup \in \Gamma$

δ a transition function δ that, given the current state and the tape symbol being read, determines the machine's next state, the next tape symbol to be written, and the direction for the tape head to move. In other words,

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}.$$

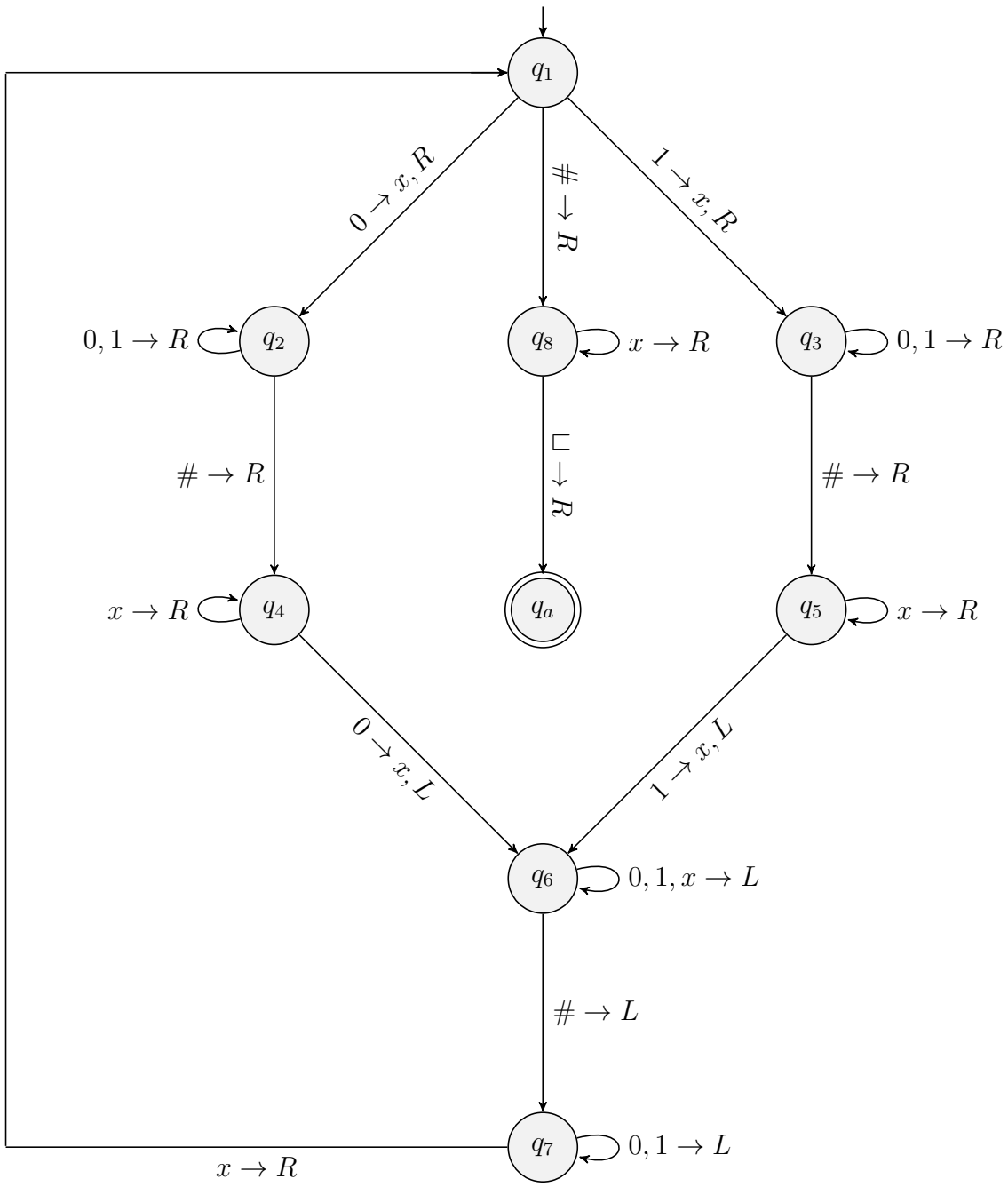
$q_0 \in Q$ the **initial state**

q_a the **accepting state**

q_r the **rejecting state**

It helps to think of a Turing machine as consisting of a one-way infinite **tape** (i.e. array) of symbols from Γ, where at the beginning of a computation the tape consists of input symbols $w_1 \cdots w_n$, followed by an infinite sequence of $\sqcup$ symbols. The **tape head** reads one tape cell and moves either right or left according to δ. Before moving, it first writes a new symbol on the current cell being read.

