CS512 LECTURE NOTES - LECTURE 20

1 Introduction to the Theory of Computation

1.1 Basic Definitions

Alphabet: Σ is a set of symbols. For example, if we are talking about binary numbers, $\Sigma = \{0, 1\}$.
Language: L is a set of strings over a given alphabet Σ
Regular operations (on languages):

- Union: $(L_1 \cup L_2)$. Usual set union of two languages
- Concatenation: $L_1 L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$.
  For example, if $L_1 = \{aa, ba\}$, $L_2 = \{aba, bb\}$, then $L_1 L_2 = \{aaaba, aabb, baaba, babb\}$
- Kleene Star: $L^* = \cup_{n=0}^{\infty} L^n$. Notice that $L^n = L^{n-1} L$.
  Example: if Σ is our alphabet, then Σ^* is the set of all possible strings over the alphabet.
  We can define then a language L over Σ^* as $L \subseteq \Sigma^*$

Machine

A machine (in theory of computation) takes as input a string $x \in \Sigma^*$ and produces two possible outputs: accept, or reject.

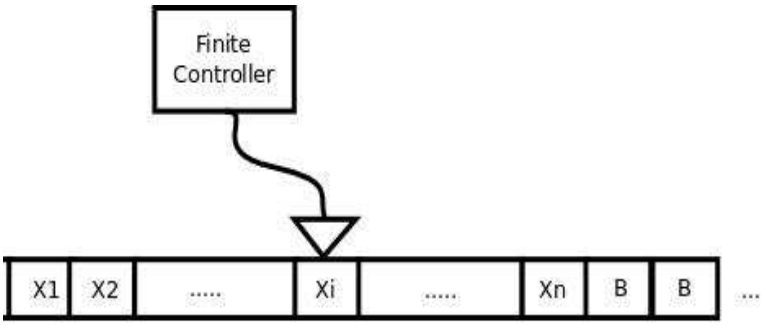
For example, we can have a machine M_P that takes as input a string x over $\Sigma = \{0, 1, 2, \ldots, 9\}$, and accepts whenever the given string represents a prime integer and rejects whenever the string does not represent a prime integer.
Definition: The language of a machine is the set of strings that are accepted by a given machine M, formally

$$L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}$$

1.2 Turing Machines

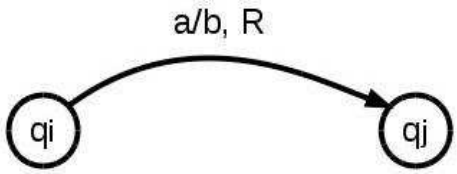
A Turing Machine has a "Finite Control" (since it has a finite number of states) that has access to an infinite tape, where symbols can

be read from and written to. It has a head that points to current position on the tape, but the head can only move one position to the right or one position to the left.



The transitions of this machine are as follows:

- Take as input a state q_i and the symbol a from the tape.
- Given (q_i,a) the machine does the following:
  - Writes a new symbol on the tape (where the head is currently pointing to)
  - Moves the head one position to the right or one position to the left.
  - Changes to another state q_j



The machine starts in an initial configuration given by the following:

- The finite control is in the start state.
- The tape contains the input string, starting from position 0. Even though the tape is infinite, it contains only Blanks after the input string.
- The head is pointing to position 0 of the tape.

These Machines are called **TURING MACHINES**.

1.2.1 Formal Definition of a Turing Machine

A Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f)$
where

- Q: Finite set of states
- Σ: Input alphabet.
- Γ: Tape alphabet. This alphabet might include other symbols not included in the input alphabet that are used by the internal operation of the Turing Machine. In particular the blank symbol Δ is in the tape alphabet, since after the input, the tape is full of Δs.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: a transition function that describes the operation of the Turing Machine.
- q_0: the start state
- q_f final state

Notice that a Turing Machine might do the following:

- Accept: stops in the final state
- Reject: stops because it would need a transition not given in the transition function
- Loop: keep going forever without ever stopping.

