LECTURE NOTES

CPSC 351 — Winter 2025

Theoretical Foundations of Computer Science II

Philipp Woelfel

Chapter 5: Turing Machines

Contents

5 Turing Machines **1**

5.1 Mathematical Definition of Turing Machines 2
5.2 The Language Recognized by a Turing Machines 4
5.3 Turing Machine Design 8
5.3.1 Memorization 8
5.3.2 Marking Symbols 9
5.4 Turing Machine Variants 10
5.4.1 Stay-Put Option 11
5.4.2 Multiple Tapes 12
5.5 Encoding Objects 13
5.6 Computing Functions with Turing Machines 15
5.7 The Universal Turing Machine 15
5.7.1 Encoding Turing Machines 15
5.7.2 Constructing the Universal Turing Machine 16
5.8 Exercises 17

5 Turing Machines

Deterministic and non-deterministic finite-state automata are very limited due to the finite space assumption. Many problems that we can solve with computer programs cannot be solved with finite space. As we have seen, it is impossible to determine with a DFA if a given string is of the form 0^n1^n, for any $n \in \mathbb{N}$.

A more powerful computational model is the *Turing machine*, named after its inventor, Alan Turing. Turing machines are powerful enough to solve every problem that can be solved by modern programming languages, such as Python or Java, assuming that there is no bound on the available amount of memory. (Strictly speaking, a computer with finite amount of memory is just a DFA Therefore, the DFA can be viewed as a hardware model, while the Turing machine is a model for software.)

A Turing machine has a *tape* for storing information. The tape is divided into an infinite number of *cells*, and each cell can store a symbol from some predefined finite *tape alphabet*, T. The machine has a *read/write head* (or just *head*, for short), which is always located on top of some tape cell, and can move across the tape. The control unit is similar to a DFA: It is always in one of a finite number of states from a set Q of states. The machine repeatedly executes the following steps:

1. The read/write head reads the symbol $x \in T$ from the cell it is located on top of.
2. Based on x and its current state q, the machine determines a new state $q' \in Q$, a new symbol $x' \in T$, and a *direction* $d \in \{L, R\}$.
3. The head then writes symbol x' into the cell at its location, overwriting the previous cell content, x. (It is possible that $x' = x$.)
4. The head moves one cell in the direction indicated by d (left or right).
5. The machine enters the new state q'.

Initially, all cells on the tape are blank, except for the *input*. Blank means that they store a special blank symbol, $\perp \in T$. The input is a string over an *input alphabet* Σ, where $\Sigma \subseteq T \setminus \{\perp\}$. That string is initially stored on a number of consecutive cells, and the read/write head is initially located at the leftmost input cell. This is called the *initial configuration*. A Turing machine also has a special state q_A, which is the *accepting state*. Whenever it reaches state q_A, it halts immediately.

Partial Functions

A *partial function* $f : A \to B$ is almost the same as a function with domain A and range B, except that there may be some element $x \in A$ for which $\delta(x)$ is not defined. For example, $f : \mathbb{Z} \to \mathbb{R}$, where $\delta(x) = \sqrt{x}$ is a partial function because $\delta(x)$ is not defined for any negative integer x.

5.1 Mathematical Definition of Turing Machines

A Turing machine is uniquely defined by the following components:

- A finite set Q of states.
- A finite *input alphabet* Σ, which does not contain the *blank symbol* $\perp$.
- A finite *tape alphabet* T, such that $\Sigma \cup \{\perp\} \subseteq T$.
- A *transition function* $\delta : Q \times T \to Q \times T \times \{R, L\}$, which can be a partial function (see the box on partial functions).
- An *initial state* $q_0 \in Q$.
- An *accepting state* $q_A \in Q$.

Thus, a Turing machine is defined by a 6-tuple, $(Q, \Sigma, T, \delta, q_0, q_A)$.

The transition function maps each pair (q, x), where q is a state and x is a symbol, to a new state q', a new symbol x', and a direction, $d \in \{L, R\}$. Thus, δ describes what happens when the machine reads symbol x while in state q: The head replaces x with x' at its current location and moves one step into direction d, and the machine enters state q'.

If a Turing machine ever enters a state q and reads a symbol x such that $\delta(q, x)$ is undefined, then it halts immediately, similar as it halts if it enters the accepting state.

The transition function, δ, can be specified in several ways. One way is to give it as a table, as for example in Figure 5.1. Another option is to list all tuples (q, x, q', x', d) such that $\delta(q, x) =$

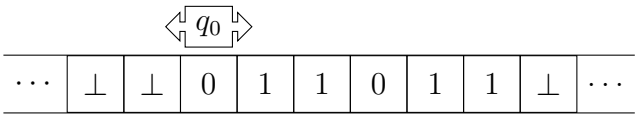
δ	0	1	$\perp$
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_3, \perp, L)$
q_1	$(q_0, 0, R)$	$(q_2, 1, L)$	$(q_3, \perp, L)$
q_2	—	$(q_0, 0, R)$	—
q_3	—	—	—

Figure 5.1: The transition function δ of a Turing machine.

