Automata, languages, and groups of automorphisms of rooted trees

Part I - Introduction to automata and languages

Marialaura Noce

Georg-August-Universität Göttingen

Outline of the course

Contents

The course consists of the following (not equally divided) three parts:

- Part I - Introduction to Automata and Languages
- Part II - Groups and Automata: a perfect match
- Part III - Groups of automorphisms of rooted trees

Plan for today

1. Outline of the course
2. Why?
3. Basic notions in automata theory
4. Automata
5. Finite state automata
6. Regular Languages
7. The pumping lemma
8. Plan for tomorrow

Why?

Why Automata Theory and Formal Languages?

Why Automata Theory and Formal Languages?

- A survey of Stanford 5 years grad students asked which of their courses did they use in their job.
  (I know, this is a *bad* question for a pure mathematician …)

Why Automata Theory and Formal Languages?

- A survey of Stanford 5 years grad students asked which of their courses did they use in their job.
  (I know, this is a *bad* question for a pure mathematician ...)
- Auotmata Theory ranked high.

Why Automata Theory and Formal Languages?

- A survey of Stanford 5 years grad students asked which of their courses did they use in their job.
  (I know, this is a *bad* question for a pure mathematician ...)
- Auotmata Theory ranked high.
- Automata Theory is relevant in many areas of Mathematics and also in Computer science.

Some history...

Some history...

- In the 1930's Gödel, Turing and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer".

Some history...

- In the 1930's Gödel, Turing and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer".
- An example of a problem is "Is an arbitrary mathematical statement true or false?".

Some history...

- In the 1930’s Gödel, Turing and Church discovered that some of the fundamental mathematical problems cannot be solved by a “computer”.
- An example of a problem is “Is an arbitrary mathematical statement true or false?”.
- To understand better a *problem*, we need formal definitions of

Some history...

- In the 1930's Gödel, Turing and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer".
- An example of a problem is "Is an arbitrary mathematical statement true or false?".
- To understand better a *problem*, we need formal definitions of
    - Computer
    - Algorithm
    - Computation

Some history...

- In the 1930’s Gödel, Turing and Church discovered that some of the fundamental mathematical problems cannot be solved by a “computer”.
- An example of a problem is “Is an arbitrary mathematical statement true or false?”.
- To understand better a *problem*, we need formal definitions of
    - Computer
    - Algorithm
    - Computation
- The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Then …why automata?

Automata Theory deals with definitions and properties of different types of “computation models”. Examples of such models are:

Then ...why automata?

Automata Theory deals with definitions and properties of different types of “computation models”. Examples of such models are:

- Finite Automata: used in text processing, compilers, and hardware design.

Automata Theory deals with definitions and properties of different types of "computation models". Examples of such models are:

- Finite Automata: used in text processing, compilers, and hardware design.
- Context-Free Grammars: used to define programming languages and in Artificial Intelligence.

Then ...why automata?

Automata Theory deals with definitions and properties of different types of "computation models". Examples of such models are:

- Finite Automata: used in text processing, compilers, and hardware design.
- Context-Free Grammars: used to define programming languages and in Artificial Intelligence.
- Turing Machines: form a simple abstract model of a "real" computer, such as your PC at home.

And now ...why groups of automorphisms of rooted trees? What is the relation?

No spoiler, we will see later. Let's say that in these lectures we will see some connection between Automata Theory and Group Theory.

Basic notions in automata theory

- Alphabets
- Strings
- Languages

Alphabets

- An *alphabet* A is a finite nonempty set.

Alphabets

- An *alphabet* A is a finite nonempty set.
- The elements of A are *letters*.

Alphabets

- An *alphabet* A is a finite nonempty set.
- The elements of A are *letters*.
- A finte sequence $a_1a_2\ldots a_k$ of elements from A is a *string* or a *word* of length k.

- An *alphabet* A is a finite nonempty set.
- The elements of A are *letters*.
- A finte sequence $a_1a_2\dots a_k$ of elements from A is a *string* or a *word* of length k.
- The string of length 0 is the empty word and it is denoted by ϵ.

Alphabets

- An *alphabet* A is a finite nonempty set.
- The elements of A are *letters*.
- A finte sequence $a_1 a_2 \dots a_k$ of elements from A is a *string* or a *word* of length k.
- The string of length 0 is the empty word and it is denoted by ϵ.

Example

- $A = \{0, 1\}$ is the binary alphabet; 0100 is a string of A of length 4.
- $A = \{a, b, \dots, z\}$ is the set of all lower-case letters.

Powers of an alphabet

- A^k denotes all the strings of length k over A.

