Lecture 1

Logistics

Class: CS 600 – "Theory of Computation"

Instructor: Prof. Evgenios Kornaropoulos

Who is this class for? Mainly Ph.D. students

Required Skill: Formal Reasoning. Ability to write/understand proofs.

Class Difficulty: ... hard? Computability + Complexity Theory

Resources: ① We are (mainly) following book:
- "Introduction to the Theory of Computation" Third Edition, Michael Sipser

Another useful book:
- "Computational Complexity: A Modern Approach" Sanjeev Arora, Boaz Barak

② Notes made in Latex (joint effort → Prof. Dov Gordon, Prof. Jon Katz)

③ Hand-written Notes (like this!)

④ Office Hours

Grading: Homework (5 sets): 25 % → every other week
Midterm : 30% (or 40%)
Final : 40% (or 30%)
Participation/Quiz: 5%

Closed Book
No notes
During class
Not cumulative

Communication: Slack

Intro: Mathematics + Computation *

The beginning ⇝ "On computable numbers, with an application to the Entscheidungsproblem", by A. M. Turing, 1936

An mathematical model of computation enabling rigorous definition of computational tasks, the algorithms to solve computational tasks, and the basic resources these require

The concept of computation revealed itself as a deep that illuminates other concepts and field in a new light

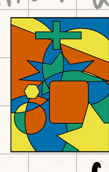
Theory of Computation progresses like any other mathematical field, researchers prove theorems and generalize, simplify and create variations based on their judgment and research taste.

The interaction between Mathematics and Computation predates Turing's work
"a mathematical understanding could solve any practical problem only through a computational process applied to the data at hand."
↳ Euclid's Greates Common Divisor algorithm was devised in 300 BCE

Interactions between Mathematics and Computation can be divided in four overlapping categories:

① Need of ToC to use general mathematical techniques.
In the beginning: logic, discrete mathematics
As the field matured: geometric techniques ⇝ Approximation algorithms
topological methods ⇝ Distributed systems
number theory
algebraic geometry ⇝ Pseudo-random objects

② Need of Mathematics to compute (i.e., use algorithms)
Software development and libraries with computational methods for algebra, topology, group theory, geometry statistics etc.
Also, programs for mathematical proof verification and proof discovery



↳ Four Color theorem proof has been partially generated by code

③ Some mathematical theorems guarantee the existence of a mathematical object. But, can the object guaranteed to exist be efficiently found?
Non-constructive existence proofs are philosophically interesting but of little practical use. Seeking constructive methods leads to deeper understanding of a field/problem.

* based on text by Avi Wigderson.

④ The study of computation leads to the production of new mathematical results, theorems, and problems.

Need both to analyze algorithms and to prove hardness results.
Devise new probabilistic concentration results, algebraic identities, statistical tests and more.

Computational Complexity Theory

Early on, ToC focused on understanding which computational problems can and which cannot be solved by algorithms.

But, this division turns out to be too coarse
↳ For many problems that can be solved in principle, the best algorithm to solve them won't terminate fast enough.

Thus, there was need for a much more refined theory that will account not only for whether a problem is solvable, but also for the performance of the solving algorithm

Computational Complexity Theory was born in 1960s with the goal of understanding efficient computation:

"Determine the minimal amounts of natural resources (time, memory, communication) needed to solve natural computational tasks by natural computational models."

Nowadays, computational modeling has expanded towards understanding more concepts such as: secret, proof, learning, knowledge, randomness interaction, evolution, strategy, synchrony etc.
↳ From "What can be efficiently computed?"
To "What can be efficiently proved?"
"What can be efficiently learned?"
"Can we effectively use natural sources of randomness?"