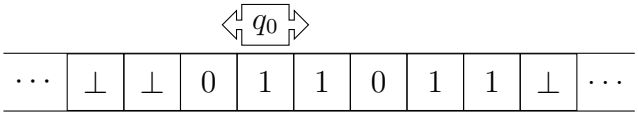
(q', x', d). For example, the transition function from Figure 5.1 can be given by:

$$(q_0, 0, q_0, 0, R), (q_0, 1, q_1, 1, R), (q_0, \bot, q_3, \bot, L),$$
$$(q_1, 0, q_0, 0, R), (q_1, 1, q_2, 0, L), (q_1, \bot, q_3, \bot, L), (q_2, 1, q_0, 0, R).$$

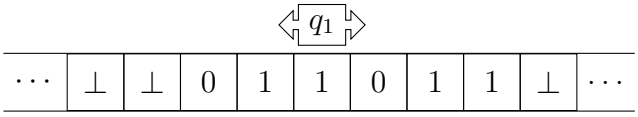
Suppose the initial configuration of a Turing machine with initial state q_0 and the above transition function is



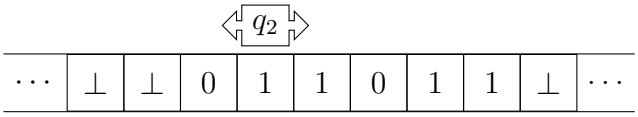
Thus, the input string is 011011, and the head is initially at the leftmost 0. In its first step, the head reads 0 from the tape. The transition function value for this situation is $\delta(q_0, 0) = (q_0, 0, R)$. Thus, the head replaces the leftmost 0 with another 0 (i.e., the cell content remains the same) and the head moves one position to the right. This yields the following configuration:



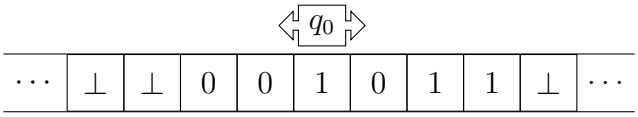
Now the Turing machine reads a 1 from the tape, while it is in state q_0. Since $\delta(q_0, 1) = (q_1, 1, R)$, it does not change the tape contents, moves the head one step to the right, and enters state q_1:



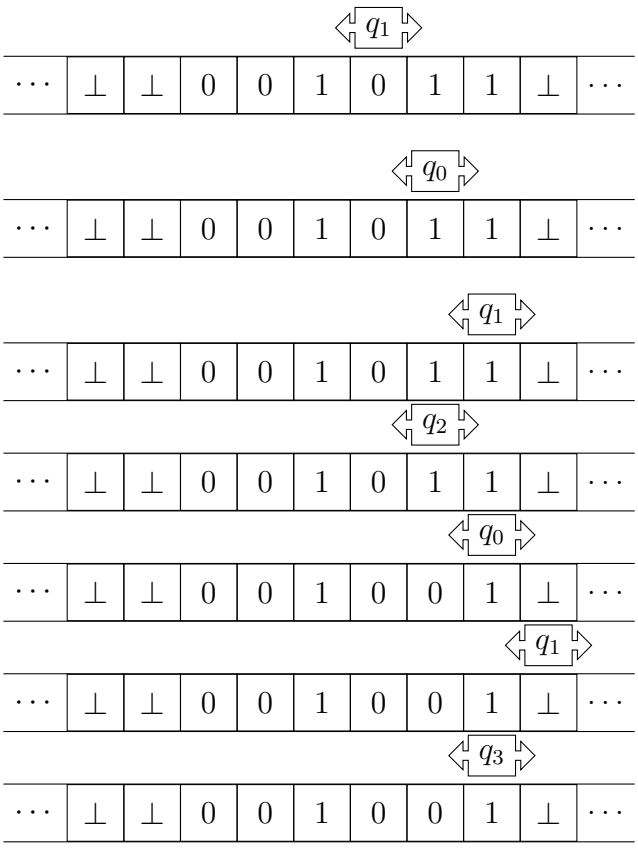
Now the Turing machine is in state q_1 and reads 1 from the tape. We have $\delta(q_1, 1) = (q_2, 1, L)$, so the machine does not change the symbol at it's head position, the head moves to the left, and the machine enters state q_2:



The machine is now in state q_2, and reads a 1. Since $\delta(q_2, 1) = (q_0, 0, R)$, it writes a 0 and moves the head to the right, resulting in



Continuing this way, we obtain the following sequence of configurations:



At this point, the Turing machine has entered state q_3 and its head is positioned over a cell with symbol 1. Since $\delta(q_3, 0)$ is not defined, the Turing machine halts.

5.2 The Language Recognized by a Turing Machines

A Turing machine $(Q, \Sigma, T, \delta, q_0, q_A)$ *accepts* a string $x \in \Sigma^*$, if, starting from the initial configuration for input x, it eventually reaches the accepting state q_A. Unlike in the case of finite automata, a Turing machine may not halt at all for a given input. In that case, it also does not accept the input. If a Turing machine halts without accepting an input (i.e., it enters a state q and reads a symbol s such that $\delta(q, s)$ is undefined), then it *rejects* the input.

Definition 5.1. *A Turing machine M with input alphabet Σ* recognizes *the language*

$$L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}.$$

Example 5.2. Construct a Turing machine that recognizes the set of bit strings of length at least two, which have a 1 as their second bit.

The Turing machine is the 6-tuple $(Q, \Sigma, T, \delta, q_0, q_2)$, where

$$Q = \{q_0, q_1, q_2\}, \quad \Sigma = \{0, 1\}, \quad T = \Sigma \cup \{\bot\},$$

and δ is given by the following table:

δ	0	1	$\bot$
q_0	$(q_1, 0, R)$	$(q_1, 1, R)$	—
q_1	—	$(q_2, 1, R)$	—
q_2	—	—	—

The Turing machine is initially in state q_0. The transition $\delta(q_0, \bot)$ is not defined; thus, if the first symbol is blank (i.e., the input is λ), then the Turing machine rejects immediately. Otherwise, the head moves one position to the right, and enters state q_1. That state indicates that the head is now positioned on top of the second input symbol. If in state q_1 the Turing machine reads a 1, it enters the accepting state q_2, and thus accepts the input. For any other case, the transition function is not defined, and thus the machine rejects. ◀

Example 5.3. Construct a Turing machine for the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$.

The machine will proceed by repeatedly executing the following five phases:

1. The head is positioned at the leftmost non-blank symbol on the tape, if such a symbol exists. If the symbol at the head's position is blank, then accept (initially, this is only the case if the input is λ). If the symbol is 1, then reject. Otherwise, it is 0. Overwrite it with a blank.
2. Sweep to the right, searching for the first blank symbol, then move one step to the left.
3. The head is positioned at the rightmost non-blank symbol on the tape, if such a symbol exists. If the symbol at the head's position is blank or 0, then reject. Otherwise, it is 1. Overwrite it with a blank.
4. Sweep to the left, searching for the first blank symbol, then move one step to the right.

