Welcome to CS-251:

Theory of Computation!

Mika Göös

EPFL School of Computer and Communication Sciences

Lecture 1

What is CS-251?

- Mathematical — problem solving, ability to write proofs. Must attend exercise sessions and solve homework problems...
- Challenging — abstract thinking, how to reduce one problem to another?

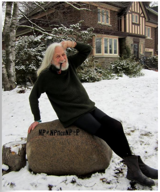
What are the fundamental capabilities and limitations of computers?

This question goes back to the 1930s...

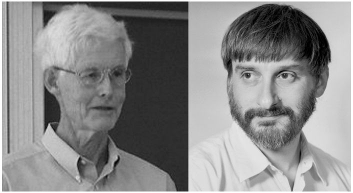
Gödel: What can be mathematically proved?

Turing: What can be computed?

? Alan Turing



? Jack Edmonds



? Stephen Cook & Leonid Levin

? Avi Wigderson

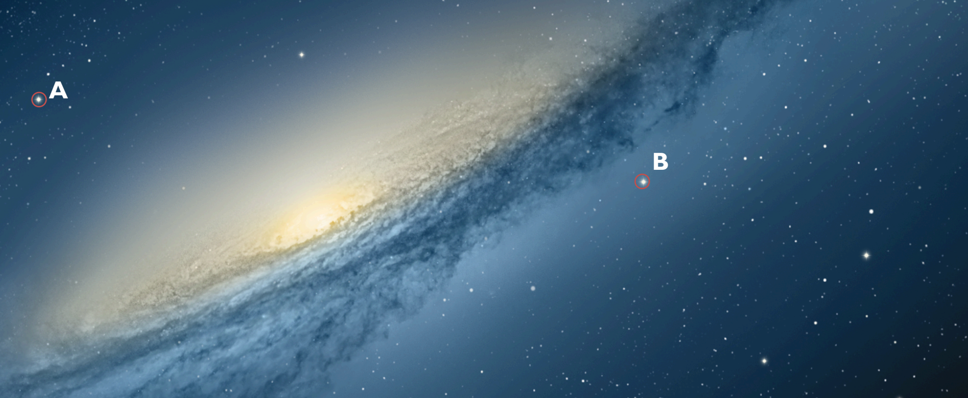
Seen as the father of computer science

"On Computable Numbers, with an Application to the Entscheidungsproblem"

- Introduced "Universal machine" that is capable of computing anything that is computable
- "Central concept of modern computer" was due to this theoretical paper published in 1936

Introduced the class **P** — the idea that practical computation is

The Computational Universe

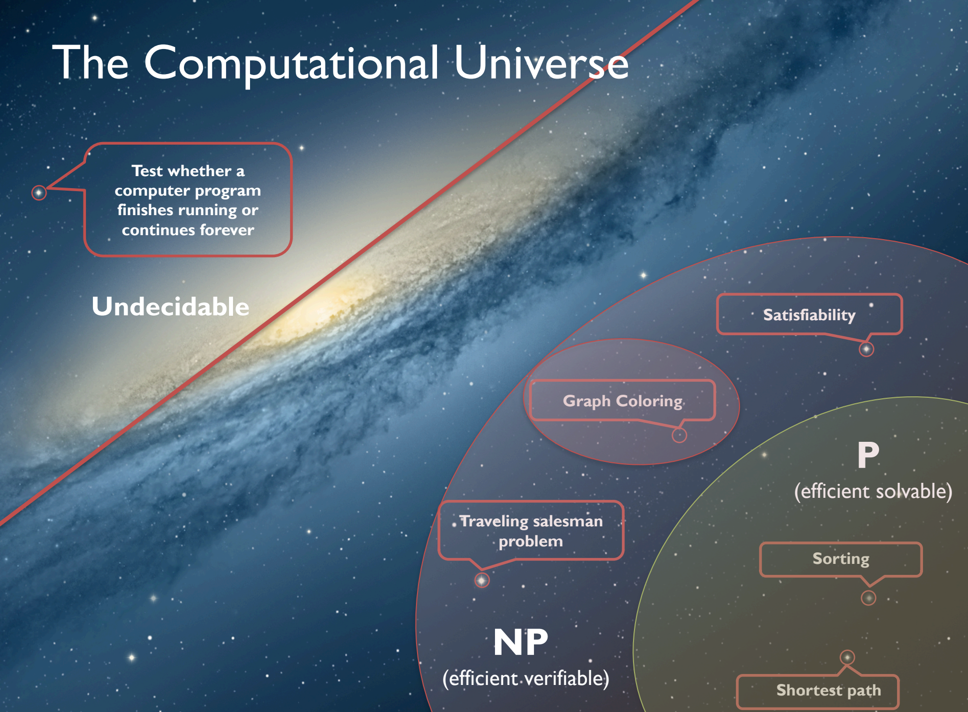


What are the relations between different problems and between different computational models?

