

CMPT 710/407 - Complexity Theory

Lecture 2: Languages and Turing machines

Valentine Kabanets

September 6, 2007

1 Review

1. Problems and Languages
2. Turing machines
3. Reductions
4. Complexity classes
5. Completeness

1.1 Problems and Languages

A *function* problem is a function from strings to strings

$$f : \Sigma^* \rightarrow \Sigma^*$$

For example, given an integer, find its prime factorization; or, given a graph G and two vertices s and t , find a shortest path from s to t in G .

A *decision* problem is a function from strings to Boolean values

$$f : \Sigma^* \rightarrow \{yes, no\}$$

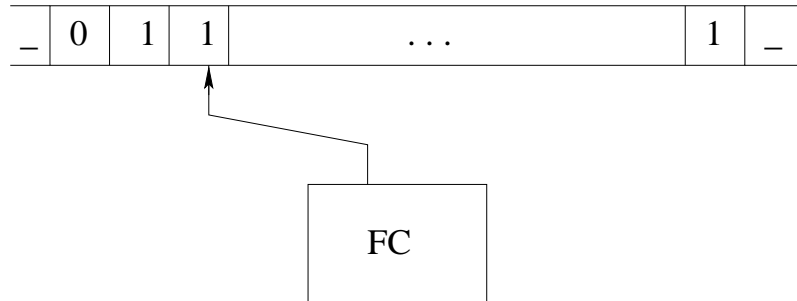
For example, given integer, is it prime?

A *language* associated with a given decision problem f is the set $\{x \in \Sigma^* \mid f(x) = yes\}$. For example, $SAT = \{\text{satisfiable Boolean formulas}\}$.

1.2 Turing Machines

A Turing machine (TM) consists of an infinite tape, divided into cells; each cell can hold one symbol from a given alphabet Σ . We denote the blank symbol by “-”, and assume it is contained in Σ . TM has a finite control (FC) that can be in any of (finitely) many states from the set Q . There are three special states: $q_{start}, q_{accept}, q_{reject}$ in Q . The behaviour of

Figure 1: Turing machine.



a TM is specified by the transition function $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, -\}$. A pictorial representation of a Turing machine is given in Fig. 1.2.

Computation of TM: TM starts in q_{start} scanning the leftmost symbol of the input, and proceeds according to δ . It accepts by entering q_{accept} , and rejects by entering q_{reject} . TM may also run forever (i.e., loop).

A k -tape Turing machine (TM) consists of k infinite tapes. Usually, one tape is the input tape; one tape is the output tape; the remaining $k - 2$ tapes are work tapes. The transition function is $\delta : Q \times \Sigma^k \rightarrow Q \times \Sigma^k \times \{L, R, -\}^k$.

Examples:

1. Consider a language $PARITY = \{x \in \{0, 1\}^* \mid x \text{ has an odd number of 1s}\}$. It is very easy to define a 1-tape TM (in fact, a finite automaton) that accepts exactly this language.
2. Consider $PAL = \{x \in \{0, 1\}^* \mid x = x^R\}$ (a language of palindromes). There is a simple 1-tape TM accepting PAL , with running time $O(n^2)$, where n is the input length. (The idea is to match the first and the last symbol of the input; if they are different, stop and reject; otherwise, erase these two symbols, and recursively continue with the remaining (shorter) string.) If we allow ourselves to use a 2-tape TM, we can decide PAL in time $O(n)$, linear time! (Can you see how?) Is it possible to invent a 1-tape TM for PAL with running time better than quadratic? It turns out the answer is No (as you'll be asked to show in the homework).

We have the following result (whose proof is left as an exercise).

Fact 1. *A k -tape TM can be efficiently simulated by a one-tape TM (with only a polynomial, in fact quadratic, slowdown). That is, if a language L can be decided by a k -tape TM in time $t(n)$, then L can also be decided by a 1-tape TM in time $O(t^2(n))$.*

Note that thanks to the quadratic lower bound for deciding PAL on a 1-tape TM, the quadratic slowdown stated in the Fact above is optimal. That is, it is impossible in general to convert any k -tape TM to a 1-tape TM with less than quadratic slowdown. (Do you see why?)

TM programming techniques: Here are some basic techniques useful when programming Turing machines.

1. Use a state to remember some information about the input (e.g., for PAL, we can use a state to “remember” the first symbol of the input so that we check if it matches the last symbol, once we get to the last symbol). More formally, this means extending the set Q of states with special states that remember some finite amount of information.
2. Partition the tape into a finite number of “tracks”. More formally, define the tape alphabet Γ to consist of k -tuples of symbols (a_1, \dots, a_k) , where we think of a_i as the symbol contained on track i of the tape. For example, we can use k tracks to contain the information in k tapes of a given k -tape TM, when performing a transformation from k -tape to 1-tape TM.
3. Assume that we can label every symbol a on the tape with $*$ (or any other marker), so that we have a^* . For example, we can label those symbols which are currently scanned by the k -tape TM, when we are simulating that k -tape TM with a 1-tape TM. Formally, this just means that we extend our tape alphabet Γ with new symbols a^* for every $a \in \Gamma$.

There are many other models of computation (multi-head TMs, multi-dimensional tape TMs, computers with RAM, etc.) All of them can be simulated by a 1-tape TM with only polynomial slowdown. This means that if we ignore polynomial speedups, any efficient (polytime) computation on any reasonable model of computation can be performed by a polytime 1-tape TM.

Recall that the Church-Turing thesis says that a Turing machine captures exactly the intuitive notion of an algorithm. That is, anything that can be algorithmically solved (in the intuitive sense of “algorithmically”) can also be solved by a Turing machine.

An extension of the Church-Turing thesis is as follows:

Extended Church–Turing Thesis *Everything efficiently computable on a physical computer (in the intuitive sense) is efficiently computable by a Turing machine (in the formal sense).*

Quantum computers pose a potential challenge to this thesis, but they haven’t been built yet.