TOC Week 2

June 2021

1 Turing Machines

Turing Machine model represents the current day computers we are familiar with. It is an abstract model just like finite automata. We'll see that Turing machines capture all of the computation in the sense that any function that can be computed by any physically realizable machine can be computed by a Turing Machine.

2 k-Tape Turing Machines

A k-tape TM has k tapes equipped with tape-heads to read and write, 1 read-only input tape for reading the input, k-1 read/write work tapes where it carries out its processing and writes output on one of them-the output tape.

Formally, a k-tape TM is give by the tuple $\langle \Sigma, q, Q \rangle$ where:

- $\Sigma = \{0, 1, >, \#\}$ is the alphabet
- Q is the set of states containing two special q_{start} and q_{halt} states
- $q : Q \times \Sigma^k \to Q \times \Sigma^{k-1} \times \{L, S, R\}$ is the transition function

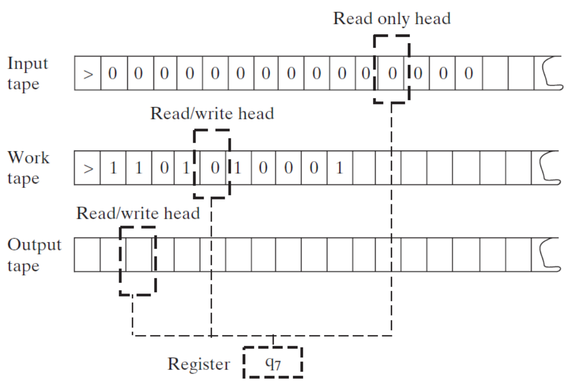

Figure 1: Birds

Although TM on first sight may seem simple, but it has too much expressive power in the sense that it can compute any function that your PC can, yes!. It can run any of your programs written in your favourite language, do Graph traversal for you, etc.

Example: Design a TM that checks if given string is a palindrome or not.
Let's see it this way, we use a 3 tape TM. In the first step we write 1(indicating true) on the output tape. Then, say in one go we just copy input to work tape, will take us n(size of input) steps. Then we bring the head of work tape to beginning again n steps, then we start matching the bits under the two heads on input and work tape and keep sliding them to left and right respectively, if at any state we find a mismatch, we write 0 on output tape and halt taking at most n steps. See that run time of this machine is $3n + 1$.

3 Computation and Complexity

Let $f : \{0, 1\}^* \to \{0, 1\}^*$ and let $T : N \to N$ be some function , and let M be a Turing machine. We say that M computes f if for every $x \in \{0, 1\}^*$, whenever M is initialized to the start configuration on input x, then it halts with $f(x)$ written on its output tape. We say M computes f in $T(n)$ - time if its computation on every input x requires at most $T(|x|)$ steps.

4 Variants of TM

- 1. TM using an arbitrary language
- 2. single tape TM
- 3. bidirectional tape TM
- 4. oblivious TM

A function computable on a TM of any of these kind can be computed on a TM of all other kinds as well.

Proof idea for 1-

5 TM as strings and Universal TM

This idea is behind the use of a single PC to do all tasks. Recall how the finite automata and TM we defined are meant to do a specific task, like adding two numbers or finding if a string is palindrome or not.

Now I tell you that any TM T can be represented by a string, yes, we can do it. Then this string can again be taken as input by some other TM $\mathcal{U}$(which we call the Universal Turing Machine) which can then simulate the behaviour of T on all its inputs. It should click you that it is the exact idea of programming languages, what we write as a program is a description of a machine designated to do a specific task which is then simulated by our PC(the Universal TM).

So how do we do it, remember the idea of encoding, first realise that if we are given only the transition function of a TM, we know everything we need to know about it, we can infer the language and set of states from the arguments and outputs of the transition function. Now to represent any TM as string, we just need to encode its transition function which can be encoded as a string containing tuples of arguments and outputs of the transition function.

Now we have ensured that every TM is represented by a string.

We can further strengthen our TM to string mapping in this way. If a TM is represented by some string S, we may pad S with arbitrarily many 1 (s), you can convince yourself this can be done. After this we can map all the remaining strings to some trivial TM, so that we now have mapped every TM to infinitely many strings and every string to some TM.

Theorem: There exists a TM $\mathcal{U}$ such that for every $x,\ \alpha \in \{0,1\}^*$, $\mathcal{U}(x,\alpha) = M_\alpha(x)$, where M_α denotes the TM represented by α. Moreover, if M_α halts on input x within T steps then $\mathcal{U}(x,\alpha)$ halts within $CTlogT$ steps, where C is a number independent of $|x|$ and depending only on M_α's alphabet size, number of tapes, and number of states.

6 Limits of TM

There are functions which a TM can't compute;

Example 1: The function UC defined as- for a string $x \in \{0,1\}^*$ if $M_x(x) = 1, UC(x) = 0$, otherwise $UC = 1$.

Proof: (Diagonalization) Say a TM M_α computes UC, give M input α say M outputs 1 $\Rightarrow UC = 0$, a contradiction. Or if M doesn't halt on x or gives output 0, $UC = 1$, again a contradiction.

Example 2: $HALT$ s.t. $HALT$ on input $\langle x, \alpha\rangle$ gives 1 iff M_α halts on x within finite number of steps.

Proof: (Reduction) The idea is to prove that if a TM can compute $HALT$ it can compute UC, but since no TM can compute UC, it means we can't compute $HALT$ as well. The idea is called reduction we reduce solving a problem A to solving a problem B. See that if A can be in this way reduced to B, B is at least as hard as A in the sense that if A can't be solved B can't be solved as well, meaning in a sense B is harder.

Say a TM computes $HALT$. We can create a TM T computing UC. First giving $HALT$ input $\langle x, x\rangle$, if output is 0, meaning M_x doesn't halt on x, T outputs 1, else T uses the uiversal TM to simulate M_x on x and output it's opposite.