Important Principles: ① Computational Modeling: Define basic operations, information exchange/processing, and resources.
② Efficiency: Try to minimize resources used (study trade-offs)
③ Asymptotic Thinking: Study problems on large instances as structure often is easier to understand in the limit
④ Classification: Organize problems into classes according to the resources they require

⑤ **Reductions:** Ignore lack of understanding, assume that you can solve a problem and explore which other problems it would help solve efficiently

⑥ **Completeness:** Identify the most difficult problems in a complexity class

⑦ **Impossibility:** Abstract all known techniques used to solve a problem and argue that they will not suffice for its resolution

The Real Lecture 1

Basics (Sipser, Chapter 0.2, Section "Strings and Languages")

- Alphabet Σ: a finite set of characters. E.g., $\Sigma = \{0,1\}$
- Language L over Σ: a set of strings containing characters from Σ
  ↳ E.g., $L = \{0, 1, 11, 00\}$
- Empty string ε

* A language doesn't have to be finite. E.g., all binary strings ending in 0

Set Operators for Languages (Sipser, Chapter 1.1, Section "The Regular Operations")

- Union Operator: $L_1 \cup L_2 = \{x \mid x \in L_1 \lor x \in L_2\}$
- Concatenation Operator: $L_1 \| L_2 = L_1 \circ L_2 = L_1 L_2 = \{xy \mid x \in L_1 \land y \in L_2\}$
- Unary Operation ⇝ (Kleene) Star Operation L^*

$$L^0 = \{\Lambda\}$$
$$L^k = L L^{k-1} = \{xy \mid x \in L \land y \in L^{k-1}\}$$
$$\text{so, } L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$$

language

⚠ L^k not to be confused with a^k which is $\underbrace{aaa\dots a}_{k \text{ times}}$

character

For example, $L = \{01, 1\}$. Then $L^0 = \{\varepsilon\}$, $L^1 = \{01, 1\}$, $L^2 = \{0101, 011, 101, 11\}$
$L^3 = \{010101, 01011, 01101, 0111, 10101, 1011, 1101, 111\}$

so, $L^* = \{\varepsilon, 01, 1, 0101, 011, 101, 11, 010101, 01011, 01101, 0111, 10101, 1011, 1101, 111, \dots\}$

deterministic finite automata

We start with a simple model of computation and a simple class of languages.
↳ regular languages

Regular Languages (Sipser, Chapter 1.3, Section "Formal Definition of Regular Expressions")

- Recursive Definition: Let $\mathcal{R}$ be the set of all regular languages over alphabet Σ
  1. $\emptyset \in \mathcal{R}$ and $\{\varepsilon\} \in \mathcal{R}$
  2. $\forall \sigma \in \Sigma$: $\{\sigma\} \in \mathcal{R}$
  3. If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$ (closed under star)
  4. If $L_1 \in \mathcal{R}$ and $L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$ (closed under concatenation)
  5. If $L_1 \in \mathcal{R}$ and $L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$ (closed under union)

- Analogy ⇝ Operations '+', '-', '×' are used to define mathematical expressions where the output is a number
  Operations '||', '∪', '*' are used to define regular expressions where the output is a language

- Sometimes the union '∪' is denoted as '+'. Also, sometime we drop '{' and '}'
  So, the regular expression $(0+1)^*$ starts with language $L = L_1 \cup L_2$ where $L_1 = \{0\}$ and $L_2 = \{1\}$, and applies the $*$ operation.

- In regular expressions, the star operation is done first, followed by concatenation, and then union, unless parentheses change this order.

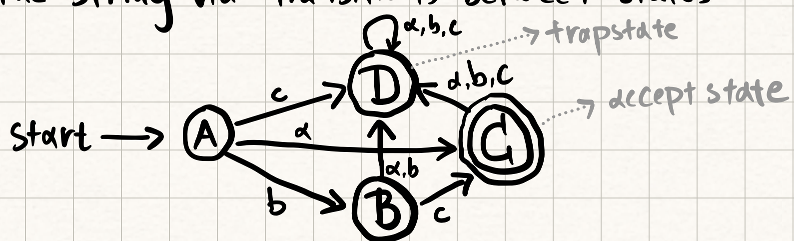
Deterministic Finite Automata (DFA) (Sipser, Chapter 1.1, Section "Formal Definition of a Finite Automaton" up to "The Regular Operations")

- A deterministic finite automaton is a state machine that takes as an input a string and outputs either "accept" or "reject"