Example 1. The following is a state diagram for a Turing machine that halts in the accept state on some input iff the input has the form $w\#w$, where $w \in \{0,1\}^*$.



State Annotation.
q_1: search for the next bit b and cross it out
q_i, $i = 2, 3$: search for a matching b bit ($i = 2$: $b = 0$, $i = 3$: $b = 1$) but first locate #
q_j, $j = 4, 5$: search for a matching b bit ($j = 4$: $b = 0$, $j = 5$: $b = 1$) right of #
q_6: move left and locate #
q_7: move left and locate the nearest x
q_8: no more bits left of #, make sure there are none to the right of #.

Simplifying conventions

1. If $\delta(q, s)$ is undefined, then we take it to mean $\delta(q, s) = (q_r, s, R)$ which results in a rejecting computation.

2. If the tape head is reading the first tape cell and is instructed to move left, then we assume the head moves to a "buffer" blank cell and that the next instruction must force the head to move back to the first tape cell without writing on the buffer cell.

Machine Configurations

Recall that, when performing a computation with some DFA M, at any step of the computation it suffices to know i) the current input symbol being read and ii) M's current state. Then the input tape tells us the next symbol to be read while we may consult M's state diagram to determine the next state. If M is a Turing machine then we need the following three pieces of information to process the current step and prepare for the next step.

1. the current state q
2. the contents of the tape memory, and
3. the location of the tape head.

Definition 1. The **memory word** of a Turing machine is a word $w \in \Gamma^*$ having length $m \geq 1$ and for which w_i equals the tape symbol that appears in cell i, $i = 1, \ldots, m$. Furthermore, m is such that either i) cell m contains a non-blank symbol and cells $m+1, m+2, \ldots$ all contain blank symbols, or ii) cell m contains a blank symbol, the head is reading cell m, and cells $m+1, m+2, \ldots$ all contain blank symbols.

Example 2. For the TM from Example 1, if the input word is 101#101, provide the first five distinct memory words that appear in the computation. What is the final memory word of the computation?

Definition 2. Let w be the memory word that is associated with a computation step of a TM M for which the current state is q and the head is reading cell i, for some $1 \leq i \leq m = |w|$. Then the **configuration (word)** associated with this computation step is the word κ over the alphabet $(\Gamma \cup Q)^*$, where Q is the set of states of M and

$$\kappa = w_1 \cdots w_{i-1} q w_i \cdots w_m.$$

Notice that a configuration for some computation step provides all the information that is needed in order to determine the next state, head location, and memory word.

Example 3. For each memory word that was listed in Example 2, provide the corresponding configuration that is associated with the first occurrence of the word during the TM computation.

Special Turing machine configurations

Initial Configuration has the form $\vec{\kappa} = q_0 w_1 \cdots w_n$, where w is the input word

Final Configuration any configuration $\vec{\kappa} = w_1 \cdots w_{k-1} q w_k \cdots w_m$, where q is either q_a or q_r. In the former case we call it an **accepting configuration**, while in the latter case we call it a **rejecting configuration**.

Turing machine computation

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ be a Turing machine. Then the **Turing machine computation** of M on input w consists of a (possibly infinite) sequence $S(M, w)$ of configurations $\vec{\kappa}_0 \vec{\kappa}_1, \ldots$ which is recursively defined as follows.

Base case $\vec{\kappa}_0 = q_0 u_1 \cdots u_n$ is the initial configuration for some $u = u_1 \cdots u_n \in \Sigma^*$

Recursive case Assume that $\vec{\kappa}_i = w_1 \cdots w_{j-1} q w_j \cdots w_m$ has been defined, $q \notin \{q_a, q_r\}$, and

$$\delta(q, w_j) = (\hat{q}, a, D),$$

where $D \in \{\mathrm{L}, \mathrm{R}\}$.

