TOC Week 1

May 2021

1 Introduction to Computation

There is no formal definition of computation, though it's notion is clear to most of us. To say, Computation is doing calculations, solving problems using processes/systems of nature that work in some mechanical way. By mechanical, you might feel like gears moving, shafts turning and giving some result, yes it's same idea, we'll precisely formulate it when time comes. You may ponder what processes one can use for computing. Well, not all, we know the universe is doing infinitude of computations as we blink, it changes it's state gets to other, there are so much varibles involved that it is not possible to account for them. Also since we don't have a control over such process, we can just be an audience to it, and can't make it do calculations for us, or can we? Who knows.

Contrast this with the history, first computing devices developed were mechanical with gears, (abacus we can't say is a computing device, it's just a tool to aid in performing simple calculations, we only have to operate it), then as people learnt to control electronic devices, they saw their possible use as computing devices, and here we are sitting in front of the devices working on same principle. Richard Feynman said in first conference on Physics and computation held by IBM and MIT - "We can only think of using nature's computational powers to their full potential if we interact with it the way it works, according to the laws of Quantum Mechanics"[1]

2 Problems and Computational Model

Before starting, we should discuss the problems we would want our computing machine to solve. We might ask it to calculate the shortest path between two locations in a city(obvioulsy that path must consist of roads and strees). But for now we focus on decision problems, where on some input the machine either accepts it or rejects it. That's basically a Boolean function we are asking our machine to compute.

We will see one particular type of machine to start with, later we'll see that it doesn't matter much. If a problem can be "efficiently" solved by a machine of one type, it can be efficiently solved by that of other type as well, doesn't matter if one of them is using 9th gen intel, and other one a pentium processor.

3 Terminology

Alphabet : A non empty finite set, it's elements are called symbols.
String : A finite sequence of symbols.
Language : A finite set of strings.

4 Finite Automata and Languages

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the states,
2. Σ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition
4. $q_0 \; \epsilon \; Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

How does this machine work(One Pass machine)?

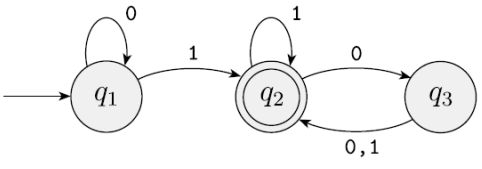

Figure 1: A finite automata

[1] He said something similar, though not exactly this :)

As we can see, this type of a machine computes a Boolean function

$$f : \Sigma^* \to \{TRUE, FALSE\}$$

(Σ^* is the set of all strings over Σ)
For a machine computing some boolean function f, we can equivalently say that the machines decides the language $L = \{x\epsilon\ \Sigma^* | f(x) = TRUE\}$ A language is called **regular** if there exists some machine that decides(recognizes) it.

5 NonDeterminism

The idea of non-determinism is similar to that of parallel computation, though not same. Till now on a pair of state and input, the next state was unique (deterministic), i.e. the transition function $\delta : Q \times \Sigma \to Q$ had one state as output. Now it is a function with output a set of states $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, it has a number of states to go into, and it makes a copy of itself and each of the copy then goes into one state. The language now gets modified to $\Sigma_\varepsilon = \Sigma \cup \varepsilon$, empty string.

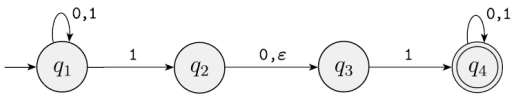

Figure 2: A non-deterministic finite automata

A non deterministic finite automata can be converted to an equivalent deterministic automata(first encounter with equivalence of two models). The set of languages recognized by them is the same.

6 Regular Operations

We define some operations over languages as: For two languages A and B,

- **Union** $A \cup B = \{x \mid x\ \epsilon\ A\ or\ x\ \epsilon\ B\}$
- **Concatenation** $A \circ B = \{xy \mid x\ \epsilon\ A\ and\ y\ \epsilon\ B\}$
- **Star** $A^* = \{x_1x_2x_3 \ldots x_l \mid x_i\ \epsilon\ A\ i\ \epsilon\ [l]\ \ l \geq 0\}$

Why are we doing all this? It turns out that the class of regular languages is closed under these operations.

Idea for closure under union : Simulation of automata Closure under operations : Using Non deterministic finite automata

7 Regular Expressions

Say that R is a regular expression if R is
1. a for some a in the alphabet Σ,
2. ε,
3. ϕ,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. R_1^*, where R_1 is a regular expression.

A language is regular iff some regular expression describes it.

Can all languages be represented by some finite automata and/or some regular expression? The first thing to be clear is that these two are the same things, a language represented by a finite automata is regular, so it will a regular expression. The question itself has its answer, only regular languages can be represented by some finite automata or regular expression.
Try this : $\mathcal{L} = \{0^n1^n \mid n \geq 0\}$

8 Context Free Grammars

It will be easier to understand grammar through an example: Below we give three rules to generate strings for a grammar say G.

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

Strings that can be generated from the grammar are - #, 0#1, 00#11.. You can see that the Language generated by this context free grammar(context free language) will be $\mathcal{L} = \{0^n\#1^n \mid n \geq 0\}$