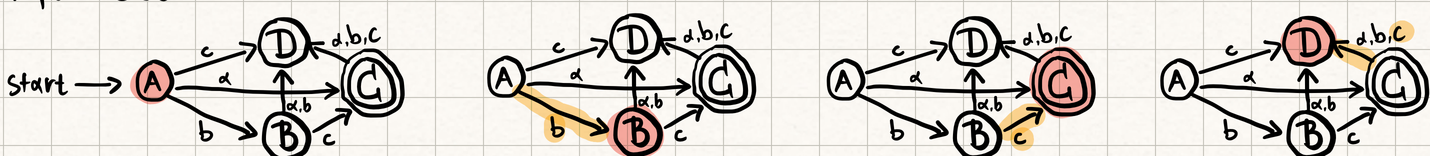
$$\text{string } w \longrightarrow \boxed{\text{DFA}} \longrightarrow \text{Accept / Reject}$$

- It processes the string via transitions between states

<u>Example:</u>



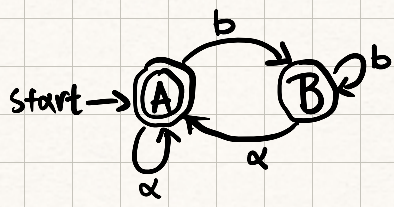
Input "bcc"



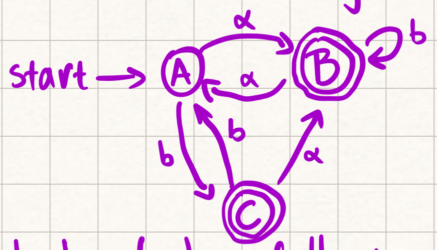
Language L_1 from regular expression $\varepsilon + (\Sigma^* \alpha)$ or more detailed $\{\varepsilon\} \cup (\{\alpha \cup b\}^* \| \{\alpha\})$

$L_1 = \{\varepsilon\} \cup \{\alpha, \alpha\alpha, b\alpha, \alpha\alpha\alpha, \alpha b\alpha, b\alpha\alpha, bb\alpha, \dots\}$

is "captured" by DFA:



Quiz 1.1: Given the following DFA,



which of the following statements is false

α) String "αbb" outputs Accept

b) String "αbαα" outputs Accept

c) String "bbα" outputs Accept

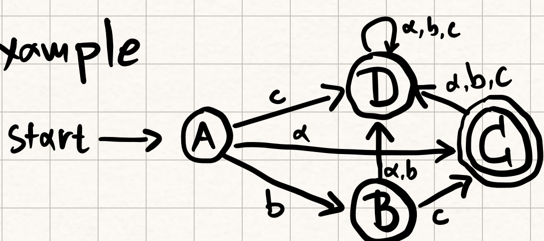
d) Sting "bbαα" outputs Accept

More formally,

A deterministic finite automaton $M = (\Sigma, Q, S, F, \delta)$ is defined by

- Alphabet, Σ
- Finite set of states, Q
- Start state, $S \in Q$
- Set of accept states, $F \subseteq Q$
- Transition function, $\delta : Q \times \Sigma \to Q$

Back to example



$\Sigma = \{\alpha, b, c\}$

$Q = \{A, B, C, D\}$

$S = \{A\}$

$F = \{C\}$

$\delta =$

	α	b	c
A	C	B	D
B	D	D	C
C	D	D	D
D	D	D	D

without trap state

	α	b	c
A	C	B	$\perp$
B	$\perp$	$\perp$	C
C	$\perp$	$\perp$	$\perp$

The term $L(M)$ means that machine M recognizes language L

That is, every $x \in L$ is accepted in M and every accepted x in M is a member of L } Let A be the set of all strings accepted by M, then $A = L$.

Accept $M \iff$ membership in L

Claim: If L_1 is recognized by DFA M_1 and L_2 is recognized by DFA M_2, then there exists a DFA M that recognizes $L_1 \cup L_2$

Proof Sketch:

Let $M_1 = (\Sigma, Q_1, S_1, F_1, \delta_1)$ and $M_2 = (\Sigma, Q_2, S_2, F_2, \delta_2)$

Then, construct a new DFA $M = (\Sigma, Q, S, F, \delta)$ such that

$$Q = \{(A,B) \mid A \in Q_1 \wedge B \in Q_2\}$$

$$S = \{(S_1, S_2)\}$$

It is enough for one of the two states to be an accept state

$$F = \{(A,B) \mid A \in F_1 \vee B \in F_2\}$$

$$\delta((A,B), x) = (\delta_1(A,x), \delta_2(B,x))$$

We have to prove that $L(M) \Rightarrow L_1 \cup L_2$ and $L(M) \Leftarrow L_1 \cup L_2$

