COL352 Lecture 2

Contents

1 Goal of this lecture

To mathematically define a computational problem

2 Deterministic Finite Automata

Notation 1. *An alphabet is a finite set Σ of symbols.*

Notation 2. *By Σ^n, we the set of strings of length n constructed from symbols in Σ.*

When $n = 0$, just to maintain consistency, we define ϵ to be an empty string, and $\Sigma^0 = \{\epsilon\}$.

Notation 3. *Σ^* is defined to be $\bigcup_{i=0}^{\infty} \Sigma^i$.*

Claim 1. *The set Σ^* is in fact countably infinite.*

Proof. Note that there are precisely $|\Sigma|^n$ strings of length n.

Consider any ordering of strings in Σ^n and map one string each to each element in the set $\{1+\sum_{i=0}^{n-1} |\Sigma|^i, \ldots, \sum_{i=0}^{m} |\Sigma|^i\}$ (it helps to visualize this as chunks of the integer number line).

Note that the range is a partition of $\mathbb{N}$, and thus, this map is a bijection from Σ^* to $\mathbb{N}$, whence we are done.

Constructing an injective map from Σ^* to $\mathbb{N}$ is enough though. ∎

Definition 1. *A language over Σ is any subset of Σ^*.*

Notation 4. *If x, y are strings in Σ^* (i.e. over Σ), we denote by $x \cdot y$ (or more simply, xy), the concatenation of x and y.*

Notation 5. *If L_1, L_2 are languages over Σ, then by $L_1 \cdot L_2$ (or more simply L_1L_2) we denote the set $\{s_1s_2 \mid s_1 \in L_1, s_2 \in L_2\}$.*

Definition 2. *Let Γ, Σ be any two finite non-empty sets. A computational problem is defined as a function $f : \Gamma^* \to \Sigma^*$.*

Example 1. *Suppose $\Sigma = \{0, 1, \ldots, 9\}$, and $\Gamma = \{0, 1, \ldots, 9, \times\}$. Define $f(x)$ to be the prime factorization of a number here (lexicographically maximal, with least length), considering 0 to be a prime. Then f is a computational problem.*

Example 2. *Let $\Sigma = \Gamma = \{0, 1\}$. Define*

$$f(x) = \begin{cases} 1 & \text{if } x \text{ represents the adjacency matrix of a connected graph in row major order} \\ 0 & \text{otherwise} \end{cases}$$

Then f is a computational problem.

Example 3. *A Python program is also a computational problem (output is the set of all possible bytecodes and syntax errors)*

Definition 3. *A decision problem is a function from* Σ^* *to* $\{0,1\}$.

Example 4. *The second last example in the previous definition is a decision problem.*

Associate with any decision problem $f : \Sigma^* \to \{0,1\}$ the language $L = \{x \in \Sigma^* \mid f(x) = 1\}$.
Conversely, we can associate a decision problem with every language (does x belong to the language L?).
Hence decision problems are equivalent to languages (in some sense).

Example 5. $\Sigma = \{0,1\}$, $f(x) = 1$ *if* x *is the binary representation of a number divisible by* 5*, and 0 otherwise.* $L = \{x \in \Sigma^* \mid f(x) = 1\}$.

How do we solve this problem?

Suppose $x = x[1 \dots n]$. Then we can just keep track of $x[1 \dots i] \pmod 5$ as follows:

```
function Solve(x[1...n])
    q ← 0
    for i ∈ {1...n} do
        if x[i] = 0 then
            q ← 2q (mod 5)
        else
            q ← 2q + 1 (mod 5)
        end if
    end for
    if q = 0 then
        return True
    else
        return False
    end if
end function
```

Another way of looking at this is to construct a node for all possible values of q, the set of which is $\mathbb{Z}_5$.

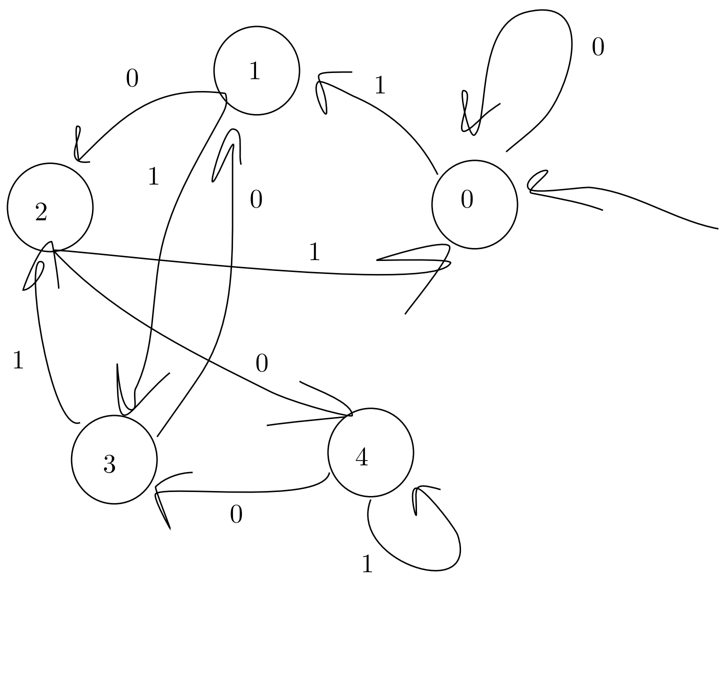

Figure 1: Automaton for the modulo 5 check

Example 6. $\Sigma = \{a,b\}, L = \{x \in \Sigma^* \mid x \text{ ends in } ab\}$.

Idea: keep track of last two characters used.

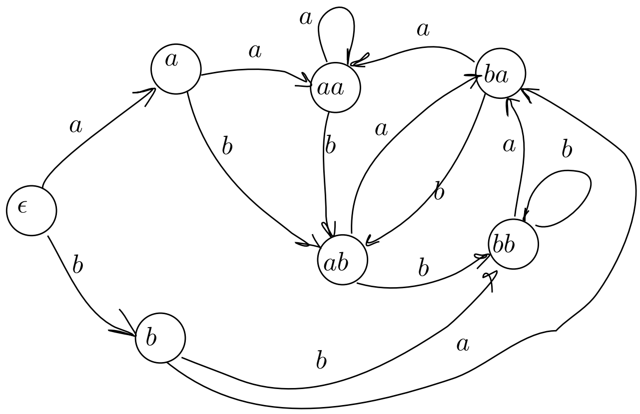

Figure 2: Automaton for checking last two characters

Definition 4. *A deterministic finite automaton (DFA) is a 5-tuple* $(Q, \Sigma, \delta, q_0, A)$ *where*

- Q *is a finite nonempty set, called the* *set of states.*
- Σ *is a finite nonempty alphabet.*
- δ *is a function* $\delta : Q \times \Sigma \to Q$*, called the* *transition function.*
- q_0 *is the* *initial state.*
- A *is the set of all* *accepting states.*