We will implement this by using five states:

- q_{LE}: The machine should be in this state during Phase 1, i.e., when the head is positioned on the leftmost symbol of the remaining substring of the input.
- q_{SR}: This is used for Phase 2, i.e., when the machine searches for the rightmost non-blank symbol.
- q_{RE}: The machine should be in this state during Phase 3, i.e., when the head is positioned on the rightmost non-blank symbol.

δ	0	1	$\perp$
q_{LE}	$(q_{SR}, \perp, R)$	—	$(q_A, \perp, R)$
q_{SR}	$(q_{SR}, 0, R)$	$(q_{SR}, 1, R)$	$(q_{RE}, \perp, L)$
q_{RE}	—	$(q_{SL}, \perp, L)$	—
q_{SL}	$(q_{SL}, 0, L)$	$(q_{SL}, 1, L)$	$(q_{LE}, \perp, R)$
q_A	—	—	—

Figure 5.2: Transition function of a Turing machine recognizing $\{0^n1^n \mid n \in \mathbb{N}_0\}$.

- q_{SL}: In Phase 4, the machine is in this state, while searching for the leftmost non-blank symbol.
- q_A: This is the accepting state.

The Turing machine is the 6-tuple $(Q, \Sigma, T, \delta, q_{LE}, q_A)$, where Q contains the five states above, $\Sigma = \{0, 1\}$, $T = \{0, 1, \perp\}$, and δ is the transition function depicted in Figure 5.2. Thus, initially the machine is in state q_{LE}. It is not hard to see, that this realizes the phases described above.

To prove that the machine is correct, we have to show that $x \in L$ if and only if the Turing machine accepts x.

We will show that if $x \in L$, then the Turing machine accepts x. It is also necessary to show that if the machine accepts an input x, then $x \in L$. We leave that part of the proof as an exercise.

First assume that $x \in L$. Hence, x is of the form 0^n1^n. We can prove by induction on n that the Turing machine accepts x.

The base case is for $n = 0$. In that case, $x = \lambda$. Hence, in the initial configuration the head is on top of a blank symbol and the machine is in its initial state q_{LE}. Since $\delta(q_{LE}, \perp) = (q_A, \perp, R)$, the Turing machine reaches the accepting state and accepts the input.

Now let $n > 0$, and suppose that the Turing machine accepts the input $0^{n-1}1^{n-1}$. Consider the input $x = 0^n1^n$. Initially, the machine is in state q_{LE} and on top of the leftmost 0 of the input. The configuration is shown in (1) of Figure 5.3. All following numbered references in parentheses refer to Figure 5.3.

The machine now reads a 0, and thus enters state q_{SR}, overwrites the leftmost 0 with a $\perp$, and moves one step to the right. Thus, the tape content is now $0^{n-1}1^n$, and the machine is on top of the first 0 in state q_{SR}—see (2). Since $\delta(q_{SR}, i) = (q_{SR}, i, R)$ for each $i \in \{0, 1\}$, the machine's head will now sweep to the right, not changing any of the tape symbols or its state, until it finds the first blank. Thus, eventually its head is on top of the first blank following the input, and the state is q_{SR}. This is depicted in (3).

Since $\delta(q_{SR}, \perp) = (q_{RE}, \perp, L)$, the head now moves one step to the left, and enters state q_{RE}. Thus, the head is now on top of the rightmost 1 of the remaining string $0^{n-1}1^n$ on the tape—see (4). As $\delta(q_{RE}, 1) = (q_{SL}, \perp, L)$, the machine overwrites the rightmost 1 with a blank, enters