Case 1: $D = \mathrm{R}$ and $j < m$. Then

$$\vec{\kappa}_{i+1} = w_1 \cdots w_{j-1} a \hat{q} w_{j+1} \cdots w_m.$$

Case 2: $D = \mathrm{L}$ and $j > 1$.

$$\vec{\kappa}_{i+1} = w_1 \cdots \hat{q} w_{j-1} a w_{j+1} \cdots w_m.$$

Case 3: $D = \mathrm{R}$ and $j = m$. Then

$$\vec{\kappa}_{i+1} = w_1 \cdots w_{m-1} a \hat{q} \sqcup.$$

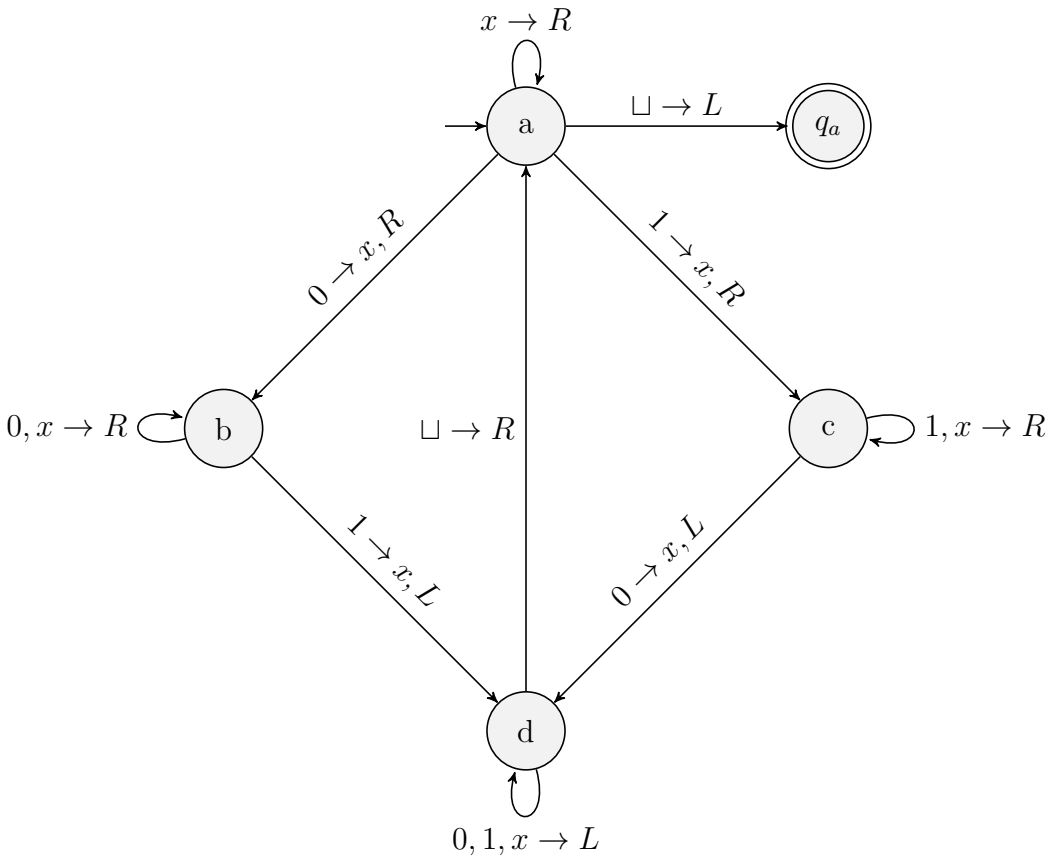
Case 4: $D = \mathrm{L}$ and $j = 1$.

$$\vec{\kappa}_{i+1} = \hat{q} \sqcup a w_2 \cdots w_m.$$

Moreover, the computation of M on input w is said to be a finite computation if and only if $|S(M, w)|$ is finite. In this case we say that M **halts** on input w. Notice also that in this case the final configuration of $S(M, w)$ must either be accepting or rejecting, in which case we either call $S(M, w)$ an **accepting computation**, or a **rejecting computation**, and say that M accepts or rejects the input. The language L accepted by M, denoted $L(M)$, consists of all words $w \in \Sigma^*$ for which M accepts w. Such a language is said to be **Turing recognizable** or **recursively enumerable**. Moreover, if M halts on all inputs, then $L(M)$ is said to be **Turing decidable** or **recursive**.

Example 4. Provide the state diagram for a Turing machine M that accepts all binary words having an equal number of zeros and ones. Show the computation of M on input 0110.

Solution.