Powers of an alphabet

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Powers of an alphabet

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$
- $A^1 = \{0, 1\}$.

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$
- $A^1 = \{0, 1\}$.
- Question: are A and A^1 the same?

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$
- $A^1 = \{0, 1\}$.
- Question: are A and A^1 the same?
- $A^3 = \{000, 010, 100, 001, 011, 101, 110, 111\}$.

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$
- $A^1 = \{0, 1\}$.
- Question: are A and A^1 the same?
- $A^3 = \{000, 010, 100, 001, 011, 101, 110, 111\}$.
- $A^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$.

- A^k denotes all the strings of length k over A.
- A^* denotes all the strings over A.

Example

Let $A = \{0, 1\}$.

- $A^0 = \{\epsilon\}$
- $A^1 = \{0, 1\}$.
- Question: are A and A^1 the same?
- $A^3 = \{000, 010, 100, 001, 011, 101, 110, 111\}$.
- $A^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$.

Note that $A^* = A^0 \cup A^1 \cup A^2 \cup \ldots$.

Concatenation of strings

Let $u = a_1 \dots a_i$ and $v = b_1 \dots b_k$ be strings.

Concatenation of strings

Let $u = a_1 \ldots a_i$ and $v = b_1 \ldots b_k$ be strings.

- The string uv is the concatenation of u and v, i.e. $uv = a_1 \ldots a_i b_1 \ldots b_k$. The string uv is of length $i + k$.

Let $u = a_1 \dots a_i$ and $v = b_1 \dots b_k$ be strings.

- The string uv is the concatenation of u and v, i.e. $uv = a_1 \dots a_i b_1 \dots b_k$. The string uv is of length $i + k$.

Remark

For any string w, we have $w\epsilon = \epsilon w = w$.

Let $u = a_1 \dots a_i$ and $v = b_1 \dots b_k$ be strings.

- The string uv is the concatenation of u and v, i.e. $uv = a_1 \dots a_i b_1 \dots b_k$. The string uv is of length $i + k$.

Remark

For any string w, we have $w\epsilon = \epsilon w = w$.

Example

Let $u = 000$, and $v = 111$. Then $uv = 000111$.

Let A be an alphabet.

Languages

Let A be an alphabet.

- A *language* is a set of strings chosen from A^*, i.e. $L \subseteq A^*$.

Let A be an alphabet.

- A *language* is a set of strings chosen from A^*, i.e. $L \subseteq A^*$.

Example

Let $A = \{a, b, c, d, e, f, g, h, i, l, m, n, o, p, q, r, s, t, u, v, z\}$. The collection of *legal* Italian words is a set of strings of A, thus *Italian* is a language over A.

Let A be an alphabet.

- A *language* is a set of strings chosen from A^*, i.e. $L \subseteq A^*$.

Example

Let $A = \{a, b, c, d, e, f, g, h, i, l, m, n, o, p, q, r, s, t, u, v, z\}$. The collection of *legal* Italian words is a set of strings of A, thus *Italian* is a language over A.

- The empty language $\emptyset$ is a language over any alphabet.

Let A be an alphabet.

- A *language* is a set of strings chosen from A^*, i.e. $L \subseteq A^*$.

Example

Let $A = \{a, b, c, d, e, f, g, h, i, l, m, n, o, p, q, r, s, t, u, v, z\}$. The collection of *legal* Italian words is a set of strings of A, thus *Italian* is a language over A.

- The empty language $\emptyset$ is a language over any alphabet.
- $\{\epsilon\}$ is a language consisting only on the empty string.

Let A be an alphabet.

- A *language* is a set of strings chosen from A^*, i.e. $L \subseteq A^*$.

Example

Let $A = \{a, b, c, d, e, f, g, h, i, l, m, n, o, p, q, r, s, t, u, v, z\}$. The collection of *legal* Italian words is a set of strings of A, thus *Italian* is a language over A.

- The empty language $\emptyset$ is a language over any alphabet.
- $\{\epsilon\}$ is a language consisting only on the empty string.

Remark

Recall that all alphabets are finite. Languages may have an infinite number of strings, but these strings consist of strings drawn from one finite fixed alphabet.

Decision problems

Decision problems

- A *decision problem* is the question of deciding whether a given string is a member of some particular language (we will understand this better later).

Decision problems

- A *decision problem* is the question of deciding whether a given string is a member of some particular language (we will understand this better later).
- If A is an alphabet and L is a language over A, then the problem L is

Decision problems

- A *decision problem* is the question of deciding whether a given string is a member of some particular language (we will understand this better later).
- If A is an alphabet and L is a language over A, then the problem L is

  Given a string w in A^*, decide whether or not w is in L.

