Lecture 2

Review of Computability and Complexity

January 8, 1998
Lecturer: Richard Ladner
Notes: Jayram S. Thathachar

In this lecture we will review the standard literature on computability and complexity. The notation used is almost consistent with Papadimitriou's book.

2.1. Turing Machines

A (deterministic) *Turing machine* T is a computing device with a finite instruction set and a head reading a one-way infinite tape (extending to the right) whose squares are filled with symbols from some appropriate alphabet. The tape contains a finite number of non-blank symbols appearing in a finite segment of the left portion of the tape. The machine operates in steps. In each step, the head reads the symbol from the square it is positioned on, possibly changes it, and then possibly moves right or left, all of which is done according to the finite instruction set. Formally, it can be described as a a 5-tuple $(K, \Sigma, \Gamma, \delta, s)$, where

- K is the set of states,
- Σ is the input alphabet,
- $\Gamma \supseteq \Sigma$ is the tape alphabet which includes the end-marker symbol (present in the leftmost square of the tape) and the blank symbol,
- s is the start state, and
- $\delta : K \times \Gamma \to (K \cup \{Y, N, H\}) \times \Gamma \times D$ is the transition function describing the operation of the machine in one step. Here $D = \{L, R, S\}$ denotes the direction moved, and $\{Y, N, H\}$ denotes the set of halting states (Y is accept, N is reject and H is halt without accept or reject).

We will adopt certain conventions about the operation of the Turing machine that make our descriptions easier, e.g. the Turing machine is not allowed to move off the left end of the tape.

A *configuration* of the Turing machine at any instant is a description of the contents of the tape, head position and state at that time. Formally it is given by the tuple (q, u, v), where q is the state, and u and v denote the contents of the tape to the left and right of the head. We will make v finite by stopping at the square after which there are only blank symbols on the tape. The notation $C \xrightarrow{T} D$ describes the move from configuration C to D in one step according to δ, and $C \xrightarrow{T^m} D$ describes a move of m steps.

The language accepted by T is the set of input strings in Σ^* which causes T to end up in the accept state Y. In other words, $x \in L(T)$ if and only if $\text{Init}(x) \xrightarrow{T^*}$ Accepting configuration. The (partial) function computed by T on any string is given by the tape contents if and when T ends in the halt state H on that string.

We can generalize the one-tape model to multi-tapes, in which case we can distinguish between input tapes, which are read-only, output tapes, which are write-only and "move right-only and add symbol", and work-tapes, which have no restrictions. Another generalization is a *non-deterministic Turing machine* for which the transition is described as a relation

$$\Delta \subseteq K \times \Gamma \times (K \cup \{Y, N, H\}) \times \Gamma \times D,$$

thereby allowing multiple potential moves in a single step.

2.2. Computability

Some definitions on types of languages:

Decidable: Language for which there is a Turing machine that halts on all inputs in Σ^* and accepts those and only those strings that are in the language. Also known as *computable.*

Recursively Enumerable (r.e.): Language for which there is a Turing machine that accepts a string if and only if it is in the language.

We can also describe Turing machines by strings which allows a *universal* Turing machine to be constructed that can simulate any Turing machine on any input. Thus the language

$$K = \{x : \text{Turing machine described by } x \text{ halts on input } x\}$$

is r.e. However,

Proposition 2.1: K is undecidable.

The key notion for proving undecidability results is *reduction.* In the simplest form we have *many-one* reduction, where $L \leq_m L'$, if there is a computable function f such that $x \in L$ if and only if $f(x) \in L'$. More generally, we can have *Turing reduction* $L \leq_T L'$. This means that there is a Turing machine deciding L that has an *oracle* tape that can answer questions about membership in L'. (The many-one reduction is a special case where the Turing machine just computes $f(x)$ and asks just one question about the membership of $f(x)$ in L' and halts with the same answer that it obtains on the oracle tape.)

The *Church-Turing* thesis states that any finite computing device is no more expressive than Turing machines. This includes RAMs, PRAMS, supercomputers with gazillions of processors, and the abacus.

2.3. Complexity

Time Complexity

Some more definitions:

TIME($t(n)$) denotes the set of languages that are decidable by Turing machines that always halt in at most $t(n)$ steps on inputs of length n. Similarly, NTIME($t(n)$) for non-deterministic Turing machines. (This means that on any string, every computation path halts in time $t(n)$ and at least one of them ends in an accepting configuration if and only the string is in the language.) The two famous complexity classes are

$$\begin{aligned} P &= \bigcup_k \mathrm{TIME}(n^k) \\ NP &= \bigcup_k \mathrm{NTIME}(n^k), \end{aligned}$$

and the big open question is

Does $P = NP$?

Similar to computability, there are analogous notions of reduction for complexity classes. Thus, we have many-one *polynomial-time* reduction, where $L \leq_m^P L'$ if there is a computable function f running in polynomial time such that $x \in L$ if and only if $f(x) \in L'$. This is a useful technique for showing that languages are equally hard (or easy) because $L' \in P \Rightarrow L \in P$ and $L' \in NP \Rightarrow L \in NP$.

Space Complexity

Here we deal with multi-tape Turing machines having input and work tapes. The space used in a computation is the total amount of cells used in all of the work tapes. (We don't count the input tape because we can now talk about sub-linear space.) Thus, we can define SPACE($s(n)$) and NSPACE($s(n)$). We define $PSPACE = \bigcup_k SPACE(n^k)$, $L = SPACE(\log n)$ and $NL = NSPACE(\log n)$

Analogously, we have *log-space* reductions, where $L \leq_m^{\log} L'$ if there is a computable function f using logarithmic space such that $x \in L$ if and only if $f(x) \in L'$.

Relations between Complexity Classes

For $t(n) \geq n$, and $s(n) \geq \log n$, we have

Proposition 2.2:

$$\mathrm{TIME}(t(n)) \subseteq \mathrm{NTIME}(t(n)) \subseteq \bigcup_c \mathrm{TIME}(2^{ct(n)})$$

$$\mathrm{SPACE}(s(n)) \subseteq \mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s^2(n))$$

For the sake of motivating these results, we prove below one of the above results, namely $\mathrm{NSPACE}(s(n)) \subseteq \mathrm{SPACE}(s^2(n))$, also known as *Savitch's Theorem*.

Proof: Let T be a $s(n)$-space bounded non-deterministic Turing machine.

We define a *m-configuration* as (q, i, j, α), where q is the state, i is the input head position, j is the work-tape head position and α is the contents of the work tape which has exactly m cells used. Thus, $q \in K$, $0 \leq i \leq n+1$, $0 \leq j \leq m$, and $\alpha \in \Gamma^m$. Note that the length of a m-configuration is $O(\log n + m)$.

For C and D which are m-configurations, let $\mathrm{REACH}(m, C, D, k) = \text{true}$ if and only if $C \xrightarrow{T \leq 2^k} D$ and every intermediate configuration is a m-configuration. In m amount of storage, T makes no more than 2^{cm} steps, where c is independent of m. Therefore, x is accepted by T if and only if $\mathrm{REACH}(s(|x|), \mathrm{Init}_{s(n)}(x), A, c \cdot s(|x|)) = \text{true}$, for some accepting configuration A, where Init_m is the initial configuration with exactly m work tape cells assigned blank.

The proof will be completed in the next lecture. $\square$