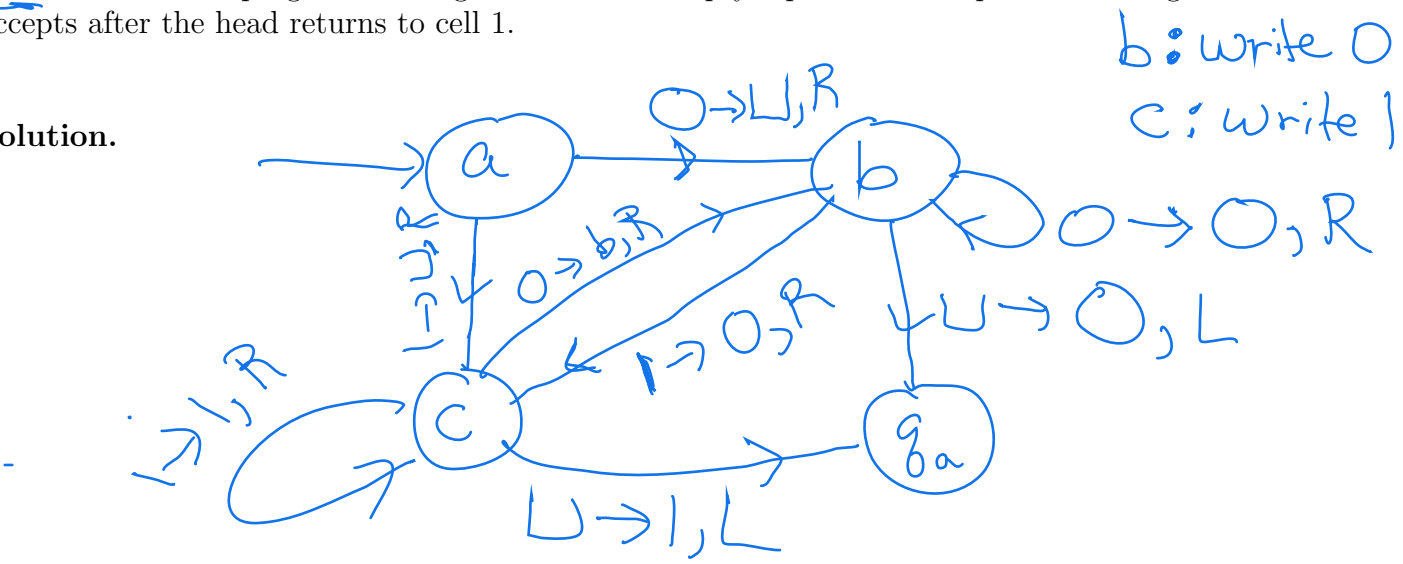


2 Writing Turing Machine Programs

"The ability to write a Turing machine program" means the ability to provide the δ-transition table for a Turing machine that performs some specified task. Similar to a DFA or NFA, a helpful first step is to first develop a state diagram, followed by translating it into a table. Furthermore, most TM programs rely on a handful of commonly used "subprograms" that peform basic tasks. For example, the Turing machine from Example 1 made use of the basic task of checking to see if two different words on the tape are equal, namely one word that was to the left of the pound symbol and the other word that was to the right of the pound. Developing programs for such basic tasks provides both the building blocks, experience, and confidence that needed to write more complex programs.

Example 5. Another basic TM programming task is that of right or left shifting a word on the tape. For example, if $w = 101$ is the input word, then, after a single right shift, the new memory word is $\sqcup 101$. Write a TM program that right shifts a nonempty input word one place to the right and then accepts after the head returns to cell 1.

Solution.



3 Church-Turing Thesis

Notice that, in theory, a human is capable of simulating a Turing machine. This is because, for any machine M and input w to M, a human is capable of maintaining a current configuration for each step of the computation of M on some input w by successively updating the configuration in accordance with the current state and the current symbol being read. Updating the configuration only requires the ability to look up information in a table, erase one symbol of the memory word, write a new symbol in its place, and move the head one place either left or right.

The observation that a human is capable of simulating a Turing machine and the fact that mathematicians, such as Alan Turing and Alonzo Church, discovered that seemingly disparate models of computation were yielding the same set of decidable problems and computable functions, led to the following famous informal thesis.

Church-Turing Thesis. Any problem that can be solved in theory by a human in a step-by-step manner using pencil and paper can be solved by a Turing machine, meaning that there is some Turing machine that, given an encoded instance of the problem, performs a finite number of steps before reaching a final configuration that encodes the solution to the problem.