- A *decision problem* is the question of deciding whether a given string is a member of some particular language (we will understand this better later).
- If A is an alphabet and L is a language over A, then the problem L is

Given a string w in A^*, decide whether or not w is in L.

Example

The problem of testing whether an integer is a prime, can be expressed by the language L consisting of all binary strings whose value as a binary number is a prime.

Automata

Automaton

From an Ancient Greek dictionary

[αὐτόματον]: *acting of one's own will, of oneself.*

From an Ancient Greek dictionary

[αὐτόματον]: *acting of one's own will, of oneself.*

Automaton

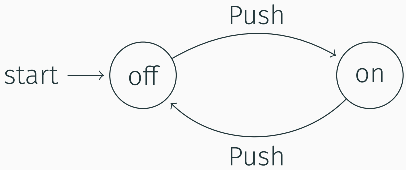
From an Ancient Greek dictionary

[αὐτόματον]: *acting of one's own will, of oneself.*

From Wikipedia

An automaton is a relatively self-operating machine, or a machine or control mechanism designed to automatically follow a predetermined sequence of operations, or to respond to predetermined instructions.

An easy example



(Still) Informal definition of automata

- An automaton $\mathcal{A}$ over an alphabet A is a device (machine) that reads input strings over A and accepts some of them (that is, given a string w, $\mathcal{A}$ accepts w by halting in an accepting state).

(Still) Informal definition of automata

- An automaton $\mathcal{A}$ over an alphabet A is a device (machine) that reads input strings over A and accepts some of them (that is, given a string w, $\mathcal{A}$ accepts w by halting in an accepting state).
- The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of all strings that $\mathcal{A}$ accepts.

Types of automata

Types of automata II

- finite state automaton ✓
  - deterministic
  - nondeterministic
- pushdown automaton ✓
- linear-bounded automaton
- Turing machine

Finite state automata

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.
- A transition function $\delta : Q \times A \to Q$, that takes as arguments a state and an input symbol and returns a state.

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.
- A transition function $\delta : Q \times A \to Q$, that takes as arguments a state and an input symbol and returns a state.
- You can think that δ is the "program" $\mathcal{A}$ that tells us what $\mathcal{A}$ can do in one step.

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.
- A transition function $\delta : Q \times A \to Q$, that takes as arguments a state and an input symbol and returns a state.
- You can think that δ is the "program" $\mathcal{A}$ that tells us what $\mathcal{A}$ can do in one step.
- A start state $q_0 \in Q$.

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.
- A transition function $\delta : Q \times A \to Q$, that takes as arguments a state and an input symbol and returns a state.
- You can think that δ is the “program” $\mathcal{A}$ that tells us what $\mathcal{A}$ can do in one step.
- A start state $q_0 \in Q$.
- A set of accepting states F from Q.

We will use the following notation:

Deterministic Finite State Automata (DFA)

A deterministic finite automaton $\mathcal{A}$ consists of:

- A finite set of states Q.
- A finite set of input symbols A. We also call A alphabet.
- A transition function $\delta : Q \times A \to Q$, that takes as arguments a state and an input symbol and returns a state.
- You can think that δ is the "program" $\mathcal{A}$ that tells us what $\mathcal{A}$ can do in one step.
- A start state $q_0 \in Q$.
- A set of accepting states F from Q.

We will use the following notation:

$$\mathcal{A} = (Q, A, \delta, q_0, F).$$

- Transition diagrams
- Transition tables

Do you remember the example of the light on and off?

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

- Vertices: states. Also:

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

- Vertices: states. Also:
  - Initial state: empty single incoming arc.

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

- Vertices: states. Also:
  - Initial state: empty single incoming arc.
  - Final state: double circle.

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

- Vertices: states. Also:
    - Initial state: empty single incoming arc.
    - Final state: double circle.

Do you remember the example of the light on and off?

A transition diagram is a graph defined as follows:

- Vertices: states. Also:
  - Initial state: empty single incoming arc.
  - Final state: double circle.
- Arc: labeled with input symbol from A (or letter from the alphabet A), they show the transition.

Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a DFA.

- The language accepted by $\mathcal{A}$, that we denote with $L(\mathcal{A})$, is the set of labels of the paths in the transition diagram of $\mathcal{A}$ that start at the initial state q_0 and end at a final state in F.

