Turing Machines

Last Updated March 18th, 2024

1 Introduction

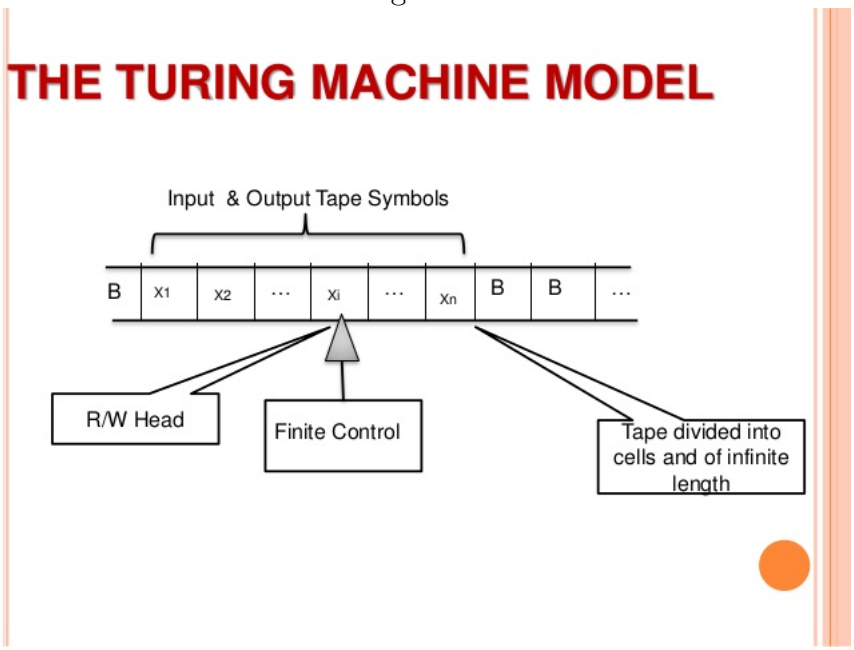
In this lecture we look at the Turing machine model of computation. A Turing machine is a kind of automaton that is more powerful than a pushdown automaton since the former is equipped with an infinite stack memory while the Turing machine has a memory consisting of an infinite array of cells that can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

