4
Theory of Computation

4.1 Turing machines

A *Turing machine* is essentially a non-deterministic finite automaton enriched with external memory in the form of an infinite sequence of cells, called the *tape*. At every stage of the computation, the machine's *head* is placed over one of the cells. In every step, the machine changes its state, overwrites the contents of the current cell, and possibly moves its head to a neighbouring cell.

Formally, a Turing machine M over an input alphabet Σ has a finite set of states Q with a distinguished initial state $q_0 \in Q$, a subset $F \subseteq Q$ of *accepting states*, a finite tape alphabet $T \supseteq \Sigma$ with a distinguished *blank* symbol $\textsc{b} \in T - \Sigma$, and a transition relation

$$\delta \subseteq Q \times T \times Q \times T \times \{\leftarrow, \circlearrowleft, \rightarrow\}.$$

A transition rule $(q, a, p, b, d) \in \delta$ applies in state q if the machine's head sees the tape letter a in its current cell. The rule allows the machine to overwrite a with b, change the state to p, and either keep the head over the same cell or move it left or right, depending on d.

A *configuration* of the machine specifies the tape contents, the machine's state, and the position of the head: for $w, v \in T^*$ and $q \in Q$, the configuration wqv describes the situation where the tape contains the word wv with all remaining cells empty (that is, containing the blank symbol B), the state is q, and the head is placed over the cell containing the first symbol of the word v.

Given an input word w, the machine starts its computation in its initial state q_0, with its head over the first (left-most) symbol of w. The initial configuration is thus q_0w. The machine's run consists of applications of transition rules, which yield a finite or infinite sequence of consecutive configurations. A configuration wqv is accepting if the state q is accepting; a run is accepting if it is finite and its last configuration is accepting. The language $L(M)$ recognized by a machine M consists of all those words $w \in \Sigma^*$ for which M has an accepting run starting in the initial configuration q_0w. Two Turing machines are *equivalent* if they recognize the same language.

Unless stated otherwise, we assume that the tape is infinite in both directions; however, one could also consider a model with *right-infinite* tape, where there are no tape cells to the left of the initial position of the head. The two models are computationally equivalent (see Problem 163).

Transition rules of a machine with k tapes are in the following format:

$$\delta \subseteq Q \times T^k \times Q \times T^k \times \{\leftarrow, \circlearrowleft, \rightarrow\}^k.$$

Thus such a machine has k heads, moving independently, but a common state is used to determine their moves. A machine with any constant number of tapes can be simulated by a machine with a single tape. Therefore, the 1-tape model is computationally equivalent to the k-tape one, for all k.

The Turing machines discussed so far are *non-deterministic*. A machine is *deterministic* if its transition relation δ satisfies the following condition: for every state q and tape symbol a, there is at most one state p, tape symbol b and direction t such that $(q, a, p, b, t) \in \delta$ (equivalently, there is *exactly* one p, b and t). Thus, in every state q and for every tape symbol a, a deterministic machine has at most one possible transition rule to apply.

Problem 160. Construct Turing machines over the input alphabet $\{0, 1\}$ recognizing the following languages:

(1) $\{ww : w \in \{0,1\}^*\}$;

(2) palindromes;

(3) sequences over $\{0, 1\}$ representing prime numbers in binary notation.

Problem 161. A directed graph with n vertices $\{0, \ldots, n-1\}$ can be represented by a word over $\{0, 1\}$ of length n^2, whose kth letter is 1 if and only if there is an edge in the graph from vertex i to vertex j, where $k = n \cdot i + j + 1$.

(1) Construct a *non-deterministic* Turing machine that recognizes the language of all those words over $\{0, 1\}$ that represent a graph with a path from vertex 0 to vertex $n-1$.

(2) Construct a *deterministic* Turing machine recognizing the same language.

Problem 162. We say that a deterministic Turing machine over input alphabet $\{a\}$ computes a function $f : \mathbb{N} \to \mathbb{N}$ in *unary* representation if, for each $n \geq 0$, the computation of the machine starting in the initial configuration $q_0 a^n$ terminates in the configuration $q_f a^{f(n)}$, where q_f is a distinguished final state.

(1) Construct a Turing machine computing the function $n \mapsto 2^n$.

(2) Construct a Turing machine computing the function $n \mapsto \lceil \log_2 n \rceil$.