Transition tables

A transition table is a conventional tabular representation of the map δ, described as follows:

Transition tables

A transition table is a conventional tabular representation of the map δ, described as follows:

Transition tables

A transition table is a conventional tabular representation of the map δ, described as follows:

- Rows: states

A transition table is a conventional tabular representation of the map δ, described as follows:

- Rows: states
- Columns: letters from A.

A transition table is a conventional tabular representation of the map δ, described as follows:

- Rows: states
- Columns: letters from A.
- The entry for the row corresponding to the state q and the column corresponding to the input a is the state $\delta(q, a)$.

Example

Let $L = \{w \mid w$ is a binary string containing an odd number of 1 $\}$.
How can we build a DFA that accepts L?

Example

Let $L = \{w \mid w$ is a binary string containing an odd number of 1 $\}$.
How can we build a DFA that accepts L?

"Human" attempt:

Example

Let $L = \{w \mid w \text{ is a binary string containing an odd number of 1 }\}$.
How can we build a DFA that accepts L?

"Human" attempt:

- The finite automaton reads the input string w from left to right and keeps track of the number of 1.

Example

Let $L = \{w \mid w \text{ is a binary string containing an odd number of 1 }\}$.
How can we build a DFA that accepts L?

"Human" attempt:

- The finite automaton reads the input string w from left to right and keeps track of the number of 1.
- After having read the entire string w, it checks whether this number is odd accepted or even rejected.

Example

Let $L = \{w \mid w$ is a binary string containing an odd number of 1 $\}$.
How can we build a DFA that accepts L?

"Human" attempt:

- The finite automaton reads the input string w from left to right and keeps track of the number of 1.
- After having read the entire string w, it checks whether this number is odd accepted or even rejected.
- Thus the automaton $\mathcal{A}$ needs a state for each $n \geq 0$.

Example

Let $L = \{w \mid w$ is a binary string containing an odd number of 1 $\}$.
How can we build a DFA that accepts L?

"Human" attempt:

- The finite automaton reads the input string w from left to right and keeps track of the number of 1.
- After having read the entire string w, it checks whether this number is odd accepted or even rejected.
- Thus the automaton $\mathcal{A}$ needs a state for each $n \geq 0$.
- Is this a good way to do it?

Idea: keep track of the number of 1 read is even or odd.

Idea: keep track of the number of 1 read is even or odd.

- States: $Q = \{q_E, q_O\}$.

Idea: keep track of the number of 1 read is even or odd.

- States: $Q = \{q_E, q_O\}$.
- Alphabet: $A = \{0, 1\}$.

Example II: good approach

Idea: keep track of the number of 1 read is even or odd.

- States: $Q = \{q_E, q_O\}$.
- Alphabet: $A = \{0, 1\}$.
- Start state: q_E.

Example II: good approach

Idea: keep track of the number of 1 read is even or odd.

- States: $Q = \{q_E, q_O\}$.
- Alphabet: $A = \{0, 1\}$.
- Start state: q_E.
- The set F of accept states: $F = \{q_O\}$.

Idea: keep track of the number of 1 read is even or odd.

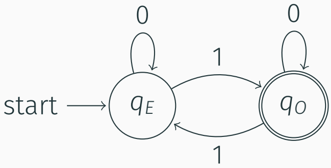
- States: $Q = \{q_E, q_O\}$.
- Alphabet: $A = \{0, 1\}$.
- Start state: q_E.
- The set F of accept states: $F = \{q_O\}$.
- The transition table for δ is:

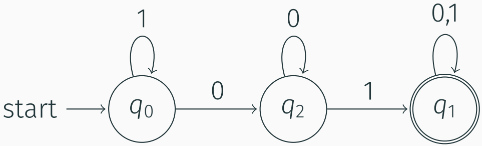
	0	1
q_E	q_E	q_O
q_O	q_O	q_E

Example

We have

- $\delta(q_E, 0) = q_E$
- $\delta(q_E, 1) = q_O$
- $\delta(q_O, 0) = q_O$
- $\delta(q_O, 1) = q_E$





Can you guess what is this?

Extending the Transition Function to Strings

- Now we define an *extended transition function* $\bar{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols from A.

Extending the Transition Function to Strings

- Now we define an *extended transition function* $\bar{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols from A.
- This can be done by induction on the length of the string w:

- Now we define an *extended transition function* $\bar{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols from A.
- This can be done by induction on the length of the string w:
    - If $|w| = 0$, then
$$\bar{\delta}(q, \epsilon) = q.$$

Extending the Transition Function to Strings

- Now we define an *extended transition function* $\bar{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols from A.
- This can be done by induction on the length of the string w:
  - If $|w| = 0$, then
$$\bar{\delta}(q, \epsilon) = q.$$
  - Let $w = xa$, where x is a string and $|a| = 1$. Then

- Now we define an *extended transition function* $\bar{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols from A.
- This can be done by induction on the length of the string w:
    - If $|w| = 0$, then

$$\bar{\delta}(q, \epsilon) = q.$$

    - Let $w = xa$, where x is a string and $|a| = 1$. Then

$$\bar{\delta}(q, w) = \delta(\bar{\delta}(q, x), a).$$

Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a DFA.