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

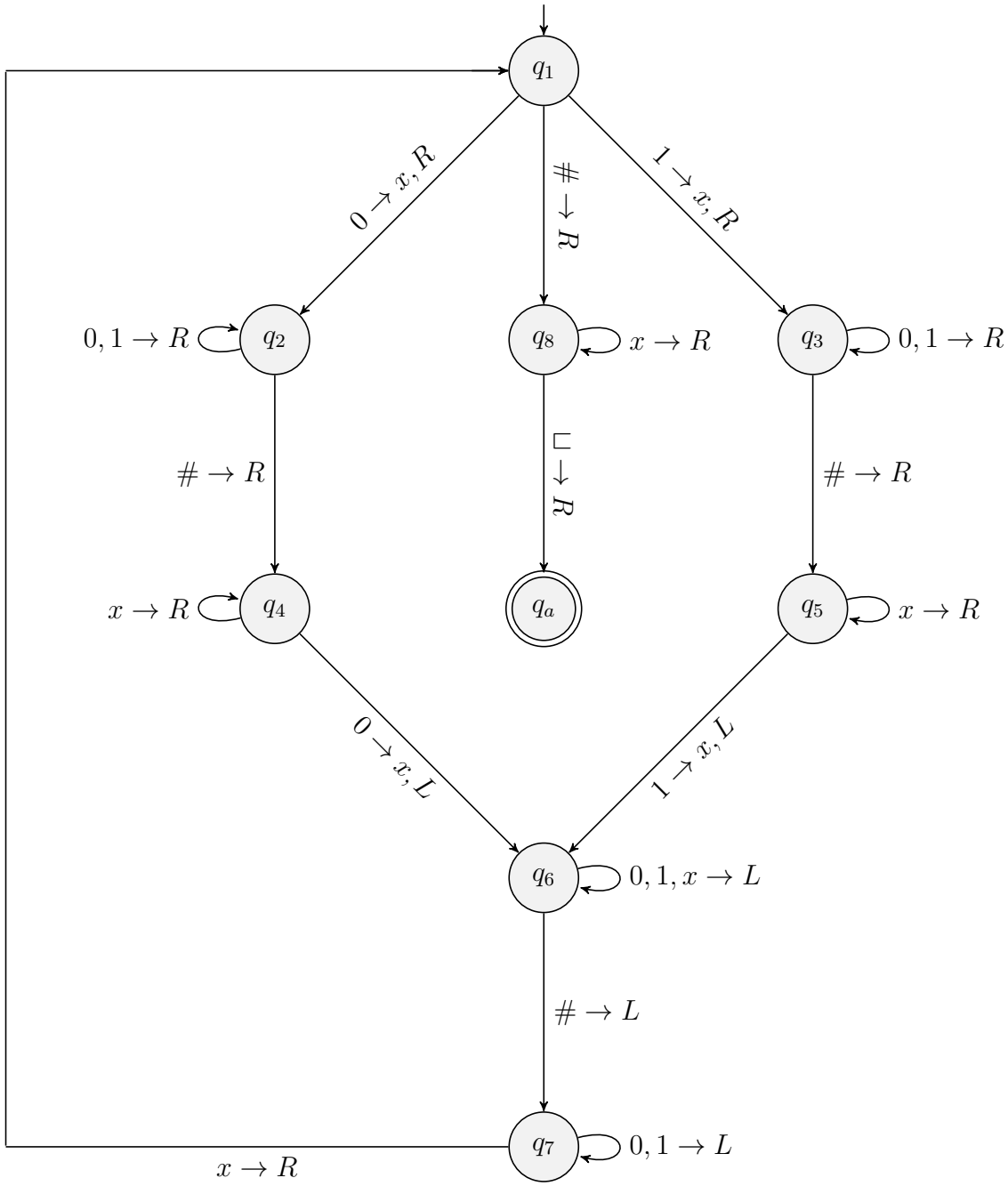
$q_0 \in Q$ the **initial state**

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 the 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 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 $w \in \{0,1\}^*$.



q_1**:** x-out the next bit of the left word, or move to q_8 if there are no remaining bits

q_2,q_3,q_6**:** look for the pound symbol

q_8**:** check if there is a remaining bit to the right of #, and accept iff there are no remaining bits

q_4,q_5**:** look for the next bit to the right of #, and continue if it matches the previous one x'd out to the left of #

q_7**:** search for the nearest x left 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.
3. The tape head never writes a blank symbol. Thus, a blank cell that borders a non-blank cell serves as a left or right delimiter for a word $w \in (\Gamma - \{\sqcup\})^*$, called the **tape word** which consists only of non-blank cells.
4. If the tape head is reading a blank and it moves right, then it must write a non-blank symbol. Thus, such a move increases the tape word's length by one.

Given Turing machine M, a **configuration** for M is a vector $\vec{\kappa}$ which represents the current state of the machine, along with its tape contents and tape-head location. Moreover, $\vec{\kappa}$ has the form $\vec{\kappa} = c_1 \cdots c_{j-1} q c_j \cdots c_k$, and is interpreted as follows.

- M is in state q.
- The tape head is reading cell c_j.
- The i th tape cell contains symbol $c_i \in \Gamma$, for all $1 \leq i \leq k$.
- Either c_k is the last cell of the tape that contains a non-blank symbol or $c_k = \sqcup$ the tape head is reading c_k.

Example 2. Given the two Turing-machine tapes below, provide configuration vectors for each tape. Assume in both cases the machine is in state q_3.

		↓					
⊔	0	1	⊔	⊔	1	⊔	⋯

				↓			
1	0	1	⊔	⊔	⊔	⊔	⋯

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 $\vec{\kappa} = c_1 \cdots c_{j-1} q c_j \cdots c_k$, 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 $\vec{\kappa}_0 = q_0 w_1 \cdots w_n$ is the initial configuration.