Problem 163. Prove that every Turing machine is equivalent to a machine with a right-infinite tape; that is, a tape that has no cells to the left of the initial position of the head.

Problem 164. Given a *non-deterministic* Turing machine construct an equivalent *deterministic* one.

Problem 165. Given two deterministic Turing machines M_1 and M_2 over the alphabet Σ, construct deterministic machines recognizing the following languages:

(1) $L(M_1) \cup L(M_2)$;

(2) $L(M_1) \cap L(M_2)$;

(3) $L(M_1)L(M_2)$;

(4) $L(M_1)^*$.

Problem 166. Prove that every Turing machine M over the input alphabet $\{0,1\}$ is equivalent to a Turing machine M' with tape alphabet $\{0,1,\text{B}\}$ which never writes the blank symbol B.

Problem 167. In a *write-once* Turing machine, whenever a transition rule overwrites a symbol a with a symbol b, either $a = \text{B} \neq b$ or $a = b \neq \text{B}$ holds.

(1) Given a 1-tape Turing machine, construct an equivalent 2-tape write-once machine.

(2) $(*)$ Prove that 1-tape write-once Turing machines only recognize regular languages.

Problem 168. $(*)$ Prove that a deterministic 1-tape Turing machine that makes $\mathcal{O}(n)$ steps on each input of length n recognizes a regular language.

Problem 169. Given a Turing machine, construct an equivalent 1-tape machine with four states.

Problem 170. The notion of pushdown automaton can be naturally extended to automata with k stacks, for any k. Show that every Turing machine is equivalent to an automaton with two stacks. Deduce further that every automaton with k stacks is equivalent to an automaton with two stacks.

An *automaton with a queue* is similar to pushdown automata, except that it performs operations on a queue, not on a stack. Transition rules are of the forms

$$(q, a, p), \qquad (q, \text{get}(s), p), \qquad (q, \text{put}(s), p),$$

where q, p are states, a is an input letter, and s is an element of a finite queue alphabet S. The first one reads a from the input; the second one is only enabled when s is the first symbol in the queue and it removes this symbol from the queue; the last one adds s to the queue as the last symbol. We assume that the queue is initially empty.

Problem 171. Prove that every Turing machine is equivalent to an automaton with a queue.

A *k-counter automaton*, for $k \geq 1$, is a non-deterministic finite automaton additionally equipped with *k counters* $c_1, \ldots, c_k$. Each counter stores a non-negative integer; initially, all the counters are set to 0, except for a distinguished counter c_1 whose initial value is understood as the input of the counter automaton. Thus, counter automata recognize sets of non-negative integers, rather than sets of words. Transitions of counter automata do not read input, but manipulate counters: every transition performs an operation on one of the counters c_i. The allowed operations are:

$c_i \overset{?}{=} 0$	(zero test),
c_i++	(increment),
c_i−−	(decrement),

but the transition c_i−− can be executed only if the current value of c_i is strictly positive. That is, counter values are not allowed to drop below 0.

Problem 172. Turing machines over a one-letter input alphabet can be viewed as acceptors of sets of natural numbers, written in unary notation. Prove that such Turing machines are equivalent to a 3-counter automata.

4.2 *Computability and undecidability*

A machine *halts* on an input word w if it has no infinite run starting from the initial configuration $q_0 w$. A language $L \subseteq \Sigma^*$ such that $L = L(M)$ for some Turing machine M that may or may not halt on all input words is called *recursively enumerable* or *semidecidable*. Problem 173 below justifies the common use of both these rather different names: it implies that a language L is accepted by a Turing machine if and only if there exists a (different) Turing machine that outputs all words in L one by one. A machine that halts on all inputs is called *total*. If $L = L(M)$ for a total machine M, then L is called *decidable*. Unsurprisingly, a language that is not decidable is called *undecidable*.

It is standard to identify a language with the computational problem of checking whether a given word belongs to the language. For example, one may say that "it is decidable whether a given number is prime", meaning that the language of all (representations of) prime numbers is decidable. In such statements one typically neglects to specify a concrete representation schema for numbers (or for automata, grammars, Turing machines or other input objects) as words, since decidability properties usually do not depend on the chosen method of representation.

