4 Turing machines

URMs as a model of computation have the advantage that they are fairly easy to use and understand. There is, however, also a drawback: The execution of a single URM instruction, e.g. $\text{jump}(m,n,q)$, can be arbitrary complex: first the contents $a_n$ and $a_m$ of the Registers $R_m$ and $R_n$ have to be read, where $a_n$ and $a_m$ may be arbitrarily large numbers, then $a_n$ and $a_m$ have to be compared. Because the possible sizes of $a_n$ and $a_m$ are unbounded, we see that the execution of the single instruction $\text{jump}(m,n,q)$ may consume arbitrarily much time and space.

For this reason URMs are unsuitable for the analysis of the complexity of an algorithm, i.e. the estimate of the number of elementary steps to execute it.

We now introduce an alternative model of computation, introduced by Alan Turing (1912-1954), which is more appropriate for complexity analysis.

The definition of computability proposed by Turing in 1936 is based on an analysis of how a human agent would carry out an algorithm using pen and paper. Turing observed that the agent’s activities can be reduced to a succession of the following two fundamental actions:

(a) writing or erasing a single symbol;

(b) transferring attention from one part of the paper to another.

At each stage the algorithm specifies the action to be performed next. This depends only on the symbol on the part of the paper currently scrutinized, and on the current state of the agent. The state of the agent may, of course, change as a result of the action taken at this stage. There are only finitely many states and only finitely many symbols (to be written, read or erased) available. However the paper is assumed to be given as a tape divided into cells which is (potentially) infinite (in both directions), reflecting the view that we do not take limitations in time and space into consideration.

Exercise. Draw the tape of a Turing machine.
4.1 The definition of a Turing machine

A Turing machine is a triple $(\Sigma, Q, P)$ where

- $\Sigma = \{s_1, \ldots, s_n\}$ is a finite set of symbols, called the alphabet, which must not contain the special symbols $L$ and $R$.
- $Q = \{q_1, \ldots, q_k\}$ is a finite set of symbols, called states,
- $P$ is the specification, which is a finite list of quadruples called instructions, which are of the form
  - $(q, s, s', q')$, where $q, q' \in Q$ and $s, s' \in \Sigma$, or
  - $(q, s, L, q')$, where $q, q' \in Q$, or
  - $(q, s, R, q')$, where $q, q' \in Q$.

It is required that any for any pair $(q, s) \in Q \times \Sigma$ there is a unique instruction in the specification $P$ of the form $(q, s, \alpha, q')$.

The action of a Turing machine $(\Sigma, Q, P)$ can be described as follows: We assume that that in each cell of the tape a symbol of the alphabet $\Sigma$ is written.

At each stage the machine $M$ 

- is in a specific state $s \in Q$ called the current state,
- reads a specific cell of the tape, called the scanned cell. We also call the scanned cell the position of the reading head.

The machine $M$ together with an inscription on the tape, the current state, and the scanned cell is called a configuration.

In order to describe the action of $M$ we assume a specific configuration to be given. Let $q \in Q$ be the current state and $s \in \Sigma$ the symbol written on the scanned cell.

There are 4 cases

1. There is an instruction $(q, s, s', q') \in P$ with $s' \in \Sigma$.

   **Action**: erase $s$ from the scanned cell, write $s'$ on it and change the current state to $q'$.

2. There is an instruction $(q, s, L, q') \in P$.

   **Action**: move the reading head one cell to the left (i.e. the scanned cell is now the left neighbor of the previously scanned cell) and change the current state to $q'$.

3. There is an instruction $(q, s, R, q') \in P$.

   **Action**: move the reading head one cell to the right (i.e. the scanned cell is now the right neighbor of the previously scanned cell) and change the current state to $q'$.

4. There is no instruction in $P$ of the form $(q, s, \alpha, q')$.

   **Action**: The machine $M$ stops.
Example. Consider the Turing machine $M := (\Sigma, Q, P)$ with
\[\Sigma := \{0, 1, \square\} \text{ (\square is used as a “blank symbol”)}\]
\[Q := \{q_1, q_2\}\]
\[P := \]
\[(q_1, 0, R, q_1)\]
\[(q_1, 1, 0, q_2)\]
\[(q_2, 0, R, q_2)\]
\[(q_2, 1, R, q_1)\]

Assume the following configuration:

The scanned cell as well as its 9 right neighbors contain the symbol 1. All other cells contain the blank symbol $\square$.

The (initial) current state is $q_1$.

Exercise. Draw this configuration and describe the actions taken by $M$.

4.2 Turing-computable functions

A Turing machine $M = (\Sigma, Q, P)$ with alphabet $\Sigma = \{0, 1, \square\}$ and a distinguished initial state $q_0 \in Q$ computes a partial function
\[M^{(k)} : N^k \rightarrow N\]

at arguments $a_1, \ldots, a_k$ as follows:

1. starting with the scanned cell, write from left to right
\[\text{bin}(a_1), \square, \ldots, \square, \text{bin}(a_k)\]
on the tape, where $\text{bin}(a_i)$ is the binary representation of $a_i$. All other cells of the tape contain the blank, $\square$.

2. Run the machine starting with the initial state $q_0$ as current state.

3. When the machine eventually stops and the inscription on the tape starting from the scanned cell to the right is of the form
\[\text{bin}(a), \square, \ldots\]
then the result is a i.e. $M^{(k)}(a_1, \ldots a_k) = a$.

4. Otherwise $M^{(k)}(a_1, \ldots a_k)$ is undefined.

A partial function $f: \mathbb{N}^k \rightarrow \mathbb{N}$ is called **Turing-computable** if $f = M^{(k)}$ for some Turing machine $M$.

**Exercise.** Show that addition is Turing-computable.