1.e Exercises for Lecture 1

Here are all the exercises from the lecture notes, reorganized and renumbered to make an exercise sheet. There are also a few new questions, to keep things interesting. Solve only what you like.

Exercise 1.e.1. In this exercise, we deal with numbers that might be greater than 2^{31} (but still nonnegative). Recall that integers are represented as sequences $n_0, n_1, \ldots, n_k, \#$, where $\# = 2^{31}$ and $\overline{n} = n_k \ldots n_1 n_0$ in base 2^{31}. Differently said: the first digit to appear is the lowest-significant one.

(a) Design a Turing machine that computes the function $n \mapsto n+1$.

(b) Let $z = 2^{31} - 1$; draw the space-time diagram of the Turing machine you designed for (a) on the input $z, z, z, \#$.

(c) Design a Turing machine that computes the function $m, n \mapsto m + n$.

(d) Design a Turing machine that computes the function $m, n \mapsto 0$ if $m = n$ and 1 otherwise.

(e) Design a Turing machine that computes the function $m, n \mapsto 0$ if $m \leq n$ and 1 otherwise.

(f) Design a Turing machine that compute multiplication of integers.

(g) Design Turing machines that compute substraction and division of integers. *Hint:* reuse your addition/multiplication machines, and use a brute-force algorithm.

Exercise 1.e.2.

(a) Write a Turing machine interpreter in Python (or any language you like).

(b) Improve your interpreter so that it shows the space-time diagram of the computation that it is running.

Exercise 1.e.3. Design a Turing machine that takes a sequence of integers between 0 and $2^{31} - 1$ (included), terminated by #, and sorts that sequence in nondecreasing order.

Exercise 1.e.4.

(a) Prove that a function f is computable by a Turing machine if and only if it is computable by a 2-memory Turing machine (cf. §1.6.6).

(b) Write the definition of a k-memory Turing machine, and prove that k-memory and ℓ-memory Turing machines are equivalent (in the sense of the previous question) for all $k, \ell \geq 1$.

(c) Prove that the set of computable functions is the same for any finite set $\mathbb{I}$ with at least two elements. In other terms, whether $\mathbb{I} = \{0, \ldots, 2^{32} - 1\}$ or $\mathbb{I} = \{0, 1, \ldots, 9\}$ or $\mathbb{I} = \{0, 1\}$ does not change which functions are computable.

(d) Prove that a Turing machine where the memory is bi-infinite, i.e., a function $\mathbb{Z} \to \mathbb{I}$ instead of $\mathbb{N} \to \mathbb{I}$, can compute the same functions as a normal Turing machine.

Exercise 1.e.5. Find, on the internet:

(a) An undecidable problem other than the Halting problem. Try to understand its proof.

(b) A compiler from any programming language to Turing machines. Try it on simple programs and look at the resulting machines.

(c) Try to read a bit of the source code of the compiler you found.

Exercise 1.e.6 ( ). Write a Python class that models an arithmetic expression, i.e., a binary tree where internal nodes are labeled by arithmetic operations $(+, -, \times, \div, =, <)$ and leaves have labeled by integers. Then, write a translator from arithmetic expressions to Turing machines.

Exercise 1.e.7 ( ). Feel free to change $\mathbb{I}$ in order to have more additional symbols: for instance, you can set $\mathbb{I} = \{2^{33} - 1\}$.

(a) For each integer i in $\mathbb{N}$, design a Turing machine that, on input:

v_0	#	v_1	#	...	v_{k-1}	#	#	0	0	0	...

writes a copy of v_i after the ##. Note that $v_0, v_1, \ldots, v_{k-1}$ are *blocks* of several cells, not just single cells. *Hint:* you need to temporarily change the original v_i in order to write a copy, but you can restore it later.

(b) Conversely, for each integer i in $\mathbb{N}$, design a Turing machine that, on input:

v_0	#	v_1	#	...	v_{k-1}	#	#	v_k	#	0	0	0	...

erases v_i and writes a copy of v_k instead. Note that v_k might be shorter or longer than v_i! Besides, we might have $i \geq k$; in this case, cells containing just 0 should be inserted to expand the array up to the right size.

Exercise 1.e.8 ( ). Suppose we have three Turing machines T_1, T_2, and T_3 for Python programs P_1, P_2, and P_3 respectively. Design Turing machines that do:

- `if`(P_1): P_2 `else`: P_3
- `while`(P_1): P_2

Exercise 1.e.9 ( ). Write a Simple Python → Turing machine compiler. (The definition of Simple Python is in Claim 1.6.2). Reuse the constructions you designed in the previous exercises!

Contacts

- Daria Pchelina (dpchelina@clipper.ens.fr)
- Guilhem Gamard (guilhem.gamard@normale.fr)