CS 580

1 Turing Machines

In this course will study theoretical models to capture the essence of natural (mechanical) processes in a unified way. The word "computer" dates back to 1613 (in written records) and used to mean "someone who computes": a person performing mathematical calculations, before electronic computers existed. The idea was that "the human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail." Groups of people were often employed to undertake long and tedious calculations, with the work divided so it could be finished in parallel. (see [wik]).

1.1 Definition

A Turing Machine (TM) consists of:

- An infinite tape. The cells of the tape are numbered $1, 2, \dots$ as a convenience when proving properties of a TM. These indexes are not known to the TM. [Memory of a Computer].
- A two-way read/write head [Program Counter: i.e. the processor register that indicates where a computer is in its program sequence.]
- Finite control (defined by a set of states Q and a transition function δ) [The Program].

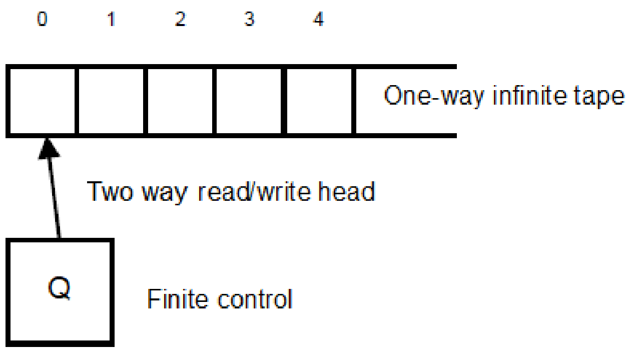

Figure 1: Turing machine

You will see some definitions where the tape is two-way (extends infinitely in both directions). These definitions do not matter (the model is robust – the classes of things computed by these different models are the same].

It is defined by:

- Q is the set of states.
- Σ is the input alphabet (not containing the blank symbol $\square$).
- Γ is the tape alphabet, with $\square \in \Gamma$, $\Sigma \in \Gamma$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
- $s \in Q$: Start state.
- "yes" $\in Q$: Accept state.
- "no" $\in Q$: Reject state.

Initially:

- The input $x_1 \ldots x_n$ is placed on cells $1 \ldots n$.
- All other cells contain the blank symbol $\square$. Head starts at position 1.

Transition details: $\delta(q, a) = (q', b, L)$ then:

- If the machine is in state q and the head reads symbol a, then the state changes to q', the machine writes the symbol b in the place of a, and the head moves left one cell.

How it works (high level):

- Given the input, the machine computes following the rules induced by the transition function. Computation continues until it enters the accept or reject states (then it halts). Otherwise, it goes on forever (Have you written a program with an infinite loop?).

1.2 Configurations & accepting sequence of configurations

Definition 1 (Configuration)**.** *A configuration of a Turing machine M is a setting consisting of* current state*, the* current tape content*, and the* head location*. It is denoted by a tuple uqv, where u is the tape content to the left of the head, q is the current state, v is the tape content to the right of v, and the head is positioned on the first letter of v.*

A configuration C yields configuration C' if the machine can go from C to C' in a single step. In particular, we have the start configuration, some accepting configurations, and some rejecting configurations.

A Turing machine M accepts an input w if there is a **sequence of configurations** $C_1 \ldots C_T$ such that:

- C_1 is the start configuration.
- Each configuration C_k yields configuration C_{k+1}.
- C_T is an accepting configuration.

1.3 Language of a TM; Recognizable, Decidable, and Undecidable Languages

The collection of strings that a Turing machine M accepts represents the **language of M** (or the language recognized by M), denoted by $L(M)$.

Definition 2 (Turing recognizable languages)**.** *A language A is Turing-recognizable if some Turing machine M recognizes it (i.e. $L(M) = A$). [aka recursive enumerable language]*

Definition 3 (Deciders)**.** *A machine can fail to accept a string in two ways: actually end in a rejecting configuration, or loop. A Turing machine that always halts is called a **decider**.*

Definition 4 (Turing decidable languages)**.** *A language A is (Turing-)decidable if some Turing machine M decides it (i.e. $L(M) = A$ and M always halts). [aka recursive language]*

2 Examples

We will skip the very low level details. Take a look at page 165 of Sipser for a complete Turing machine.

Definition 5 (Undecidable problem)**.** *We say a yes/no* problem *is undecidable if there is no Turing machine that always halts with a correct yes/no answer (similar for decidable).*

Problem 1. *Find a TM for deciding $L = \{0^{2^n} \mid n \geq 0\}$.*

Proof. Here is a Turing machine that decides the language:

1. Move the head across the tape, from left to right, crossing off every other 0.
2. If at step 1 the tape contained only one 0, accept.
3. If at step 1 the tape contained more than one 0 and the number was odd, reject.
4. Return the head to the left end of the tape.
5. Go to step 1.