Although the Church-Turing Thesis cannot be formally proved, evidence that supports it includes the fact that several different models of computation (e.g. Unlimited Register Machines, Turing machines, Church's Lambda Calculus, all procedural programming languages used in practice, to name a few) have all proven to be equivalent, in that they solve the same set of problems.

The thesis allows us to unleash our creative imaginations when devising an algorithm and, so long as we are able to describe how the algorithm works in a deterministic step-by-step manner, we may assume that it can be programmed by a Turing machine or any other equivalent model of computation.

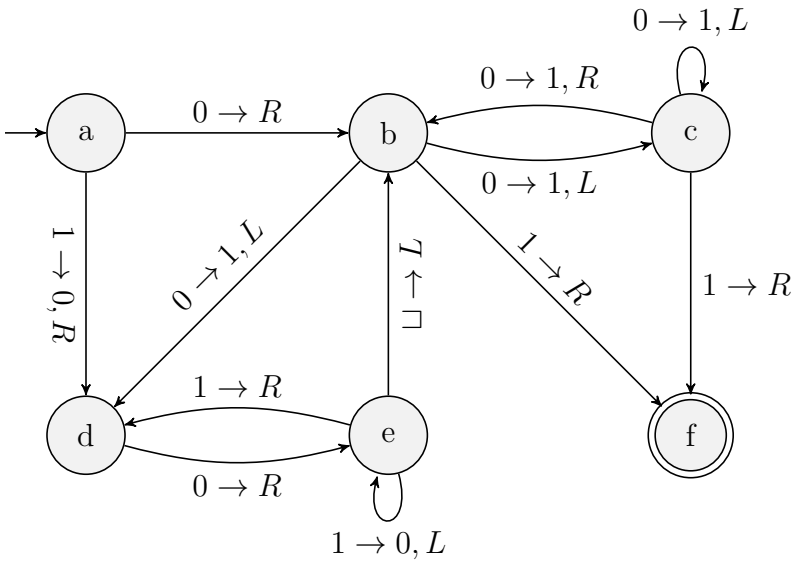
Nondeterministic Turing Machines

Definition 3. A **nondeterministic Turing machine (NTM)** N is one whose δ-transition function is now defined as

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}),$$

meaning that the δ-transition function outputs a subset of (next state, write symbol, direction)-triples. Thus, an NTM computation on some input w takes the form of a **computation tree** $T(N, w)$, where each branch of the tree represents a computation sequence, and input w is accepted so long as at least one of the branches of $T(N, w)$ is finite and ends in an accepting configuration.

For the NTM machine N with state diagram shown below, draw the computation tree $T(N, 10)$; i.e. the computation tree of N on input 10. Is this an accepting computation?



Solution.

Example 6. Provide a state diagram for a nondeterministic Turing machine M that starts with a blank input tape and proceeds to generate eight different configurations, with each configuration having the form $wq_3\sqcup$, where $w \in \{0,1\}^3$ and no two configurations have the same binary word.

NTM's and Complexity Class NP

Let N be an NTM and define function $s : \mathcal{N} \to \mathcal{N}$ as follows. For input $n \geq 0$, let $s(n)$ denote the longest branch of any computation tree $T(N, w)$, where $|w| = n$. For example, suppose the inputs to N are binary words and consider $n = 5$. Then there are $2^5 = 32$ different length-5 inputs to N and hence 32 different conputation trees for inputs of length 5. Moreover, $s(5) = L$, where L is the length of the longest branch of any of these trees. We call $s(n)$ the **(worst-case) time step function** for N.

An alternative definition of NP

Definition 4. Let A be some language over an alphabet Σ. Then A is a member of complexity class NP iff there is some NTM N for which $A = L(N)$ and N's time-step function $s(n)$ is bounded by some polynomial with respect to n. Thus, NP stands for those languages that can be recognized by a **N**ondeterministic Turing machine in a **P**olynomial number of steps.

The most "natural" NP-complete problem

The following theorem is left as an exercise.

Theorem 1. Consider the decision problem for which an instance consists of i) an NTM N, ii) an input word w to N, and iii) the word 1^m for some nonnegative integer m. Then this problem is NP-complete.

Cook's Theorem

Theorem 2. (Cook's Theorem) SAT is NP-complete.

Proof.