Let string $w = w_1 w_2 \ldots w_k$ be the input, where w_i is the i-th character.

sketch

$L(M) \Rightarrow L_1 \cup L_2$: We will show that there exists a sequence of states in M, i.e., (S_1, S_2), (A_1, B_1), ..., (A_k, B_k), such that

① $\delta((S_1, S_2), w_1) = (A_1, B_1)$ → The first pair of states are starting states and transition correctly

② $\delta((A_i, B_i), w_i) = (A_{i+1}, B_{i+1})$ → Every intermediate transition is valid

③ $A_k \in F_1$ OR $B_k \in F_2$ → The last pair of states contains at least one accept state

Without loss of generality, let's assume that we are in case $A_k \in F_1$ (as opposed to $B_k \in F_2$)
By the way that M was constructed it follows that $\delta_1(S_1, w_1) = A_1$
as well as $\forall i \in [1, k-1]: \delta_1(A_i, w_{i+1}) = A_{i+1}$. Similar argument holds for case $B_k \in F_2$.
Since ①, ②, and ③ hold, if w is accepted then it is a member of $L_1 \cup L_2$.

One needs to also show the other direction, $L(M) \Leftarrow L_1 \cup L_2$. ∎

so far we allowed a single transition from state A with input character x.

↓ Extend to allow multiple.

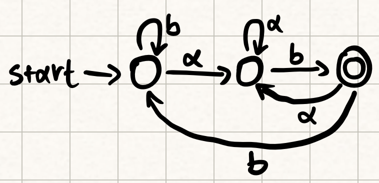
nondeterministic finite automata!

Nondeterministic Finite Automata (NFA) (Sipser, Chapter 1.2, up to "Closure under Regular Operations")

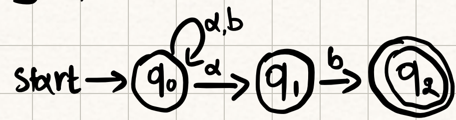
- DFA does not allow any ambiguity on how transitions are made
  ↳ a state and an input character allowed only a single transition

- Representing multiple potential transitions with a fixed state and input character, allows more flexibility in our design.

- Consider $L = \{ w \in \{a,b\}^* \mid w \text{ ends in } ab \}$, we can use either DFA or NFA.



DFA



NFA

A machine accepts an input **if there exists** some sequence of allowable transitions that ends in an accept state

- Two major changes compared to the definition of DFAs:

  ① Instead of $\delta: Q \times \Sigma \to Q$ we have $\delta: Q \times \Sigma \to 2^Q$ → Power Set of Q. All possible subsets.

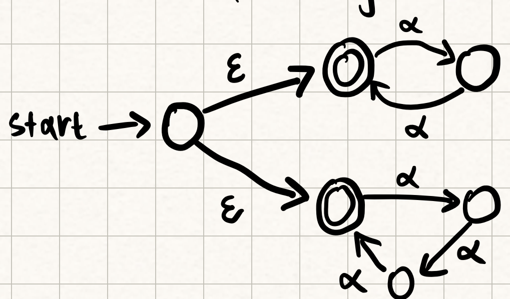
  ② Allow ε transitions. Change state using the empty string (no input characters)

- Thus, the transition function of the above NFA is

$\delta =$

	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\perp$	$\{q_2\}$
q_2	$\perp$	$\perp$

Example: L contains all strings of the form a^k where k is a multiple of 2 or 3.



Question: How much additional power does this nondeterminism gives us?

Equivalence of DFAs and NFAs

DFA $\Rightarrow$ NFA : It is clear that every DFA is also an NFA

NFA $\Rightarrow$ DFA : Need to show that for every $\underbrace{\text{NFA}}_{M}$ there exists a $\underbrace{\text{DFA}}_{M'}$ such that $L(M') = L(M)$

Proof Sketch: The intuition is similar to the proof that there exists a DFA that recognizes the union of two languages.