Example

Design a Turing Machine that accepts the language of n a's followed by n b's:

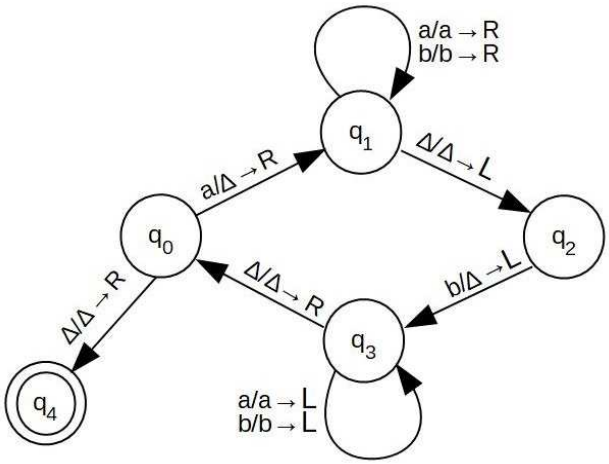
$$L = \{a^n b^n | n \geq 0\}$$

We can write a pseudocode description of our Turing Machine:

1. if symbol = a, replace it with a blank (Δ) and move Right (else reject)
2. move right over a's and b's until a Δ is found
3. move left (now the head should be pointing to the last b)

4. if symbol = b, replace it with Δ and move Left (else reject)
5. move left over a's and b's until a Δ is found
6. move right (now the head should be pointing to the first a, or Δ if no a's or b's left)
7. if symbol=Δ then accept
8. otherwise, repeat the process all over again from step 1

Using this algorithm we can design the following machine:



1.2.2 Computation

Let us trace the computation of this Turing Machine on input *aabb*.

In order to trace the computation we again use instantaneous configurations. In the case of a Turing machine an instantaneous configuration includes the contents of the tape before the position of the head w_1 the current state q_i, and the contents of the tape starting from the position of the head w_2:

$$w_1 q_i w_2$$

We can write the contents of the *computation* on a table where each row corresponds to an instantaneous configuration. The number of rows is the total number of operations performed by the Turing Machine where the last row is an accepting configuration (including a final state).

Example

Let us trace the computation of the Turing Machine of the previous example on input *aabb*.

q_0	a	a	b	b	Δ
Δ	q_1	a	b	b	Δ
Δ	a	q_1	b	b	Δ
Δ	a	b	q_1	b	Δ
Δ	a	b	b	q_1	Δ
Δ	a	b	q_2	b	Δ
Δ	a	q_3	b	Δ	Δ
Δ	q_3	a	b	Δ	Δ
q_3	Δ	a	b	Δ	Δ
Δ	q_0	a	b	Δ	Δ
Δ	Δ	q_1	b	Δ	Δ
Δ	Δ	b	q_1	Δ	Δ
Δ	Δ	q_2	b	Δ	Δ
Δ	q_3	Δ	Δ	Δ	Δ
Δ	Δ	q_0	Δ	Δ	Δ
Δ	Δ	Δ	q_4	Δ	Δ

The final configuration is an accepting configuration since q_4 is the final state, and the string *aabb* is accepted by this Turing Machine.

1.3 Types of Problems

- **Decision problems** are those for which the answer is either YES or NO. For example, is n prime?
- **Optimization problems** are those for which there is a quantity that must be minimized or maximized. For instance, the Max Flow problem discussed before is an optimization problem since we must find the value of the maximum flow.

  Notice that an optimization problem only answers a number (the max flow, for example), but does not give the actual flow function.
- **Search problems** are those that return the actual structure that maximizes or minimizes the objective value. For instance,

in the case of max flow, the answer would be the flow function that maximizes the flow value.

There is an easy way to rephrase an optimization problem as a decisions problem. For example, in the case of the Max Flow problem we can rephrase it as a decision problem:

Given a network $G = (V, E)$, a capacity function $c : E \to \mathbb{N}$, and a positive integer $k \in \mathbb{N}$.
Is there a flow f such that is value $v(f)$ is equal to k?

Notice that it is also possible to solve the optimization version of the problem if we can solve the decision version, by using a strategy similar to binary search.

Suppose that we have an algorithm `Alg(`I, k`)` that is able to solve the decision version of a maximization problem Π given an instance I of Π, i.e. `Alg(`I, k`)` will return `true` iff there is a solution for instance I of Π with value equal to k.

We want to use `Alg(`I, k`)` to solve the optimization version of the problem `Opt(`I`)`:

Algorithm `Opt(`I`)`
$k = 1$;
while `Alg(`I, k`)`
    $k = 2k$
if $k = 1$ return 0
$first = k/2$
$last = k$
while $(first < last)$
    $k = \lfloor \frac{first+last}{2} \rfloor$
    if `Alg(`I, k`)`
        $first = k$
    else $last = k$
return $last$

In our discussion of complexity classes we are going to assume that the problems we are talking about are decision problems, since it makes it easier to relate to Turing Machines. We can say that a Turing Machine accepts and instance of a decision problem if it answers YES. We will also say that a Turing Machine M decides (or solves) a problem (language) iff for every instance of the problem,

M answers either YES or NO. In what follows we will only talk about decidable decision problems (languages).