Let L be a decision problem in NP via verifier $v(x, y)$, where $v(x, y)$ is computable in $O(q(|x|))$ steps and variable y can be encoded using $O(q(|x|))$ bits, for some polynomial q. We describe how to compute a Boolean formula $F_x(y)$ whose size is bounded by a polynomial in $|x|$, and which is satisfiable iff x is a positive instance of L. Thus, $F_x(y)$ depends on x and its variables are the Boolean variables that encode y.

To accomplish this, we assume the Turing-machine model of computation. Let $M = (Q, \delta, \{0, 1\}, \Gamma, q_0, q_a)$ be a Turing machine that decides $v(x, y)$, where

1. Q is its set of machine states, including the respective initial and acccepting states q_0 and q_a (a rejecting state is implied in case $\delta(q, s)$ is undefined for some pair $(q, s) \in Q \times \Sigma$)
2. its transition function is
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\},$$
.

We may assume that x is encoded with m bits, y with n bits, $m + n + 1 \leq t$, where $t = O(q(m))$ is a bound on the number of configurations needed for the computation $M(x, y)$. Note also that the computation requires at most the first t tape cells. Thus, we may define a polynomial (with respect to t) number of Boolean variables that capture each configuration of the computation of $M(x, y)$. And for each of the t configurations, the variables must indicate

1. the current state,
2. the tape head location, and
3. the contents of each of the first t cells.

It then follows that we need at most $\lceil \log |Q| \rceil$ variables to describe M's state. Also, we may use t variables to encode the head location. Finally, we need at most $t\lceil \log |\Gamma| \rceil$ variables to encode the contents of each of the first t cells. Thus we need a total of $O(t^2) = O(q^2(m))$ variables to describe all of M's configurations. These variables are defined as follows. Note: for simplicity, instead of defining individual Boolean variables that encode each bit of the encoding of the machine state, we instead define a vector $\vec{s}$ of Boolean variables. We also define a vector of Boolean variables to represent the current symbol stored in a cell.

Defining the variables.

1. $\vec{s}_i$, $0 \leq i < t$, is a $\lceil \log |Q| \rceil$-bit vector of variables that encodes the state of the i th configuration of $M(x, y)$.

2. h_{ij} equals 1 iff, for the i th configuration, the head is located at cell j, $0 \leq i, j < t$.

3. $\vec{c}_{ij}$, $0 \leq i, j < t$, is a $\lceil \log |\Gamma| \rceil$-bit vector of variables that encodes the symbol stored in cell j in the i th configuration of $M(x, y)$.

Then the following equations completely define each of the Boolean variables. Reminder: we assume $|x| = m$ and $|y| = n$.

Base Cases.

1. $\vec{s}_0 = q_0$.
2. $h_{00} = 1$.
3. $\vec{c}_{0j} = x_j$, $0 \leq j < m$, encodes the jth bit of x.
4. $\vec{c}_{0m} = \#$.
5. $\vec{c}_{0(m+1+j)} = y_j$, $0 \leq j < n$, encodes the jth bit of y.
6. $\vec{c}_{0j} = \sqcup$, $m + n + 1 < j < t$.

In what follows, for simplicity we assume the head never moves (left) off the tape. We may also think of δ as the list of 5-tuples

$$(q_1, \alpha_1, q_1', \beta_1, d_1), \ldots, (q_r, \alpha_r, q_r', \beta_r, d_r),$$

where, for all $u = 1, \ldots, r$, $q_u, q_u' \in Q$, $\alpha_u, \beta_u \in \Gamma$, and $d_u \in \{L, R\}$.

Recursive Cases.

Next State For all $i = 1, \ldots, t-1$,

$$[(\vec{s}_{i-1} = q_1 \wedge h_{(i-1)1} \wedge \vec{c}_{(i-1)1} = \alpha_1 \wedge \vec{s}_i = q_1') \vee \cdots \vee (\vec{s}_{i-1} = q_r \wedge h_{(i-1)1} \wedge \vec{c}_{(i-1)1} = \alpha_r \wedge \vec{s}_i = q_r')]$$

$$\vee \cdots \vee [(\vec{s}_{i-1} = q_1 \wedge h_{(i-1)(t-1)} \wedge \vec{c}_{(i-1)(t-1)} = \alpha_1 \wedge \vec{s}_i = q_1') \vee \cdots \vee (\vec{s}_{i-1} = q_r \wedge h_{(i-1)(t-1)} \wedge \vec{c}_{(i-1)1} = \alpha_r \wedge \vec{s}_i = q_r')].$$