- A string w is accepted by $\mathcal{A}$ if $\mathcal{A}$ starting at the initial state ends in an accepting state after reading the string.

Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a DFA.

- A string w is accepted by $\mathcal{A}$ if $\mathcal{A}$ starting at the initial state ends in an accepting state after reading the string.
- In other words, a string w is accepted if $\bar{\delta}(q_0, w) \in F$.
- The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined to be the set of all strings that are accepted by $\mathcal{A}$:

$$L(\mathcal{A}) = \{w \mid w \text{ is a string over } A \text{ and } \mathcal{A} \text{ accepts } w\}.$$

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.

Extending the Transition Function to Strings: Example

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$

Consider the language $L = \{w \mid w$ has an even number of 0 and 1$\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:
  - q_0: the number of 0 and 1 is even

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:
  - q_0: the number of 0 and 1 is even
  - q_1: the number of 0 is even and of 1 is odd

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:
  - q_0: the number of 0 and 1 is even
  - q_1: the number of 0 is even and of 1 is odd
  - q_2: the number of 0 is odd and of 1 is even

Extending the Transition Function to Strings: Example

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:
  - q_0: the number of 0 and 1 is even
  - q_1: the number of 0 is even and of 1 is odd
  - q_2: the number of 0 is odd and of 1 is even
  - q_3: the number of 0 and 1 is odd

Extending the Transition Function to Strings: Example

Consider the language $L = \{w \mid w \text{ has an even number of 0 and 1}\}$.
What is a DFA $\mathcal{A}$ accepting L?

- Idea: the states of $\mathcal{A}$ must count the number of 0 and the number of 1 modulo 2.
- Consider the alphabet $A = \{0, 1\}$
- Consider 4 states:
  - q_0: the number of 0 and 1 is even
  - q_1: the number of 0 is even and of 1 is odd
  - q_2: the number of 0 is odd and of 1 is even
  - q_3: the number of 0 and 1 is odd
- The state q_0 is the initial and the final state.

Summarizing:

$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\}).$$

Summarizing:

$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\}).$$

Transition table for δ:

	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

...continuing II

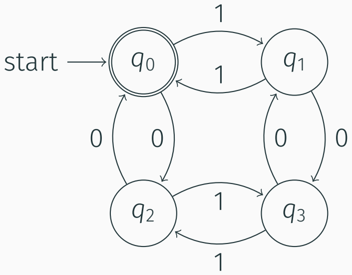
$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

Transition diagram for $\mathcal{A}$:

$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

Transition diagram for $\mathcal{A}$:



- Take $w = 1010$.

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

- $\bar{\delta}(q_0, \epsilon) = q_0$

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

- $\bar{\delta}(q_0, \epsilon) = q_0$
- $\bar{\delta}(q_0, 1) = \delta(\bar{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

- $\bar{\delta}(q_0, \epsilon) = q_0$
- $\bar{\delta}(q_0, 1) = \delta(\bar{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$
- $\bar{\delta}(q_0, 10) = \delta(\bar{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_3$

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

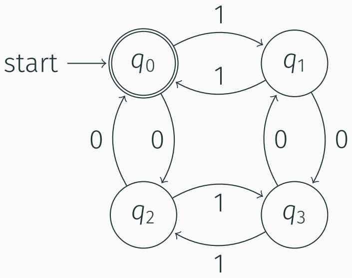
- $\bar{\delta}(q_0, \epsilon) = q_0$
- $\bar{\delta}(q_0, 1) = \delta(\bar{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$
- $\bar{\delta}(q_0, 10) = \delta(\bar{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_3$
- $\bar{\delta}(q_0, 101) = \delta(\bar{\delta}(q_0, 10), 1) = \delta(q_3, 1) = q_2$.

- Take $w = 1010$.
- In this string the number of 0 is even and the number of 1 is even.
- Then we expect that $\bar{\delta}(q_0, w) = q_0$.