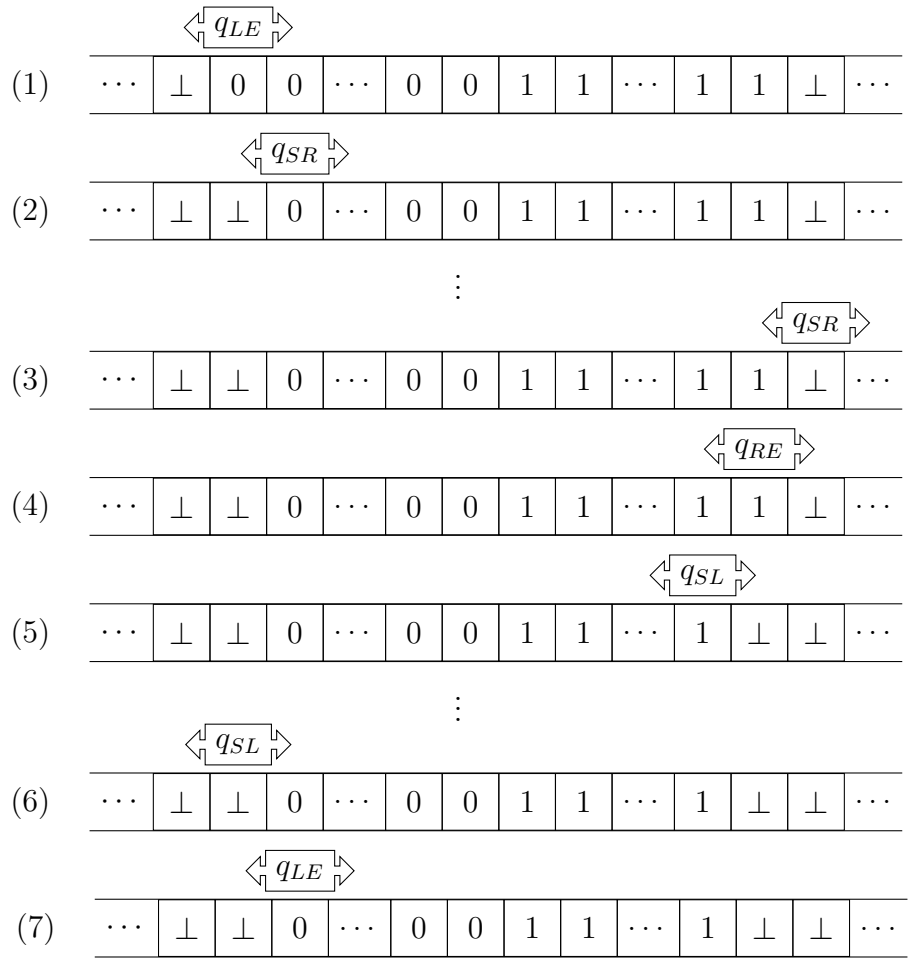


Figure 5.3: Configurations of the Turing machine when accepting 0^n1^n.

state q_{SL}, and the head moves one step to the left. Now the remaining non-blank string on the tape is $0^{n-1}1^{n-1}$. The configuration is shown in (5).

In state q_{SL}, the head sweeps to the left until it finds the first blank (just left of the first 0 of $0^{n-1}1^{n-1}$), resulting in the situation depicted in (6). When that happens, because $\delta(q_{SL}, \bot) = (q_{LE}, \bot, R)$, the head simply moves one step to the right, and the state changes to q_{LE}.

Now the Turing machine is in state q_{LE}, and its head is on the leftmost position of the remaining string $0^{n-1}1^{n-1}$, as depicted in (7). Thus, the machine is in the same configuration, as the initial one for the input $0^{n-1}1^{n-1}$. By the inductive hypothesis, the machine accepts $0^{n-1}1^{n-1}$. Hence, it accepts.

This proves that if $x \in L$, then the Turing machine accepts the input x. It remains to show that if the Turing machine accepts x, then $x \in L$. We leave this as an exercise. ◀

5.3 Turing Machine Design

In the previous section, we have given a *formal description*, of a Turing machine, by specifying the corresponding 6-tuple. It is not always feasible to do that. In particular, once we understand how Turing machines work, we will often only give *high level* descriptions. This means, we will just on a general level explain how a Turing machine works, but we will not specify, for example, all state transitions.

In this section, we will present some techniques that Turing machines can use, and that help with Turing machine design. We will illustrate these techniques through examples. For the first example, we will provide a formal Turing machine description, and in the second one, a high level one.

5.3.1 Memorization

The Turing machine can use its state to memorize something about what it has learned from processing the input. Consider for example the following simple task: Initially, there is only an input $x =\in \{0,1\}^*$ on the tape. The goal is to insert the symbol # before the first 1. Thus, if the input is $x = 010110$, then the Turing machine should replace this with 0#10110.

We design a machine, which first finds the first 1. We use the state $q_{\#}$, to memorize that it is currently searching for the first 1 (this should also be the initial state). Once it has found the first 1, the machine shifts each symbol by one position to the right, and writes # into the free position. To do that, whenever it reads a symbol, it memorizes that symbol with a special state: q_0 for 0 and q_1 for 1. In each step, it reads a symbol, writes the previously memorized symbol to the tape, memorizes the new symbol, and moves the head one step to the right. Finally, once the machine reads $\bot$, it replaces that symbol with the most recently memorized symbol, and

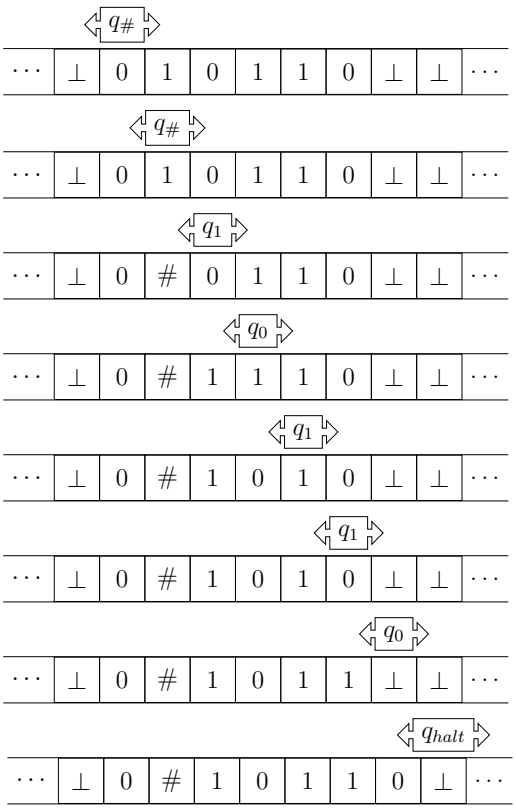


Figure 5.4: Execution of a Turing machine inserting # before the first 1.

enters a state q_{halt} that forces it to halt (because no transitions are defined). The corresponding transition function is as follows:

δ	0	1	#	$\perp$
$q_\#$	$(q_\#, 0, R)$	$(q_1, \#, R)$	—	—
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$	—	$(q_{halt}, 0, R)$
q_1	$(q_0, 1, R)$	$(q_1, 1, R)$	—	$(q_{halt}, 1, R)$
q_{halt}	—	—	—	—

Consider, for example, the input 010110. The execution of the Turing machine is depicted in Figure 5.4.

5.3.2 Marking Symbols

Another useful technique is to mark symbols. We can do that, by extending the tape alphabet to have a "marked" version for each regular symbol.

For example, assume that we want to design a Turing machine, that takes an input $x \in \{0,1\}^*$, and replaces it with xx on the tape. We will use the "marked" symbols $\widetilde{0}$ and $\widetilde{1}$ to mark some positions of the input. Thus, the tape alphabet of our machine is $T = \left\{0, 1, \bot, \widetilde{0}, \widetilde{1}\right\}$.