Cell Values For all $i = 1, \ldots, t-1$ and for all $j = 0, \ldots, t-1$,

$$(\vec{c}_{ij} = \vec{c}_{(i-1)j} \wedge \overline{h_{(i-1)j}}) \vee$$

$$h_{(i-1)j} \wedge [(\vec{s}_{i-1} = q_1 \wedge \vec{c}_{(i-1)j} = \alpha_1 \wedge \vec{c}_{ij} = \beta_1) \vee \cdots \vee (\vec{s}_{i-1} = q_r \wedge \vec{c}_{(i-1)j} = \alpha_r \wedge \vec{c}_{ij} = \beta_r)].$$

Head Location For all $i = 1, \ldots, t-1$, there exists a $j = 0, \ldots, t-1$, such that

$$h_{(i-1)j} \wedge [(\vec{s}_{i-1} = q_1 \wedge \vec{c}_{(i-1)j} = \alpha_1 \wedge ((h_{i(j-1)} \wedge d_1 = \mathrm{L}) \vee (h_{i(j+1)} \wedge d_1 = \mathrm{R})) \vee \cdots$$

$$\vee (\vec{s}_{i-1} = q_r \wedge \vec{c}_{(i-1)j} = \alpha_r \wedge ((h_{i(j-1)} \wedge d_r = \mathrm{L}) \vee (h_{i(j+1)} \wedge d_r = \mathrm{R}))].$$

Head in Single Location For all $i = 0, \ldots, t-1$ and for all $j, k = 0, \ldots, t-1$, $j \neq k$,

$$h_{ij} \rightarrow \overline{h_{ik}}.$$

Final State is Accepting

$$\vec{s}_{t-1} = q_a.$$

Finishing the Proof.

1. Boolean formula $F_x(y_0, \ldots, y_{n-1})$ is the conjunction of the total of

$$t(t+2) = \mathrm{O}(t^2) = \mathrm{O}(q^2(m))$$

number of Boolean formulas defined in both the base and recursive cases. Moreover, since each formula has size $\mathrm{O}(1)$, it follows that F may be constructed in a polynomial number of steps with respect to $|x| = m$.

2. By completely adhering to M's program from the initial state to the final state, we see that $F_x(y_0, \ldots, y_{n-1})$ is satisfiable iff there is some certificate y for which $v(x, y) = 1$, iff x is a positive instance of L.

3. Therefore, $L \leq_m^p \texttt{SAT}$. □

A Summary of Different Kinds of Turing Machines

Recognizer A Turing machine is a **recognizer** iff it halts on all inputs that it accepts. Note: every Turing machine that recognizes a language is by definition a recognizer.

Decider A Turing machine is a **decider** iff it halts on all inputs.

Multitape A **multitape Turing machine** has more than one tape along with a head for each tape. Moreover, its transition function simultaneously controls each of its heads. Indeed, if M has $k \geq 2$ tapes, then its transition function δ accepts $k+1$ inputs (one state and k read symbols) and provides a ($2k$-tuple output (one next state, $k-1$ write symbols, and k head directions). Note that one tape is designated as the input tape for which the input can be read. The input tape is a read-only tape.

Nondeterministic A **nondeterministic Turing machine (NTM)** is one whose δ-transition function is now defined as

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}),$$

meaning that, for a given configuration, there could be more than one next configuration.

Enumerator An **enumerator Turing machine** M starts with a blank tape, and has an extra write-only tape on which it is capable of writing a potentially infinite set L of words. In this case we say that L is **enumerated** by M.

Universal A Turing machine U is said to be **universal** iff for any input $\langle M, w \rangle$, where M is a Turing machine and w is some input word for M, then $U(w) = M(w)$. Note: here we are restricting M so that its input alphabet is the same as U's. Examples of universal Turing machines include a computer operating system and a human who is capable of following a Turing-machine computation using pencil and paper.

Transducer A Turing machine T is called a **transducer** if it serves the purpose of computing a function $f : \Sigma^* \to \Sigma^*$, where Σ is the machine's input alphabet. The main difference between a transducer and a decider is that, instead of having accept and reject states q_a and q_r, T has a single halt state q_h. Similar to a decider, on input $w \in \Sigma^*$, the computation begins with an initial configuration and produces a sequence of configurations by folllowing T's transition function. If the halt state is never reached, then $f(w)$ is undefined. Otherwise, $f(w)$ is defined by examining the final halting configuration, and setting $f(w)$ equal to the longest tape prefix in Σ^*. For example, if the final configuration has tape contents $01100010\sqcup\sqcup 01101\sqcup\cdots$, then $f(w) = 01100010$, since $\sqcup \notin \Sigma$. In other words, the output ends with the first instance of a blank cell.