We start from ϵ and then we increase size

- $\bar{\delta}(q_0, \epsilon) = q_0$
- $\bar{\delta}(q_0, 1) = \delta(\bar{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$
- $\bar{\delta}(q_0, 10) = \delta(\bar{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_3$
- $\bar{\delta}(q_0, 101) = \delta(\bar{\delta}(q_0, 10), 1) = \delta(q_3, 1) = q_2$.
- $\bar{\delta}(q_0, 1010) = \delta(\bar{\delta}(q_0, 101), 0) = \delta(q_2, 0) = q_0$.

In other words, a transition function is a path in the transition diagram. Take again the string $w = 1010$.




Take a break

Let's start slowly and play a game

Let's play a game on a 3x3 chessboard.

1	2	3
4	5	6
7	8	9

- Goal: start at 1 and go to 9.
- Rules: move to an adjacent square.

- States: squares of the chessboard, that is $Q = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $A = \{b, p\}$, where
  - b = move to any adjacent blue square
  - p = move to any adjacent pink square
- Initial state: $q_0 = 1$
- Final state: $q_F = 9$

If there are choices where to go, we try all.

A non-deterministic finite automaton consists of:

A non-deterministic finite automaton consists of:

- A finite set of states Q.

A non-deterministic finite automaton consists of:

- A finite set of states Q.
- A finite set of input symbols A.

Non-Deterministic Finite State Automata (NDFA)

A non-deterministic finite automaton consists of:

- A finite set of states Q.
- A finite set of input symbols A.
- A transition function $\delta : Q \times A \cup \{\epsilon\} \to P(Q)$, that takes as arguments a state and an input symbol and returns a a set of states.

Non-Deterministic Finite State Automata (NDFA)

A non-deterministic finite automaton consists of:

- A finite set of states Q.
- A finite set of input symbols A.
- A transition function $\delta : Q \times A \cup \{\epsilon\} \rightarrow P(Q)$, that takes as arguments a state and an input symbol and returns a a set of states.
- A start state $q_0 \in Q$.

Non-Deterministic Finite State Automata (NDFA)

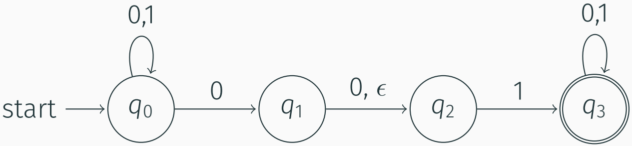
A non-deterministic finite automaton consists of:

- A finite set of states Q.
- A finite set of input symbols A.
- A transition function $\delta : Q \times A \cup \{\epsilon\} \rightarrow P(Q)$, that takes as arguments a state and an input symbol and returns a a set of states.
- A start state $q_0 \in Q$.
- A set of accepting states F from Q.

We will use the following notation:

$$\mathcal{A} = (Q, A, \delta, q_0, F).$$

Example



What happens if we take the string $w = 000110$?

Example II

Non-Deterministic Finite State Automata (NDFA)

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.

Non-Deterministic Finite State Automata (NDFA)

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.
- Formally:

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.
- Formally:
  - An NDFA $\mathcal{A}$ accepts a string w, if w can be written as $w = y_1 y_2 \ldots y_m$, where $y_i \in A \cup \{\epsilon\}$ for all $1 \leq i \leq m$, and there exists a sequence of states $r_0, r_1, \ldots, r_m$ in Q, such that

Non-Deterministic Finite State Automata (NDFA)

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.
- Formally:
    - An NDFA $\mathcal{A}$ accepts a string w, if w can be written as $w = y_1 y_2 \ldots y_m$, where $y_i \in A \cup \{\epsilon\}$ for all $1 \leq i \leq m$, and there exists a sequence of states $r_0, r_1, \ldots, r_m$ in Q, such that
        - $r_0 = q$
        - $r_{i+1} \in \delta(r_i, y_{i+1})$
        - $r_m \in F$

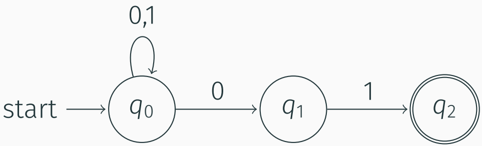
Non-Deterministic Finite State Automata (NDFA)

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.
- Formally:
    - An NDFA $\mathcal{A}$ accepts a string w, if w can be written as $w = y_1 y_2 \ldots y_m$, where $y_i \in A \cup \{\epsilon\}$ for all $1 \leq i \leq m$, and there exists a sequence of states $r_0, r_1, \ldots, r_m$ in Q, such that
        - $r_0 = q$
        - $r_{i+1} \in \delta(r_i, y_{i+1})$
        - $r_m \in F$
    - In other words a strings w is accepted if $\bar{\delta}(q_0, w) \cap F \neq \emptyset$, that is that contains at least one accepting state.