$\underbrace{\text{NFA}}_{\text{known}}: M = (\Sigma, Q, q_0, F, \delta)$ $\qquad$ $\underbrace{\text{DFA}}_{\text{we need to build}}: M' = (\Sigma, Q', S', F', \delta')$

- To construct Q' we create a new state for every possible subset of Q. For example, if the NFA at hand is the one from the previous page, the set of states Q' is

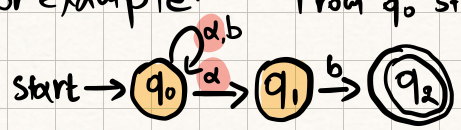
$$Q' = \{\text{"}q_0\text{"}, \text{"}q_1\text{"}, \text{"}q_2\text{"}, \text{"}\{q_0,q_1\}\text{"}, \text{"}\{q_0,q_2\}\text{"}, \text{"}\{q_1,q_2\}\text{"}, \text{"}\{q_0,q_1,q_2\}\text{"}\}$$

⚠ Each state of Q' is a label of a collection of states not the actual collection of states. Recall that states of Q' are part of a DFA, therefore they cannot be an actual collection of states.

$$\text{"}\{q_1,q_2\}\text{"} \neq \{q_1,q_2\}$$

part of a DFA $\qquad$ part of an NFA

For example: From q_0 state, input a takes us to either q_0 or q_1



$\Rightarrow$ Create a DFA state

- If any of the q_1, q_2 NFA states then the DFA state "$\{q_1,q_2\}$" is an accept state.

- Formally, for M' we have:

For example, since q_2 is an accept state, all subsets that include q_2 are accept $\Rightarrow$ "$\{q_2\}$", "$\{q_0,q_2\}$", ...

- Q' = labels of 2^Q
- $S' = \text{"}\{q_0\}\text{"}$
- $F' = \{T \in Q' \mid \exists t \in T \text{ such that } t \in F\}$
- $\delta'(T,x) = \underbrace{\bigcup_{q \in T} \delta(q,x)}$

represented as a set but it is the label of the set and a single state in DFA M'.

- To prove equivalence between Finite Automata we need to show:
  1. If $w \in L(M)$, then $w \in L(M')$
  2. If $w \in L(M')$, then $w \in L(M)$ $\rightarrow$ see notes

- Let $w = w_0 w_1 \dots w_{k-1}$ be the input string and suppose that NFA M accepts w. (known)
We will show that the constructed DFA M' accepts w as well.

- Because M is an NFA we know that there exists some sequence of states $q_0, q_1, \dots, q_k$ such that $q_{i+1} \in \delta(q_i, w_i)$ ① and $q_k \in F$ for input string w.
↳ of NFA ↳ of NFA

- Let's switch now to DFA.
- Let $T_0, T_1, \dots, T_k$ be a sequence of states of M' s.t. $T_{i+1} = \delta'(T_i, w_i)$

* Recall that because M' is a DFA, there is only one sequence of states/transitions for input w

- We need to prove that these transitions lead to an accept state in the DFA.

[Claim: The subset-labeled DFA states $T_0, \dots, T_k$, contain the accept-leading sequence of NFA states $q_0, \dots, q_k$. More formally, $\forall i \in [0,k]$: $q_i \in T_i$ → state of DFA
↳ state of NFA

Proof: Induction ⟹ Base Case: $T_0 = S' = \{q_0\}$
Inductive Hypothesis: Suppose the claim holds for $q_i \in T_i$
Inductive Step: Show that $q_{i+1} \in T_{i+1}$

From definition of T_{i+1}:

→ NFA; From Inductive Hypothesis, $q_i \in T_i$

$$T_{i+1} = \underbrace{\delta'}_{\text{DFA}}(T_i, w_i) = \bigcup_{q \in T_i} \delta(q, w_i) = \delta(q_i, w_i) \cup \left\{ \bigcup_{q \in T_i} \delta(q, w_i) \right\}$$

$$= q_{i+1} \cup \left\{ \bigcup_{q \in T_i} \delta(q, w_i) \right\}$$

↳ from definition of q_{i+1}, i.e., ①

Thus, $q_{i+1} \in T_{i+1}$. From the induction we conclude that $q_k \in T_k$ and since $q_k \in F$ it follows that $T_k \in F'$.
↳ of NFA ↳ of DFA

∎