CSE200: Complexity theory
Computational models and uncomputability

Shachar Lovett

January 6, 2020

1 Turing machines

A Turing machine (TM) is a relatively simple mathematical model, which still captures the full power of general purpose computing. A Turing machine has an input tape, an output tape and at least one work tape. Tapes are divided into cells, each of which can record a single symbol. There is a reading head on each tape which can be moved one step in each operation. It also has a finite state which can be used to represent code, registers, etc. Formally, a TM is defined as follows.

Definition 1.1 (k-tape Turing machine)**.** *A k-tape Turing machine for $k \geq 3$ is defined as follows.*

- *The first tape is the input tape, the second tape is the output tape, and the remaining $k-2$ tapes are work tapes. The input tape is read-only, and the output and work tapes are read-write.*
- *Γ is the alphabet (a finite set of symbols, e.g ASCII).*
- *Q is a finite set of internal states (e.g. code, location in code, registers).*
- *$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ represents the state change, tape rewrite and tape head movement (Left, Stay or Right) given the current state and the characters under the reading heads.*
- *The TM starts at state q_{start} and terminates when it reaches state q_{end}.*

The alphabet contains a special blank symbol $\perp$. Each tape has only a finite number of non-blank symbols. The *content* of each tape is the prefix of the tape, which is followed by blank symbols. The *configuration* of a TM is a snapshot of its current state, the contents of the tapes and the location of the reading heads on each tape.

Turing machines can be used to compute partial functions. A partial function $f : \{0,1\}^* \to \{0,1\}^* \cup \{\text{undefined}\}$ maps every input $x \in \{0,1\}^*$ either to an output $f(x) \in$

$\{0,1\}^*$, or is undefined, in which case $f(x) =$ undefined. We assume that the alphabet Γ contains the symbols $0, 1$ so we can encode the input in the input tape.

Definition 1.2 (Turing machines compute a partial function)**.** *Let M be a Turing machine. It computes a partial function $f : \{0,1\}^* \to \{0,1\}^* \cup \{undefined\}$ as follows:*

- *Let $x \in \{0,1\}^*$ be the input.*
- *The TM is initialized as follows: the initial state is q_{start}, the input tape has the input x padded by blanks, and the remaining tapes are blank.*
- *The TM is run.*
- *If the TM reaches the state q_{end} then it halts, and we set $f(x)$ to be the content of the output tape. If the TM never halts, then we set $f(x) = undefined$.*

A TM is *decidable* if it halts on all inputs. In particular, such a TM computes a *total* function. For such TMs it makes sense to measure the time they take to halt on inputs.

Definition 1.3 (Time complexity)**.** *Let $T : \mathbb{N} \to \mathbb{N}$ be a time bound. A decidable TM has* time complexity *T if $M(x)$ halts in at most $T(|x|)$ steps for all inputs $x \in \{0,1\}^*$.*

The main thing we will show is that **the model doesn't matter**. That is, with only small increases in running time, we can simulate any reasonable variant of a Turing machine with a "standard TM" that has exactly 3 tapes and a binary alphabet.

Lemma 1.4 (The alphabet doesn't matter)**.** *Assume $f : \{0,1\}^* \to \{0,1\}^* \cup \{undefined\}$ can be computed by a TM with alphabet Γ. Then f can also be computed by a TM with alphabet $\Gamma' = \{0, 1, \bot\}$.*

Proof. Assume $\Gamma = 2^c$. Let M be a TM computing f with alphabet Γ. We will simulate M by a TM M' which has alphabet $\Gamma' = \{0, 1, \bot\}$ and a few more states. Formally, assume M has $W_1, \ldots, W_k$ as contents of the tapes, reading heads at locations $i_1, \ldots, i_k$ and state $q \in Q$. Then

- Each symbol in the tapes of M is encoded by c bits.
- The reading heads in M' tapes are in the first bit of the corresponding symbols, namely at indices $c \cdot i_1, \ldots, c \cdot i_k$.
- M' reads the c bits in each location and stores them in internal registers (that is, part of the state space Q' of M').
- M' simulates the change of state of M, writes the corresponding bits encoding the new symbols, and moves the heads accordingly.

