# Contents

# 1  01.21.21

## 1.1  Deterministic Finite Automata

- We know that $\delta = Q \times \Sigma \to Q_2$

- We want a function that takes a starting state and a string, then returns the state after the machine has read that string

- Let's define $\delta^* = Q \times \Sigma^* \to Q$

  - $\delta^*$ takes a state and a string
  - $\delta$ takes a state and a symbol

- Now, we need a recursive definition

  - Base case:
    * Let $q_i \in Q$
    * $\delta^*(q_i, \epsilon) = q_i$
  - Recursive step:
    * If $q_i \in Q$, $w \in \Sigma^*$, and $c \in \Sigma$
    * then $\delta^*(q, w \cdot c) = \delta(\delta^*(q_i, w), c)$

## 1.2  Aside: On $\Sigma$ and $\Sigma^*$

- $\Sigma^*$ is the universe of all strings over $\Sigma$

  - $\Sigma = \{0,1\}$
  - $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 11, 10, 000, \dots\}$

- We can see this recursively

  - Base step: $\epsilon \in \Sigma^*$
  - Recursive step:
  - Let $w \in \Sigma^*$, let $c \in \Sigma$
  - Then $w \times c \in \Sigma^*$

- We can see this recursion graphically

| w | c | w · c | step |
|---|---|---|---|
| $\epsilon$ | | - | base |
| $\epsilon$ | 1 | 1 | recursive |
| 1 | 0 | 10 | recursive |
| 10 | 1 | 101 | recursive |

## 1.3 Recursively Testing 101

- Solve $\delta^*(q_1, 101)$

- $\delta(\delta^*(q_1, 10), 1)$

- $\delta^*(q_1, 10)$

    - $\delta(\delta^*(q_1, 1), 0)$

# 2 01.19.21

## 2.1 Tuples & DFAs

- Tuples are sequences which are always finite in length

- The deterministic finite automaton shown is a 5-tuple:

    1. Q: finite nonempty set of states
        - state: configuration of logic of a machine
    2. $\Sigma$ (Sigma) - input alphabet
        - alphabet: a finite, nonempty set of symbols where symbols are an object of length 1
    3. $\delta$ (Delta) - transition function
    4. $Q_0 \in Q$ - starting state
    5. $F \subset Q$ - set of final states

- For this deterministic finite automaton,

    - $\delta: Q \times \Sigma \rightarrow Q_2$

    Represented as a table,

    | Step | State | Input | Transition |
    |------|-------|-------|------------|
    | 1 | $Q_1$ | 1 | $Q_1 \rightarrow Q_2$ |
    | 2 | $Q_2$ | 0 | $Q_2 \rightarrow Q_1$ |
    | 3 | $Q_1$ | 1 | $Q_1 \rightarrow Q_2$ |
    | 4 | $Q_2$ | 1 | $Q_2 \rightarrow Q_2$ |

## 2.2 Domains & Codomains

- Domain: set of all possible function inputs

- Codomain: set of all possible outputs

3

## 2.3   Strings

- In computer science, strings are character arrays

- In mathematics, strings are sequences of symbols

- Specifically a string over an alphabet, $\Sigma$, is a sequence of symbols belonging to $\Sigma$

- $\epsilon$ is the empty string

- Concatenation: If $w_1, w_2 \in \Sigma$, $w_1 \cdot w_2 = w_1 w_2$

- If $c \in \Sigma$, then $\epsilon \cdot c = c \cdot \epsilon = c$

## 2.4   TODO Review Recursive Definitions

- Base step: a step that can not be broken down any further, a fact that is always true regardless of the input

- Recursive step:

- Defining the length of a string over $\Sigma$

    - Base: $|\epsilon| = 0$
    - Recursive:
        * let w be a string over $\Sigma$, and $c \in \Sigma$
        * then $|w \cdot c| = |w| + 1$

- Using this to define $|1011|$,

    1. $|1011| = |101 \cdot 1| = |101| + 1 =$
    2. $|10 \cdot 1| + 1 = |10| + 1 + 1 =$
    3. $|1 \cdot 0| + 1 + 1 = |1| + 1 + 1 + 1 =$
    4. $|\epsilon \cdot 1| + 1 + 1 + 1 =$
    5. $|\epsilon| + 1 + 1 + 1 + 1 =$
    6. $0 + 1 + 1 + 1 + 1 = 4$

## 2.5 Languages

- Languages over $\Sigma$ - a set of finite strings over $\Sigma$

- Langauges recognized by an automaton, M, L(M) is the language accepted by M

- $\emptyset$ is the empty language

- $\epsilon \neq \emptyset$

- $\epsilon \neq \{\epsilon\}$

- $\epsilon$ is not a symbol in any alphabet

# 3 01.14.21

## 3.1 Automaton (automata)

- Self running machine requiring a continuous power source

  - Historically used power sources include water, steam, and electricity

- Course revolves around defining the mathematics powering machines

## 3.2 The Mathematics of Automata

### 3.2.1 Mathematicians & History

- Cantor defines sets as collections of objects

- Cantor also argues that infinites can be of different magnitudes - there are infinitely more real numbers than natural numbers

- Goedel eventually derives his incompleteness theorem

  - No logical system that contains the natural numbers can prove its own soundness

  - Every sound logical system containing the natural numbers contains valid statements that cannot be proved or disproved

- In 1936, Turing proves The Halting Problem is not decidable, it is impossible

– The Halting Problem is an algorithm that can analyze any other algorithm and determine whether or not it goes into an infinite loop

• Turing creates the turing machine as an object consisting of sets and processes wherein the object can use any finite process to complete an action.

• Turing machine sets the basis for a computer, which leads to a series of important questions:

– What can & can't a machine do?

– What does it mean for a problem ot be harder than another?

– What does it mean for a machine to be more powerfule than another?

### 3.2.2 Sequential Logic

• Sentential Logic- based on boolean results

– Predicated on AND, OR, NOT

– XOR, XAND, etc. can be derived using the above

## 3.3 Necessary Review

• Textbook Ch. 0

• Logic Statements

• Set Theory

• Functions

## 3.4 Functions

• Functions - something that maps objects from one set to another

• Given f: a $\to$ b;

– Everything in a is mapped to something in b

  * For every x, such that x is an element of a, there exists a y, such that y is an element of b

6

- No one point in the domain can be mapped to two different points in the codomain
  * Logically, you can't have a function that takes in one input and returns two different outputs
  * If f maps x $\rightarrow$ y1 and $\rightarrow$ y2, y1 = y2

  -$\forall$ x $\in$ A $y_1,y_2 \in$ B $[f(x)=y_1 \land f(x)=y_2 \rightarrow y_1 = y_2]$

## 3.5 TODO Types of Functions - Definition & Logical Statement

- Injective Functions

- Surjective Functions

- Proof by Induction ($\forall$)

- Proof by Contradiction ($\neg\exists$)

## 3.6 Finite Automaton (Finite State Machine)

- States are logical confirgurations

- States are generally based upon input

- Purpose of a state machine is to make a yes/no decision