Our Turing machine will work as follows: Recall that initially, the head is positioned on top of the leftmost input symbol. So if the input is $x = x_1 \dots x_n$, then the head is on top of x_1.

In the first phase, the machine will sweep from left to right across the input, replacing each input symbol with its marked version. Then the head will move back to the first symbol (by sweeping to the left until it finds a blank, and then going one step to the right). Thus, after the sweep the tape contents is now $\widetilde{x_1} \dots \widetilde{x_n}$, where $\widetilde{x_i}$ is the marked version of x_i. The head is on top of $\widetilde{x_1}$.

After that, the Turing machine performs the following steps.

1. Sweep left until the first blank is found.
2. Sweep right until either a blank symbol or a marked symbol is found. In case of a blank, stop. Otherwise, suppose the symbol we found is $\widetilde{a}$, where $a \in \{0,1\}$.
3. Replace $\widetilde{a}$ with its unmarked version, a.
4. Sweep to the right until the first blank is found.
5. Replace the blank with a.

This is repeated, until the machine stops in the second step.

With each iteration of those steps, the machine replaces the leftmost marked symbol with its unmarked version, and appends it to the rightmost symbol of the string on the tape.

We leave it as an exercise to give a formal description of the Turing machine.

5.4 Turing Machine Variants

We will now discuss some variants of Turing machines, which make it easier to recognize languages. For each variant, we will then show that it can be simulated by a regular Turing machine. Thus, the variants are not more powerful than the regular Turing machine model.

5.4.1 Stay-Put Option

Note

This section was not taught in Winter 2025.

A regular Turing machine must in each step move the head either left or right. For a Turing machine with stay-put option, this is not the case: In a step, the head may stay in place. On a formal level, the only difference is that a Turing machine with stay-put option has a transition function of the form

$$\delta : Q \times T \to Q \times T \times \{L, R, S\},$$

where Q is the set of states and T the tape alphabet. Thus, it is possible that $\delta(q, a) = (q', a', S)$, which means that if the Turing machine reads a in state q, it will enter state q', write a', and the head will not move.

Suppose we have a Turing machine $M = (Q, \Sigma, T, \delta, q_0, A)$ with stay-put option. We can easily simulate M with a regular Turing machine M. The idea is that whenever M decides to not move its head, M' memorizes the state M is in, moves the head one step to the right, and then one step back to the left.

For that, we use the state set $Q' = Q \times \{0, 1\}$. Thus, each state of M' is of the form $(q, 0)$ or $(q, 1)$, where q is a state of M. A state $(q, 0)$ indicates that M would be in state q with the head at the same position as the head of M'. But M' is in state $(q, 1)$, whenever M is in state q, but the head of M is one cell further to the left than the head of M' (because M' just moved its head to the right, whereas M's head stayed put).

Thus, if δ' is the transition function of M', then

$$\delta'\big((q, 1), a\big) = \big((q, 0), a, L\big) \text{ for each } q \in A \text{ and } a \in T. \tag{5.1}$$

This ensures that if the head of M' is one cell too far to the right (as indicated by the 1 in state $(q, 1)$), then the head of M' will simply move one step to the left. Moreover, if M has a transition $\delta(q, a) = (q', a', d)$ defined, where $d \in \{L, R, S\}$, then M' allows the transition

$$\delta'\big((q, 0), a\big) = \begin{cases} \big((q', 0), a', d\big) & \text{if } d \in \{L, R\}, \text{ and} \\ \big((q', 1), a', R\big) & \text{if } d = S. \end{cases}$$

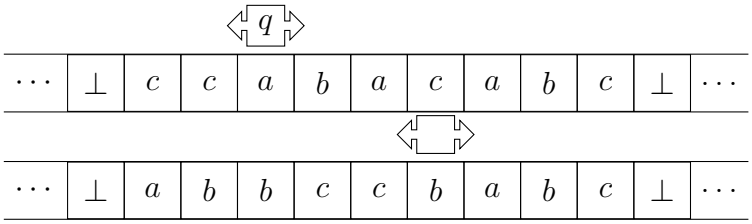
This means that if M enters state q, writes a', and its head moves left or right, then M' simulates this by entering state $(q', 0)$, also writing a', and moving the head in the same direction as M. But if M's head stays put, then M' transitions into state $(q', 1)$, writes a', and moves the head to the right. In the following step, by eq. (5.1), M' will move the head back to the left, and enter state $(q', 0)$.

5.4.2 Multiple Tapes

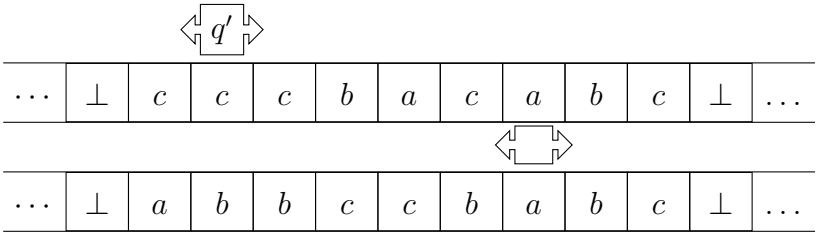
A *k-tape* Turing machine has k tapes instead of one. Each of the tapes has its own read/write head. In each step, each head reads the symbol at its cell position from the tape. Then, based on the Turing machine state, and on all k symbols read, the Turing machine now enters a new state, all heads write a new symbol, and then each head moves independently in one direction (left or right). Thus, the transition function is now

$$\delta : Q \times T^k \to Q \times (T \times \{L, R\})^k.$$

Consider, for example, a Turing machine with 2 tapes. Suppose the first Turing machine head is on top of a cell with symbol a, and the second head on top of a cell with symbol b. Let q be the state of the machine. The situation is depicted below:

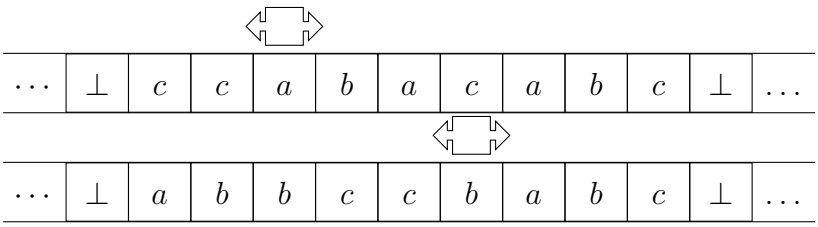


If $\delta(q, a, b) = \big(q', (c, L), (b, R)\big)$, then this means that the machine enters state q', the first head overwrites the a with a c and moves left, and the second head moves right, not changing the cell at its position. The result is:

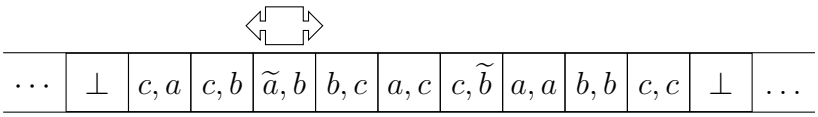


For any $k \in \mathbb{N}$, $k > 1$, we can simulate a k-tape Turing machine M with a regular Turing machine R. To that end, we encode the contents of the k tapes of M on R's single tape: If at some position, the i-th tape of M stores the symbol a_i, $i \in \{1, \ldots, k\}$, then at the same position R's tape stores the k-tuple $(a_1, \ldots, a_k)$. R also needs to keep track of M's k head positions. It does so by marking the corresponding symbols: Suppose the j-th head of M is in position i, and the tape cell at that position stores the value a_j. Then R represents that situation by marking symbol a_j in the tuple $(a_1, \ldots, a_k)$ that it stores in position i.

For example, suppose our 2-tape machine M is in the following configuration:



Then the single tape of R should look like this:



In addition, R always memorizes M's state. Then R can simulate a single step of M as follows: Assume the head of R is on top of the leftmost marked symbol, i.e., in the same position as the leftmost head of M (as in the picture above). R memorizes the marked symbol and the tape of M that contains the symbol (in the example above, it would memorize that the first head of M is on top of symbol a). Then R's head sweeps towards the right, until it finds the second marked symbol. Once R has found the second marked symbol, it knows M's state and the symbols at both head positions of M. Thus, R can compute the transition that M applies, i.e., the symbols M writes, the head movements, and M's new state. Using another sweep, R can update its tape accordingly.

By repeatedly simulating every single step of M, R can simulate the entire execution of M on a given input.

5.5 Encoding Objects

We are interested in Turing machines performing computations on objects, such as numbers, sets, graphs, or other Turing machines. Any object we typically deal with can be encoded by a bit string, or a string from some larger finite alphabet. We will use angled brackets to indicate the encoding of an object over some finite alphabet Σ (which is defined by the context): If O is an object, then $\langle O \rangle$ denotes the encoding of O.

For example, a non-negative integer $x \in \mathbb{N}_0$ can be encoded using a bit-string of x 0s

$$\langle x \rangle = \underbrace{0 \dots 0}_{x} = 0^x.$$

This is called the *unary* encoding. Obviously, this is not a very (space) efficient encoding, and for most applications binary encoding is more suitable.

A non-negative integer $x \in \mathbb{N}_0$ has the binary encoding $\langle x \rangle_2 = b_{n-1} \dots b_0 \in \{0,1\}^n$,

$$b_0 \cdot 2^0 + b_1 \cdot 2^1 \cdots + b_{n-1} 2^{n-1} = x.$$

Note that this encoding is not unique, because adding a string of 0s as a prefix does not change the value of the encoded integer. However, we can easily make it unique by requiring that there are no leading 0s. (Note that then the integer 0 is then encoded by the empty string λ, i.e., $\langle 0 \rangle_2 = \lambda$.)

We can encode multiple objects by encoding each individual object separately, and then appending the individual codes separated by a special symbol. For example, a set $\{x_1, \dots, x_k\}$ of non-negative integers can be encoded by a string over $\{0, 1, \#\}$, by using the binary encoding for each integer x_i, and using $\#$ to separate integers:

$$\langle x_1, \dots, x_k \rangle = \langle x_1 \rangle \# \langle x_2 \rangle \# \dots \# \langle x_k \rangle .$$

For example, the set $\{1, 3, 4, 11\}$ could be encoded as follows:

$$1\#11\#100\#1011.$$

Observe that any code using three or more symbols can easily be changed to a binary code, i.e., a code using only two symbols. For example, consider a code that uses the symbols $\{0, 1, \#\}$, as above. In an encoding of an object, we can replace each 0 with 00, each 1 with 11, and each $\#$ with 01. For example, the encoding of the two sets $\{1, 3\}$ and $\{2, 4\}$ from above would become

$$\langle \{1,3,4,9\} \rangle = 11\,01\,11\,11\,01\,11\,00\,00\,01\,11\,00\,00\,11.$$

Often we want to encode more complicated objects. For example, we may want to encode a directed graph $G = (V, E)$, where V is the set of vertices and E the set of edges. If we don't care about the vertex labels (which is usually the case), then we may assume that $V = \{0, 1, \dots, k-1\}$ for some $k \in \mathbb{N}_0$. Thus, we can encode V as $\langle k \rangle_2$. Each edge $e = (u, v) \in E$ is then a pair of integers $u, v \in \{0, \dots, k-1\}$, and we can encode it as $\langle u \rangle_2 \# \langle v \rangle_2$. To encode the entire graph, we simply write down the code of V followed by $\#$ and then the code for E.