At the end, the configuation of M' encodes the next configuation of M. Note that if M runs in time $T(n)$ on inputs $x \in \{0,1\}^n$ then M' runs in time $O(T(n) \log |\Gamma|)$. □

Lemma 1.5 (The number of tapes doesn't matter)**.** *Assume $f : \{0,1\}^* \to \{0,1\}^* \cup \{undefined\}$ can be computed by a TM with $k \geq 3$ tapes. Then f can also be computed with a TM with three tapes (input, output and work tapes).*

Proof. Let M be the k-tape TM computing f. We simulate M using a 3-tape TM M' as follows:

- Encode reading head location in each tape as part of the input alphabet. That is, for each alphabet character $c \in \Gamma$, we add a new character c' which symbolizes that the head is above that symbol. This increase the alphabet size by a constant factor.
- Concatenate the working tapes of M on the single work tape of M', with a new symbol used to separate them.
- To simulate a step in M: initially all reading heads of M' are at the beginning of their tape. First, M' scans for the reading heads of all tapes and saves the symbol under each one of them (as part of its internal state). Then, it simulates the change in M, by scanning the tape again and making the appropriate changes.

If $x \in \{0,1\}^n$ is the input of M and runs in time $T(n)$, then the tapes have at most $k \cdot T(n)$ non blank symbols, hence each step if M is simulated by $O(k \cdot T(n))$ steps in M'. Hence M' runs in time $O(k \cdot T(n)^2)$. □

A machine with a RAM can read a word from memory by a single instruction. Formally, a RAM TM has a special RAM tape, on which the TM can write an index. It has a special state which reads the "main memory" tape at that location. A RAM TM can be simulated by a standard TM.

Lemma 1.6 (RAM doesn't matter)**.** *If $f : \{0,1\}^* \to \{0,1\}^* \cup \{undefined\}$ can be computed by a TM with RAM then it can also be computed by a standard TM.*

Proof. Simulate the RAM read by scanning the working tapes. Increases time from T to $O(T^2)$. □

The simulations we saw increase runtime by at most squaring it. So the models discussed above are all equivalent up to a quadratic factor in the runtime.

2 Universal Turing machines

We can encode a TM M by a string $\langle M \rangle \in \{0,1\}^*$ representing its "program" (e.g. state space, alphabet and transition function). A universal machine is simply an interpreter for these programs. We say a Turing machine U taking two inputs is a universal machine if it satisfies:

$$U(\langle M \rangle, x) = M(x),$$

and if M loops forever so does U. If $\langle M \rangle$ is not a legal program (does not compile) then U outputs some default output to indicate this.

Lemma 2.1. *There exist universal Turing machines.*

Proof. Assume for simplicity[1] that M has a single work tape and alphabet $\Gamma = \{0, 1, \perp\}$. U will have two work tapes: one that is a copy of the worktape of M, and another auxiliary work tape for internal computation of the "interpreter" U. The auxiliary work tape will be used to remember the state of M. To simulate a step in M:

- Read input from work tape of M.
- Scan the transition function (given in the input) to match the state of M (given in the auxiliary work tape). Write the new state of M in the auxiliary work tape, and implement the changes in the work tape of M that we maintain (change bit value, move head).

If $M(x)$ runs in time $T(n)$ then $U(\langle M \rangle, x)$ runs in time $O(|\langle M \rangle| \cdot T(n))$. □

It can be shown (though we won't show it here) that even if M has more than one work tape, it can still be simulated by a single work tape universal machine in time $O(T(n) \log T(n))$.