Non-Deterministic Finite State Automata (NDFA)

- Informally (remember the picture before): an NDFA accepts a string if there exists at least one path in the state diagram that starts at the initial state, and ends at an accept state.
- Formally:
  - An NDFA $\mathcal{A}$ accepts a string w, if w can be written as $w = y_1 y_2 \dots y_m$, where $y_i \in A \cup \{\epsilon\}$ for all $1 \leq i \leq m$, and there exists a sequence of states $r_0, r_1, \dots, r_m$ in Q, such that
    - $r_0 = q$
    - $r_{i+1} \in \delta(r_i, y_{i+1})$
    - $r_m \in F$
  - In other words a strings w is accepted if $\bar{\delta}(q_0, w) \cap F \neq \emptyset$, that is that contains at least one accepting state.

Similarly as before, the language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined to be the set of all strings that are accepted by $\mathcal{A}$.



- Can you guess what are the strings recognized by this automaton?
- Why is this NDFA and not DFA?

DFA vs NDFA

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol.	The transition from a state can be to multiple next states for each input symbol.
Empty string transitions are not allowed.	NDFA permits empty string transitions.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions.

Exercise for you

Formalise what we have said with the example of the 3x3 chessboard.

Question: It seems that NDFA are more powerful than DFA. Is this true?

Question: It seems that NDFA are more powerful than DFA. Is this true?

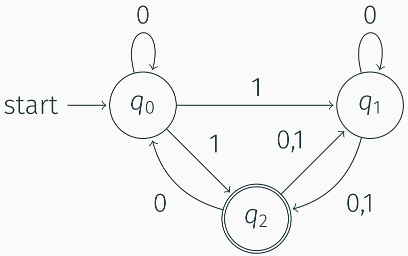
A language can be accepted by a DFA if and only if it can be accepted by an NDFA.

Question: It seems that NDFA are more powerful than DFA. Is this true?

A language can be accepted by a DFA if and only if it can be accepted by an NDFA.

- Of course it is easy to convert a DFA to a NDFA.
- What about the converse?
- We will forget about the ϵ-transition.

From NDFA to DFA



From NDFA to DFA II

Transition table:

	0	1
q_0	q_1	q_1, q_2
q_1	q_1, q_2	q_2
q_2	q_0, q_1	q_1

Some observations:

Some observations:

- After conversion, the number of states in the resulting DFA may or may not be same as NDFA

Some observations:

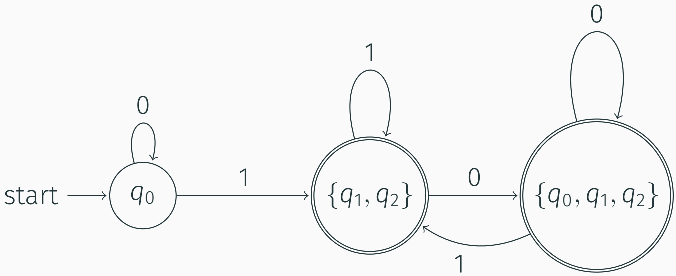
- After conversion, the number of states in the resulting DFA may or may not be same as NDFA
- The maximum number of states is at most $2^{|Q|}$

Some observations:

- After conversion, the number of states in the resulting DFA may or may not be same as NDFA
- The maximum number of states is at most $2^{|Q|}$
- In the resulting DFA, all those states that contain the final state(s) of NDFA are treated as final states

Let's now build this DFA

And the final result should be this ...



Finite state automata may have outputs corresponding to each transition. There are two types of finite state machines that generate output:

- Mealy Machine
- Moore machine

We will see this later (where *later* means tomorrow) ...

Regular Languages

Definition

A language K is called regular if there exists a finite state automaton $\mathcal{A}$ such that

$$K = L(\mathcal{A}).$$

There are three operations on languages. Let L_1 and L_2 be two languages over the same alphabet.

Regular operations

There are three operations on languages. Let L_1 and L_2 be two languages over the same alphabet.

- The *union* of L_1 and L_2 is

$$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}.$$

There are three operations on languages. Let L_1 and L_2 be two languages over the same alphabet.

- The *union* of L_1 and L_2 is

$$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The *concatenation* of L_1 and L_2 is

$$L_1 L_2 = \{ww' \mid w \in L_1 \text{ and } w' \in L_2\}.$$

There are three operations on languages. Let L_1 and L_2 be two languages over the same alphabet.