Turing Machine Core Exercises

1. Provide the δ-transition table for the Turing machine whose state diagram was shown in Example 1.

2. Provide the δ-transition table for the Turing machine whose state diagram was shown in Example 4.

3. The tape of a Turing machine currently stores a nonempty binary word w and the head is reading one of the bits of w. Let b be the value of the bit being read. Provide a state diagram for the machine which, when followed by the machine, has the effect of appending to w two copies of b followed by moving the head back to its original location and accepting. For example, if 110010 is currently stored on the tape and the head is reading the second 1, then, after executing your program, the final configuration should be $1q_a1001011$, where q_a is your program's accepting state. Make sure to indicate which state of your program is the initial state. Hint: assume the tape alphabet is $\Gamma = \{0, 1, \sqcup, x\}$.

4. The tape of a Turing machine currently stores a nonempty binary word w and the head is reading one of the bits b of w. Provide a state diagram for the machine which, when followed by the machine, has the effect of printing a b at the beginning of w, followed by printing a $1 - b$ at the end of w and then accepting with the head reading the blank cell that follows the $1 - b$ printed bit. For example, if 110010 is currently stored on the tape and the head is reading the second 1, then, after executing your program, the final configuration should be $11100100q_a\sqcup$, where q_a is your program's accepting state. Make sure to indicate which state of your program is the initial state. Hint: assume the tape alphabet is $\Gamma = \{0, 1, \sqcup, x\}$.

5. The tape of a Turing machine currently stores a nonempty binary word w and the head is reading one of the bits b of w. Provide a state diagram which, when followed by the machine, has the effect of accepting iff the first bit of w matches with its last bit. For example the machine should accept 10011, but should reject 100110 . Finally, design your program so that it erases all of w before accepting (respectively, rejecting).

6. Provide the state diagram for a Turing machine that converts the binary input word $w = w_1 \cdots w_n$ to the word $w = w_1 w_1 \cdots w_n w_n$. For example, if $w = 10011$ then w is replaced by 1100001111. Furthermore, the output word should make use of the same tape cells as w as well as the $|w|$ tape cells that follow those cells.

7. Provide the state diagram for a Turing machine that counts down from the binary input word w when viewed as a nonnegative integer. For example, if $w = 101$, then the three tape cells that store w will together successively store 100, 011, 010, 001, and 000, in that order.

8. Repeat the previous exercise, but now, for each time the counter is decremented, the entire counter is shifted over by tape cell, with an 'x' written in all cells that were once used as part of the counter. For example, on input 101, the sequence of memory words should be 101, x100, xx011, xxx010, xxxx001, xxxxx000. Thus, the number of x's equals the amount decremented from the counter. Remember that the entire counter must be successively shifted one unit to the right.

9. Provide the state diagram for a Turing machine M that accepts all binary palindromes. Provide the sequence of configurations that comprises the computation of M on input 01110.

Additional Exercises

A. Provide the state diagram for a Turing machine that accepts all inputs of the form $x\#y$, where $x, y \in \{0,1\}^+$, $|x| = |y|$, and $x \geq y$ when x and y are viewed as binary numbers.

B. Provide the state diagram for a Turing machine that accepts the language

$$L = \{x = y + z | x, y, z \in \{0,1\}^+ \text{ and } x = y + z\}.$$

For example $101 = 11 + 10 \in L$ since

$$(5)_2 = (3)_2 + (2)_2.$$

C. Provide the state diagram for a Turing machine that accepts the language

$$L = \{x = y + z | x, y, z \in \{0,1\}^+ \text{ and } x = y - z\}.$$

For example $11 = 101 - 10 \in L$ since

$$(3)_2 = (5)_2 + (2)_2.$$

D. Provide the state diagram for a Turing machine that, on input $x \in \{0,1\}^+$, replaces x with x 1's. Here, we are thinking of x as a binary number. For example, 1001 would be replaced with 111111111, while 00 is replaced with λ.

E. For the Turing machine M with state diagram shown below, draw the computation tree $T(M, 001)$. Is this an accepting computation? Explain.