3 Uncomputable functions

Not all functions are computable. Define a function $HALT : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ as

$$HALT(\langle M \rangle, x) = \begin{cases} 1 & \text{If } M(x) \text{ halts} \\ 0 & \text{If } M(x) \text{ doesn't halt, or } \langle M \rangle \text{ is not a valid code} \end{cases}$$

Theorem 3.1. *$HALT$ is not computable by Turing machines.*

Proof. We abbreviate $h = HALT$. Assume h is computable by a Turing machine. Then so is the following partial function

$$g(\langle M \rangle) = \begin{cases} 0 & \text{If } h(\langle M \rangle, \langle M \rangle) = 0 \\ \text{undefined} & \text{If } h(\langle M \rangle, \langle M \rangle) = 1 \end{cases}$$

Assume g is computable by a Turing machine M. We reason by cases on the value of $g(\langle M \rangle)$:

- If $g(\langle M \rangle) = 0$ then $h(\langle M \rangle, \langle M \rangle) = 0$, meaning $M(\langle M \rangle)$ never halts, so $g(\langle M \rangle)$ should be undefined.
- If $g(\langle M \rangle) =$ undefined then $h(\langle M \rangle, \langle M \rangle) = 1$, meaning $M(\langle M \rangle)$ halts, so it must be that $g(\langle M \rangle) = 0$.

We see that $g(\langle M \rangle)$ cannot be defined, meaning that g (and h) cannot have a TM computing them. □

[1] By the above discussion showing that alphabet and number of tapes does not matter, this assumption is without loss of generality.

There are many uncomputable functions. Let's show another one — the busy beaver function. For this one it would be convenient to consider the following variant of a TM without input. These machines have alphabet $\Gamma = \{0, 1\}$ and just one tape. The initial configuration is where the tape is all zeros, the head is in the beginning of the tape, and the machine is in the start state. When the machine halts, the output is the content of the tape (removing any suffix of all zeros). Define the Busy Beaver function $\mathrm{BB} : \mathbb{N} \to \mathbb{N}$ as the maximal number of 1's which can be written by a **halting** TM with at most n states.

Theorem 3.2. *The busy beaver function is uncomputable.*

Proof. Assume BB can be computed by a TM M which has k states. Formally, we assume that M maps the input 1^n0^* to $1^{\mathrm{BB}(n)}0^*$. Let M_{DBL} be a TM which, running on a tape with content 1^n0^*, doubles the number of ones, e.g. ends in $1^{2n}0^*$. Assume M_{DBL} has ℓ states. Let $n \geq 1$ be a parameter to be chosen later, and consider the following TM N:

1. Write n ones on the tape; move head back to beginning. // Tape = 1^n0^*
2. Apply M_{DBL}. // Tape=$1^{2n}0^*$
3. Apply M. // Tape=$1^{\mathrm{BB}(2n)}0^*$
4. Add one more 1. // Tape=$1^{\mathrm{BB}(2n)+1}0^*$

The TM N outputs $\mathrm{BB}(2n) + 1$ ones. Lets see how many states are needed to implement it. Step 1 requires $n + O(1)$ states (the n states to write the n ones, the additional $O(1)$ states to move the head back to the beginning). Step 2 requires ℓ states. Step 3 requires k states. Step 4 requires additional $O(1)$ states. In total N has $n + k + \ell + c$ states, where $c = O(1)$, and it outputs $\mathrm{BB}(2n) + 1$ many ones. Let $r = k + \ell + c$, and recall that we can still choose n freely. We have

$$\mathrm{BB}(n + r) \geq \mathrm{BB}(2n) + 1 \qquad \forall n \geq 1.$$

But if we choose $n \geq r$ we reach a contradiction, as BB is a monotone function. □

Many other mathematical/computational problems are uncomputable. For example:

- Rice's theorem: The only computable functions $f : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}$ for which $f(\langle M \rangle, x)$ depends just on $M(x)$ are the constant functions.
- Generalized Collatz conjecture ($3n + 1$ conjecture).
- Hilbert's 10 problem: given a polynomial with integer coefficients, find if it has an integer solution.