□

Definition 6 (Time complexity)**.** *We denote by $time_M(x) = \#$ steps that M takes to halt on input x.*

Problem 2. *Find a TM for deciding $PAL = \{x_1 x_2 \ldots x_n \in \Sigma^* \mid x_1 \ldots x_n = x_n \ldots x_1\}$.*

Proof. Here is a TM for it:

1. Read symbol of the string, and let the finite control remember it (by state), write the blank symbol, and move to the right until a blank symbol is reached (if a blank symbol is immediately reached, accept).
2. Move 1 step left, read the symbol, and compare it with the remembered symbol (and reject when they are not equal).

3. Write a blank symbol, and move left until reaching a blank symbol (if a blank symbol is immediately reached, accept). Move 1 step right.

4. Repeat.

What is the runtime of this? Convince yourself that it is $O(n^2)$. □

Problem 3. *Find a TM for deciding: "Given an integer k and Turing machine M promised to be in P , is the runtime of M in $O(n^k)$ with respect to input length n?*

Proof. This is not decidable! □

3 Turing Machine Variants

There are many variants of Turing machines, including multi-tape machines, non-deterministic machines, enumerators, etc. The model is robust, that is, all these models are equivalent in terms of what languages they recognize.

3.1 Multi-tape Machines

A **multi-tape** Turing machine is just like a usual TM, but with multiple tapes instead of one. Each tape has its own head for reading and writing. The first tape starts with the input, the others start empty (with blank symbols all the way). The transition function allows reading, writing, and moving the heads simultaneously; formally, the transition function for a k-tape TM is of the form $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$.

- The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$ means that if the machine is in state q_i and the heads $1 \ldots k$ are reading symbols $a_1 \ldots a_k$, then the machine goes to state q_j, writes symbols $b_1 \ldots b_k$, and directs each head to move left or right or stay put, as specified.

Theorem 3.1 (Tape reduction)**.** *Every multi-tape Turing machine has a one-tape equivalent.*

Proof. The high level idea is given in Figure 2. Given k-tape machine M, construct one-tape machine S that works as follows (S will have a slightly augmented language). On input $w_1 \ldots w_n$:

- Start by formatting the tape of S in a way that represents all the k-tapes of M (back to back, from left to right). See figure 3.

- How to simulate a single move? S scans the tape from left to right to determine the symbols under the tape heads. Then S figures out what it should transition to, and makes a second pass to update the symbols under the tape heads and the new positions of the heads.

- What if the space allocated for one of the tapes (in S) is not enough? This can be detected: If at any point S moves a head to the right onto a #, this means that M has moved the corresponding head to a previously blank portion of the tape. Then S writes a blank symbol on this cell and shifts everything one unit to the right. Then it continues the simulation.

□

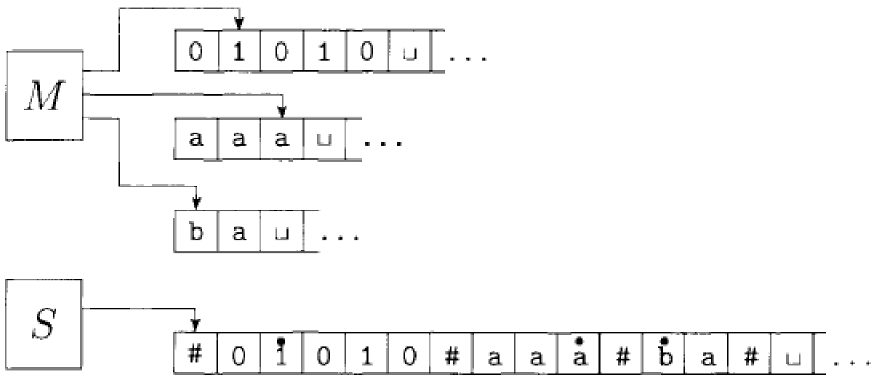

Figure 2: Tape reduction

$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#$$

Figure 3: One tape representing k tapes

Theorem 3.2 (Universal Turing machine)**.** *For any k, there exist a $k+1$ tape Turing machine U_k, that given as input a description of a k-tape Turing machine M as well as input x simulates M on input x in $O(time_M(x))$. steps, where the constant in the $O(\cdot)$ notation depends on M.*

Proof. Simple simulation. Keep description and state on tape $k+1$. Other tapes work just like on M (except an encoding of the tape alphabet is used). □

More details can be found in Chapter 3 of Sipser [Sip12].

References

[Sip12] Michael Sipser. *Introduction to the theory of computation.* Cengage Learning, 2012.

[wik] Human computer. https://en.wikipedia.org/wiki/Human_computer. Online; accessed Jan 2018.