Recursive case Assume that $\vec{\kappa}_i = c_1 \cdots c_{j-1} q c_j \cdots c_k$ has been defined, $q \notin \{q_a, q_r\}$, and $\delta(q, c_j) = (\hat{q}, a, \mathrm{D})$.

Case 1: D=R. Then

$$\vec{\kappa}_{i+1} = c_1 \cdots c_{j-1} a \hat{q} c_{j+1} \cdots c_k$$

if $j < k$, and

$$\vec{\kappa}_{i+1} = c_1 \cdots c_{k-1} a \hat{q} \sqcup$$

if $j = k$

Case 2: D=L. Then

$$\vec{\kappa}_{i+1} = c_1 \cdots \hat{q} c_{j-1} a c_{j+1} \cdots c_k$$

if $j > 1$, and

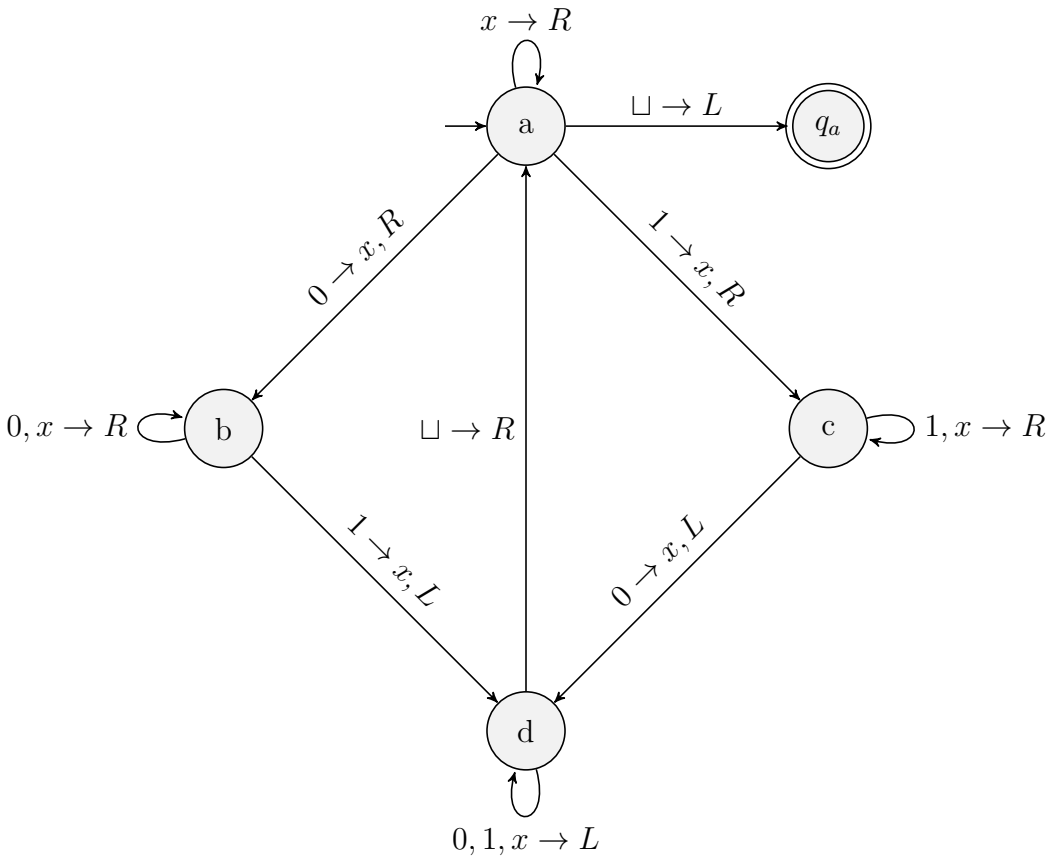
$$\vec{\kappa}_{i+1} = \hat{q} a c_2 \cdots c_k$$

if $j = 1$.

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 3. 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.



Special Kinds of Turing Machines

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

Recognizer A Turing machine is a **recognizer** iff it halts on all inputs that it accepts.

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 $k > 0$ tapes, then its transition function δ accepts $k + 1$ inputs (one state and k read symbols) and provides a $(2k + 1)$-tuple output (one next state, k write symbols, and k head directions).