- The *union* of L_1 and L_2 is

$$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The *concatenation* of L_1 and L_2 is

$$L_1 L_2 = \{ww' \mid w \in L_1 \text{ and } w' \in L_2\}.$$

- The *star* of L_1 is

$$L_1^* = \{u_1 u_2 \ldots u_k \mid k \geq 0 \text{ and } u_i \in L_1 \text{ for all } i = 1, 2, \ldots, k\}.$$

Let $L_1 = \{\text{empty}, \text{full}\}$ and $L_2 = \{\text{cup}, \text{bottle}\}$.

- What is $L_1 \cup L_2$?
- What is $L_1 L_2$?
- What is L_1^*?

Is the set of all regular languages closed under these operations?

Theorem

The set of regular languages is closed under the union operation, i.e., if L_1 and L_2 are regular languages over the same alphabet A, then $L_1 \cup L_2$ is also a regular language.

The proof works as follows:

Proof.

□

Is the set of all regular languages closed under the other operations seen before (concatenation and star)?

Theorem

Yes.

Proof.

Exercise ;) □

The pumping lemma

Therapist: The Pumping Lemma is not real, it can not hurt you

The Pumping Lemma:

Let L be a regular language. $\Rightarrow \exists$ constant $n \in \mathbb{N}$:

$$\underset{\substack{w \in L \\ |w| \geq n}}{\forall} \exists\, x, y, z \in \Sigma^* \text{ such that } w = xyz, y \neq \epsilon, |xy| \leq n, \underset{k \geq 0}{\forall} xy^k z \in L$$

- We saw that the class of regular languages is closed under some operations.

- We saw that the class of regular languages is closed under some operations.
- Regular languages can be described by finite state automata.

- We saw that the class of regular languages is closed under some operations.
- Regular languages can be described by finite state automata.
- All these tools help to prove that a language is regular.

- We saw that the class of regular languages is closed under some operations.
- Regular languages can be described by finite state automata.
- All these tools help to prove that a language is regular.
- What if one wants to prove that a language is *not* regular?

- We saw that the class of regular languages is closed under some operations.
- Regular languages can be described by finite state automata.
- All these tools help to prove that a language is regular.
- What if one wants to prove that a language is *not* regular?
- Let $L = \{0^m 1^m \mid m \geq 0\}$. Can you establish if this language is regular?

The *pumping lemma* is a property that all regular languages must possess.

The pumping lemma II

Informal statement:

Theorem

If a language is regular, all sufficiently long string in the language can be pumped.

Formal statement:

Theorem

Let L be a regular language. Then there exists an integer $p \geq 1$ (called the pumping length) such that the following holds: Every string s in L, with $|s| \geq p$, can be written as $s = xyz$, such that

- $|y| \geq 1$
- $|xy| \leq p$
- *for all* $i \geq 0$, $xy^iz \in L$.

This means that by replacing the portion y in s by zero or more copies of it, the resulting string is still in the language L.

Any regular language L has a magic number p
and any long-enough word in L has the following property:
among its first p symbols is a segment you can find
whose repetition or omission leaves x among its kind.
So if you find a language L which fails this acid test,
and some long word you pump becomes distinct from all the rest,
by contradiction you have shown that language L is not
a regular guy, resilient to the damage you have wrought.
But if, upon the other hand, x stays within its L,
then either L is regular, or else you chose not well.
For w is xyz, and y cannot be null,
and y must come before p symbols have been read in full.

Consider the language $L = \{0^m 1^m \mid m \geq 0\}$.
Claim: this language is not regular.

Let $L = \{0^n \mid n \text{ is a prime number}\}$.
Is L a regular language?

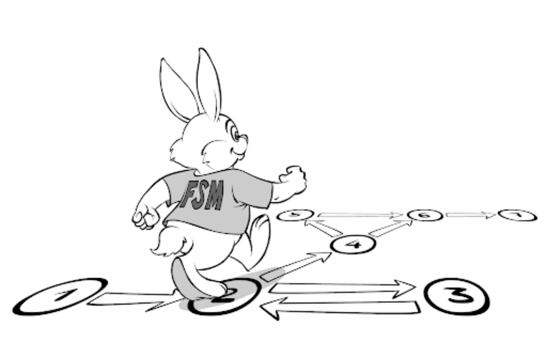
Plan for tomorrow

Plan for tomorrow

- Part I - Introduction to Automata and Languages:
  - Introduction to grammars
  - Context-free grammars
  - Pushdown automata
- Part II - Groups and Automata: what is this match?

Questions or answers?

Obrigada :)



(Picture of me trying to go to Brasilia with a finite state machine)