CS221: Computational Complexity Prof. Salil Vadhan

Lecture 2: Turing Machines,Computational Problems, and Complexity Measures

9/20 *Scribe: Noam Zeilberger*

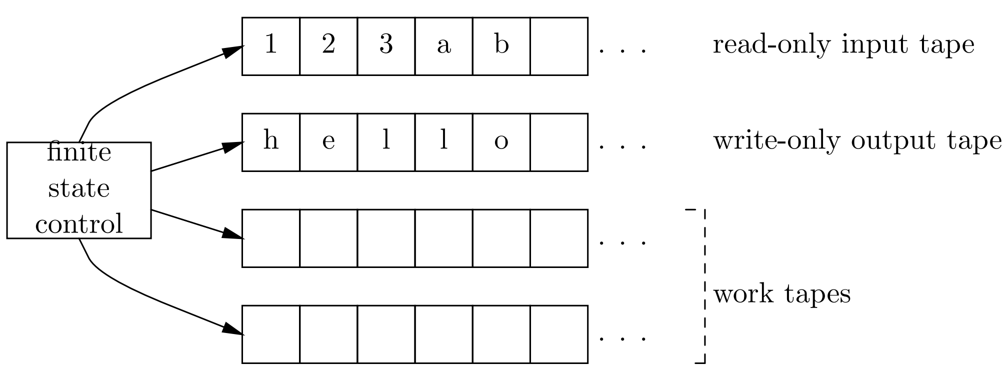
Contents

1 Announcements

- There will be no class on Monday.
- Professor Vadhan's office hours next week are Tuesday 10-12, 1-4.
- Reading: Papadimitrou §2, §7.2 (can also scan §1,§3)

2 Turing Machines

To measure complexity, we must fix a precise model of computation. The choice is inessential, but we shall use multitape Turing Machines (TMs), with the following organization:



At the start, the input tape contains the input string followed by blanks, while all other tapes are fully blank. The TM has a special halt state so that we can define the output of a TM M on input x as follows:

$$M(x) = \begin{cases} \textit{contents of output tape} & M \text{ halts on input } x \\ \perp & \text{otherwise} \end{cases}$$

Formally, a TM is defined as a quadruple $M = (K, \Sigma, \delta, s)$, where K is the set of states, Σ is the alphabet, δ is the transition function (i.e. a function from states to states) and s is the start state. (We will rarely refer to this formalism.)

3 Computational Problems

The most general type of computational problem is the *search problem.* For a given function $S : \Sigma^* \rightarrow$ finite subsets of Σ^*, M solves S if $\forall x.M(x) \in S(x)$.

There are special cases of the search problem:

- if $\forall x.|S(x)| = 1$, then the TM must compute a function.
- if $\forall x.S(x) \in \{\{0\}, \{1\}\}$, then the TM must decide a language, i.e. compute its characteristic function.

3.1 Example: Satisfiability

Let φ be a formula in conjunctive normal form (CNF) (e.g. $\varphi(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3 \vee \neg x_3)$). φ is satisfiable if there exists a truth assignment $\vec{x}$ such that $\varphi(\vec{x}) = \textit{true}$. Satisfiability gives rise to the following computational problems:

- "decision" — $S(\varphi) = \begin{cases} \{1\} & \varphi \text{ satisfiable} \\ \{0\} & \text{otherwise (o.w.)} \end{cases}$
- "search" — $S(\varphi) = \begin{cases} \{\text{satisfying assignments to } \varphi\} & \varphi \text{ satisfiable} \\ \Sigma^* & \text{o.w.} \end{cases}$
- "counting" — $S(\varphi) = \{\# \text{ satisfying assignments}\}$
- "approximate counting" — $S(\varphi) = \{N \mid .99N \leq \# \text{ satisfying assignments} \leq 1.01N\}$

You may recall from earlier courses that the decision problem is "hard" (specifically, **NP**-complete, by Cook's theorem) for formulas in CNF. All of the other problems listed are hard at least as hard as the decision problem because an algorithm for any of them can be used to solve the decision problem (e.g., if M solves approximate counting, then φ is satisfiable if and only if $M(\varphi) > 0$).

However, it's not true in general that all of these problems have the same complexity. In the case of DNF (disjunctive normal form, i.e. OR of ANDs), the decision and search problems are easy (examine each conjunction individually). On the other hand, the DNF counting problem is hard, as for each CNF formula φ there is a corresponding DNF formula $\neg\varphi$ (by de Morgan's law), and we have

$$\#\text{ satisfying assignments to } \varphi = 2^n - \#\text{ satisfying assignments to } \neg\varphi$$

It is not obvious, but the approximate DNF counting problem is easy.

We'll touch upon all of these types of problems (and many others) during the course.

4 Measuring Complexity

4.1 Time and Space

Time: the *running time* of M on input x is the # of steps M takes before halting on x, or else ∞ if M does not halt.

Space: the *space* used by M is the total # of cells M uses on its *work tapes* over the entire computation (i.e. not including the input and output). Note that this definition makes space $< n$ meaningful, as could be useful when we consider input coming from an external source (e.g. the Internet). (Also, the output size will often be insignificant for us as we deal primarily with problems in which the output size is polynomial in the input size, or even only a single bit, as in decision problems.)

4.2 The Extended Church-Turing Hypothesis

The Extended Church-Turing hypothesis states that every "reasonable" model of computation can be simulated by a TM with only a polynomial slowdown in running time and a constant factor increase in workspace. Here, "reasonable" can mean a variety of things:

- a model that matches our intuitive notions of computation/algorithms
- a model in which only a constant amount of "work" is done at each step
- any physically realizable model

The last notion is controversial, and the hypothesis might be contradicted by quantum computers, were it possible to build them.

Why do people believe the Extended Church-Turing hypothesis? First, all known algorithms can be implemented on TMs. Second, there are a number of simulation results, such as:

- a k-tape TM can be simulated by a 1-tape TM with quadratic slowdown (i.e. time $t \to t^2$)
- a k-tape TM can be simulated by a 2-tape TM with logarithmic slowdown (i.e. time $t \to t \log t$)
- the RAM model can be simulated by a TM with cubic slowdown

4.3 More about TMs

Other important facts about TMs are:

- Every TM can be encoded as a (not necessarily unique) string in a manner such that every string represents some TM, and so the TMs can be enumerated, e.g. in lexicographic order.
- TMs can simulate each other, i.e. there exists a universal TM $U(M, x) = M(x)$, and the time of U on (M, x) is $O(|M|^2 \cdot (\text{time of } M \text{ on } x)^2)$, while its space usage is $O(|M| + (\log |M|) \cdot (\text{space of } M \text{ on } x))$.
- For any fixed set of inputs and outputs, there exists a TM that computes the function in linear time (by hardwiring a lookup table into the finite-state control). Hence, we are mainly interested in the *asymptotics* of computational problems.