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

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

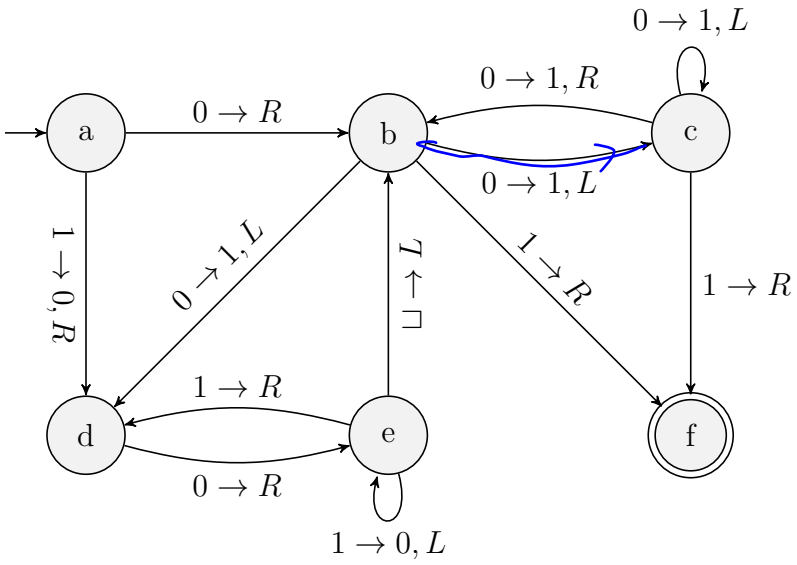
meaning that, for a given configuration, there could be more than one next configuration.

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␣␣01101␣$\cdots$, then $f(w) = 01100010$, since $\sqcup \notin \Sigma$. In other words, the output ends with the first instance of a blank cell.

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?

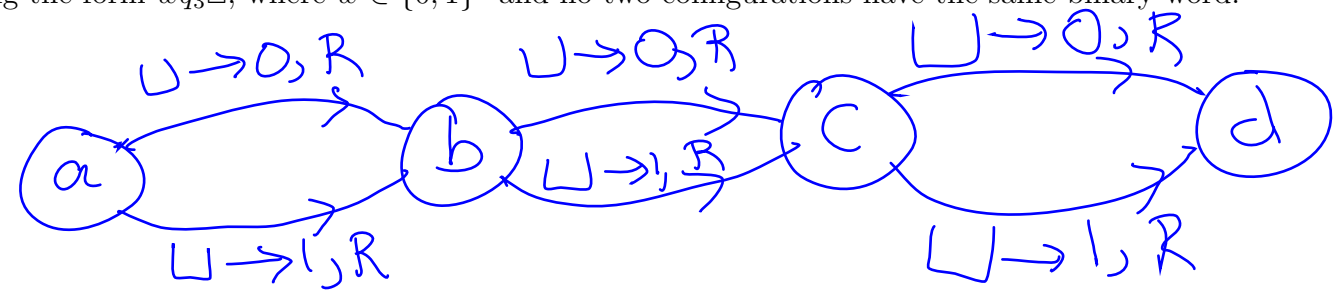


Solution.

for an NTM that uses $O(f(n))$ space

Stack size = $O(2^{c \cdot f(n)})$

Example 4. 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.



2 Cook's Theorem

We now provide an example of why Turing machines are considered the computing model of choice in the study of computational complexity theory.

Theorem 1. (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.

vector of Boolean variables

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

Σ

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

$m = |x|$

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.

$t \lceil \log |\Gamma| \rceil$

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 $t\lceil \log |\Sigma| \rceil$ 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. $\vec{c}_{0j} = x_j$, $0 \le j < m$, encodes the jth bit of x.
4. $\vec{c}_{0m} = \#$.
5. $\vec{c}_{0(m+1+j)} = y_j$, $0 \le j < n$, 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 $j, k = 0, \ldots, t-1$, $j \neq k$,

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

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, $L \leq_m^p \texttt{SAT}$. □

Exercises

1. 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.

2. 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.

3. 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

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

4. 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

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

5. 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.

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