Many natural computational problems are known to be undecidable, the halting problem for Turing machines being the archetypical example. Consider a representation of Turing machines as words over some fixed finite alphabet (for instance, as a list of alphabet letters followed by a list of transition rules). The halting problem is then the problem of checking, given a representation $[M]$ of a machine M and a word w over the input alphabet of M, whether M halts on input w. Other undecidable problems include checking whether a given machine accepts a non-empty language, a regular or context-free language, etc. In fact, the well-known Rice theorem says that every non-trivial question about the language accepted by a given Turing machine is undecidable.

Turing machines can be seen as devices for computing functions. One way to do that, for functions on natural numbers, is used in Problem 162. Another, more general way is to consider functions $f : \Sigma^* \to \Gamma^*$ for some alphabets Σ and Γ. If a machine M with input alphabet Σ, given input $w \in \Sigma^*$ as input, halts in an accepting configuration with a word $v \in \Gamma^*$ written on its tape, then we say that M computes a function f such that $f(w) = v$. In general the function computed by a machine is partial, because the machine may reject some inputs and may not halt on some inputs. Such a function is called a *partial computable function*. If the machine accepts every input then the function is total, and is simply called a *computable function*.

Problem 173. Prove that the following conditions on a non-empty language L are equivalent:

(a) L is recursively enumerable;

(*b*) L is the domain of some partial computable function;

(*c*) L is the image of some partial computable function;

(*d*) L is the image of some computable function.

Problem 174. Prove that a set $L \subseteq \mathbb{N}$, treated as a language L over the alphabet $\{0, 1\}$ via the standard binary representation of natural numbers, is decidable if and only if it is finite or it is the image of some strictly increasing computable function $f : \mathbb{N} \to \mathbb{N}$.

Problem 175. Prove the following Turing–Post theorem: if a language and its complement are both recursively enumerable, then they are both decidable.

Problem 176. $(*)$ Prove that there exists a recursively enumerable set whose complement is infinite but does not contain any infinite, recursively enumerable subset.

HINT: *Construct the set by choosing, for every Turing machine M that accepts an infinite language, a single word accepted by M. Choose wisely, so that the complement of your set remains infinite.*

Problem 177. Recall the definition of a k-counter automaton from Problem 172. Prove that there exists a 2-counter automaton A such that it is undecidable whether A halts on a given input number n.

Problem 178. $(*)$ Prove that the following *Post problem* is undecidable: Given two lists of words $u_1, \ldots, u_n \in \Sigma^*$ and $w_1, \ldots, w_n \in \Sigma^*$, is there a sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that

$$u_{i_1} \ldots u_{i_m} = w_{i_1} \ldots w_{i_m}?$$

If such a sequence exists then the word $u_{i_1} \ldots u_{i_n}$ (equal to $w_{i_1} \ldots w_{i_n}$) is called a *solution* of the instance of the Post problem.

HINT: *As an intermediate step, use a modification where i_1 has to be* 1.

Problem 179. Prove that the following *universality problem* is undecidable: given a context-free grammar G over an alphabet Σ, is it the case that

$$L(G) = \Sigma^* \,?$$

Problem 180. Are the following problems decidable?

(1) Given a context-free grammar G, does $L(G)$ contain a palindrome?

(2) Given two context-free grammars G_1 and G_2, is it the case that

$$L(G_1) \cap L(G_2) = \emptyset \,?$$

(3) Is a given a context-free grammar G unambiguous?

Hint: Use the Post problem.

Problem 181. Assume that we know that a 1-tape, deterministic Turing machine M makes at most one "turn"; that is, once the head moves to the left it never moves to the right again. Is it decidable whether a given machine M with this property accepts a given word w?

Problem 182. Is the following problem decidable: given words $u, w \in \Sigma^*$ and a number k, is there a word $x \in \Sigma^*$ of length at least k such that $\#_u(x) = \#_w(x)$?

Problem 183. Fix an encoding of Turing machines that represents a machine M as a word $[M]$ over $\{0, 1\}$. Consider a function C that maps a pair $([M], v)$ to the minimal length of a word w such that $M(w) = v$, or to a special symbol ∞ if there is no such w.

(1) Prove that the function C is not computable.

(2) Prove that the function C can be approximated in the following sense: there is a Turing machine that, for input $([M], v)$, produces an infinite sequence of numbers that eventually stabilizes at the value $C\left([M], v\right)$.

Note that there is no contradiction between (a) and (b), since an observer of an infinite sequence can never say whether it has already stabilized.