For example, the directed triangle $0 \to 1 \to 2 \to 0$ would be encoded as

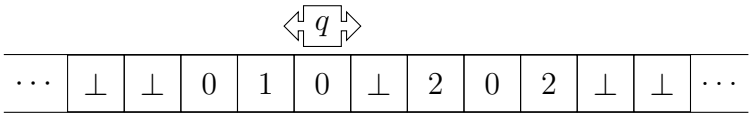
$$\langle 3 \rangle_2 \# \langle 0 \rangle_2 \# \langle 1 \rangle_2 \# \langle 1 \rangle_2 \# \langle 2 \rangle_2 \# \langle 2 \rangle_2 \# \langle 0 \rangle_2 = 11\#\#1\#1\#10\#10\#.$$

(Recall that the empty string encodes 0, i.e., $\langle 0 \rangle_2 = \lambda$.)

5.6 Computing Functions with Turing Machines

Consider a Turing machine M with input alphabet Σ and tape alphabet T. Then M computes a partially defined function $f : \Sigma^* \to T^*$. If M does not halt for some input $x \in \Sigma^*$, then $f(x)$ is not defined. But if M halts, then $f(x)$ is the contents of the tape between the leftmost non-blank symbol and the rightmost non-blank symbol.

Suppose, for example, a Turing machine halts for some input x in the following configuration:



Then the function f computed by that machine satisfies $f(x) = 010\bot202$.

Example 5.4. Construct a Turing machine that computes the partially defined function $f : \{0,1\}^* \to \{0,1\}^*$, where for any two non-negative integers $f(\langle x\rangle\,\langle y\rangle) = f(\langle x+y\rangle)$. Assume the unary encoding of an integer x, where $\langle x\rangle = 0^x1$.

Given an input 0^x10^y1, the Turing machine can simply shift the first x 0s by one position to the right, as in Section 5.3.1. Then it halts, and thus its output is $0^{x+y}1$. ◀

5.7 The Universal Turing Machine

We will now discuss a Turing machine that can simulate other Turing machines. This *Universal Turing machine* takes as input a description of another Turing machine, say M, as well as an input w, and then it simulates M on w. Thus, the Universal Turing machine is similar to a Python interpreter written in Python, which takes as input a Python program and then executes it.

5.7.1 Encoding Turing Machines

There are many ways of encoding Turing machines using a finite alphabet, but for the sake of concreteness, we will describe one particular encoding. We will use the alphabet $\Sigma = \{0,1\}$.

Recall that a Turing machine is a 6-tuple $(Q, \Sigma, T, \delta, q_0, q_A)$. We can assume that $Q = \{0,\ldots,q\}$ for some integer q, $\Sigma = \{0,\ldots,s\}$ for some integer s, and $T = \{0,\ldots,t\}$ for some integer $t > s$. The blank symbol is t, and thus it is not in Σ as required. Hence, to encode Q, Σ, and T, we just need to encode the integers q, s, and t.

The transition function, δ, is a list of quintuples: The transition $\delta(q, a) = (q', a', d)$ for $q, q' \in Q$, $a, a' \in T$ and $d \in \{L, R\}$, corresponds to a quintuple (q, a, q', a', d). For d we just use the numbers 0 instead of L and 1 instead of R. Then each element of the quintuple (q, a, q', a', d) is a number. Thus, the entire transition function can be encoded by a sequence of numbers. Finally, the starting state, $q_0 \in Q$ is a number in $\{0, \ldots, q\}$, and the set of accepting states is also just a set of numbers. Thus, we can encode each component of the Turing machine by encoding a number or a sequence of numbers. We can separate the codes for each component with the symbol #. Finally, replacing each 0 with 00, each 1 with 11, and each # with 01 (as discussed in Section 5.5), we obtain an encoding of the Turing machine that uses the alphabet $\{0, 1\}$.

5.7.2 Constructing the Universal Turing Machine

Let Σ be some finite alphabet of size at least 2, which does not contain #. The Universal Turing machine, U, takes as input a string $\langle M \rangle \# w$, where $\langle M \rangle \in \Sigma^*$ is the encoding of a (single tape) Turing machine M with input alphabet Σ, and $w \in \Sigma^*$.

We assume that U has three tapes (we know from Section 5.4.2, that we can then construct an equivalent single tape machine from U). Now, U simulates M on the input w as follows: Recall that the input to U is $\langle M \rangle \# w$. First, U copies w to the second tape. Then it moves the head of the second tape to the left end of w. Finally, U searches through the description of M for the code of M's initial state. It copies that to the third tape.

From now on, U's second tape will always look like M's tape in the simulated execution. Also, U's second head position will always be in the same position as M's head in the simulated execution. On the third head, U will store the state that M is in.

To simulate one step of M, U's first head searches for the description of the transition that M would apply. To do that, U can find M's simulates state on the third tape, and the symbol that M would read on the second tape. Once U has found the description of M's next transition, it can apply it by updating the second and third tape accordingly. I.e., the second head writes what M would write and moves in the same direction as M's head would, and the third head writes M's new state onto the tape. This way, U can simulate M step by step, until either M accepts or rejects (and then U can accept or reject, too).

5.8 Exercises

Turing machines

5.1 Let M be a Turing machine $(Q, \Sigma, T, \delta, q_0, q_2)$, where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $T = \{0, 1, \bot\}$, and δ is defined by the following table:

δ	0	1	$\bot$
q_0	$(q_1, 1, R)$	$(q_1, 0, R)$	$(q_1, 0, R)$
q_1	$(q_2, 1, L)$	$(q_1, 0, R)$	$(q_2, 0, L)$
q_2	—	—	—

For each of the following inputs, determine the final tape contents (when the Turing machine halts):

(a) 0011

(b) 101

(c) 11.

Turing machines

5.2 Let M be a Turing machine $(Q, \Sigma, T, \delta, q_0, q_2)$, where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $T = \{0, 1, \bot\}$, and δ is defined by the following table:

δ	0	1	$\bot$
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$	$(q_2, \bot, R)$
q_1	$(q_1, 0, R)$	$(q_0, 1, R)$	$(q_2, \bot, R)$
q_2	—	—	—

Describe the behaviour of M in case the input is

(a) 1^k for some $k \in \mathbb{N}$;

(b) an arbitrary string in $\{0, 1\}^*$.

Turing machines

5.3 Construct a Turing machine (by specifying the corresponding 6-tuple) with tape alphabet $T = \{0, 1, \bot\}$, which for an input $x \in \{0, 1\}^*$

(a) replaces the first 1 in x with a 0 and does not change any other tape symbols;

(b) appends a 1 to the end of x and does not change any other tape symbols;

(c) replaces the first occurrence of 01 in x with 10 and does not change any other tape symbols;

(d) halts in an accepting state if and only if x contains the sub-string 010.

Turing machines

5.4 For each of the following languages construct a Turing machine that recognizes it. Give high level and formal descriptions (in form of 6-tuples).

(a) $L_1 = \{x \in \{0,1\}^* \mid x \text{ contains an even number of 0s}\}$;

(b) $L_2 = \{x \in \{a,b\}^* \mid x \text{ contains equally many } a\text{'s and } b\text{'s}\}$;

(c) $L_3 = \{a^{2n}b^n \mid n \geq 0\}$;

5.5 Give a formal description of a Turing machine that accepts the language

$$L = \{w\#x \mid w, x \in \{a,b\}^* \text{ and } w \text{ is a substring of } x\}.$$

Turing machine variants

5.6 Consider a variant of the Turing machine model, where the tape is bounded to the left (but still has infinite length to the right). If the head is in the leftmost position, and the transition functions requires the head to move left, then the head simply stays put. Initially the input is written from left to right starting at the leftmost cell. Show that such a Turing machine can simulate a regular Turing machine, which has a doubly infinite tape.

Turing machine variants

5.7 *Challenge question:* Consider a variant of the Turing machine model, where the tape is bounded to the left as in the previous question. In addition, the head cannot move a single position to the left. Instead, it can either only move one position to the right, or it "jumps" to the leftmost position (we call this a head reset). Hence, the transition function is of the form

$$\delta : Q \times T \to Q \times T \times \{R, RESET\}.$$

Show that such a Turing machine can simulate a regular Turing machine and vice versa.

Proofs, Turing machines

5.8 Complete the proof of Example 5.3: Show that if the Turing machine from that example accepts a string x, then $x \in \{0^n1^n \mid n \in \mathbb{N}_0\}$.

Turing machines

5.9 Give a formal description of the Turing machine from Section 5.3.2, which replaces the input $x \in \{0,1\}^*$ with xx.

Functions computed by Turing machines

5.10 For a non-negative integer $x \in \mathbb{N}$ let $\langle x \rangle_2$ denote the binary encoding of x. For $\Sigma = \{0, 1, \#\}$, let $f : \Sigma^* \to \Sigma^*$ be the partial function that maps each string $\langle x \rangle_2 \# \langle y \rangle_2$, where $x, y \in \mathbb{N}$, to $\langle x + y \rangle_2$. Give a Turing machine that computes f. (If $f(w)$ is not defined for some input $w \in \Sigma^*$, then the output can be arbitrary.)

Functions computed by Turing machines

5.11 Describe a multi-tape Turing machine that computes for each integer $x \in \mathbb{N}$ the function $f(\langle x \rangle_2) = \langle 2^x \rangle_2$. You may assume that the Turing machine has special states, which it can enter, upon which it computes on one of its tapes one of the following functions without changing any of the other tapes:

- *add*, where $add(\langle a \rangle_2 \# \langle b \rangle_2) = \langle a + b \rangle_2$ for $a, b \in \mathbb{N}$.
- *sub*, where $sub(\langle a \rangle_2 \# \langle b \rangle_2) = \langle a - b \rangle_2$ for $a, b \in \mathbb{N}$.
- *mult*, where $mult(\langle a \rangle_2 \# \langle b \rangle_2) = \langle a \cdot b \rangle_2$ for $a, b \in \mathbb{N}$.
- *comp*, where $comp(\langle a \rangle_2 \# \langle b \rangle_2) = 1$ if $a < b$ and $comp(\langle a \rangle_2 \# \langle b \rangle_2) = 0$, otherwise.

Turing machine variants

5.12 A *write-once* Turing machine is a single-tape Turing machine that can alter each tape cell at most once (including the input portion of the tape). Show that any standard Turing machine has an equivalent write-once Turing machine.

Hint: As a first step you may try to simulate M by a "write-twice" TM. Also, you will need to use a lot of tape space for the simulation.

Turing machine variants

5.13 Note that a Turing machine must have exactly one accepting state. But similar as for DFAs, we could define Turing machines with multiple accepting states. Prove that for any Turing machine M with multiple accepting states, there is an equivalent Turing machine M' that has exactly one accepting state. I.e., for each input x, M' halts if and only if M halts, and M' accepts if and only if M accepts.

Encoding objects

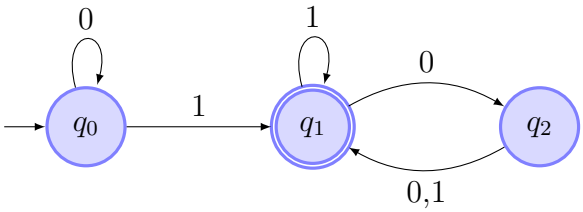
5.14 Describe encoding schemes for the following objects, using the alphabet $\{0, 1, \#\}$.

(a) A fraction, i.e., a number in $\mathbb{Q}$.

(b) A directed graph, where each edge has an associated *weight*, which is a non-negative integer. (Weights can be used to describe distances between endpoints, if the graph represents geographical locations and connections between them.)

Encoding Objects

5.15 Describe an encoding scheme for DFAs using the alphabet $\{0, 1, \#\}$. Encode the following DFA using your scheme:


0
1
0
1
q_0
q_1
q_2
0,1