Does randomness help? Quantum?

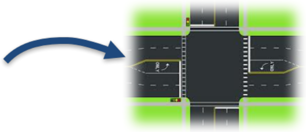
If I can solve problem A, can I solve problem B?

The Computational Universe



Why I (and hopefully you will) love this topic

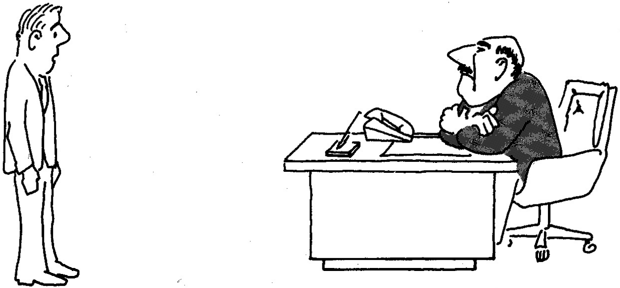
- Abstraction



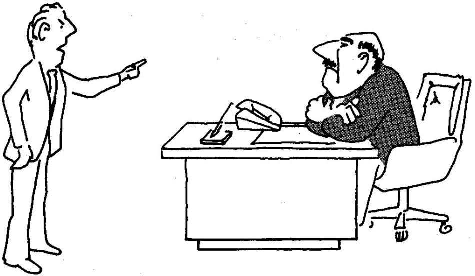
- No limits

  - “model of computing” introduced before computers
  - Quantum computers

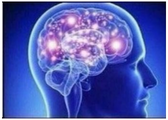
- Profound impact not only on computer science
  - **P** vs **NP** one of the seven most important math problems
  - Biology: nature does computation all the time (e.g. evolution)
  - Game theory: analysis of city planning, economics, etc.

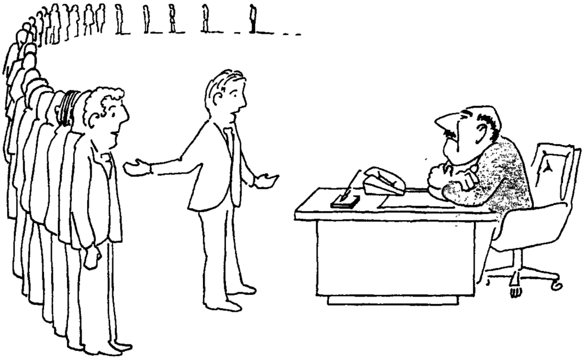


"I can't find an efficient algorithm, I guess I'm just too dumb."



"I can't find an efficient algorithm, because no such algorithm is possible!"





"I can't find an efficient algorithm, but neither can all these famous people."

WHO WILL TEACH YOU ALL THE COOL MATERIAL?

The Dream Team of Teaching Assistants

Happy to help and answer any questions!

PhD	BA/MA
Ziyi (head TA)	Fedor
Valentin (exercises)	Antoine
Anastasia	Adrien
Artur	Madeline
Hristo	Georgios
Ekaterina	Martin
Zijing	Pierre

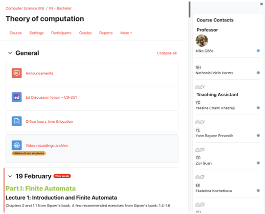
Who am I (the course responsible)?

- Mika Göös
  - *mika.goos@epfl.ch*
  - *https://theory.epfl.ch/mika/*
- Faculty of computer science
  - Research in *Complexity Theory*
- Feedback welcome!

THE COURSE ESSENTIALS

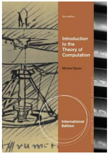
Resources

Moodle: *http://moodle.epfl.ch/*



Dynamic: course material (these slides will be there), exercises+solutions, links, etc.

Textbook



Introduction to the Theory of Computation, 3rd international edition (2013) by Michael Sipser

Nice reading. To learn best solve as many exercises as you can

Time and Location

Lectures:

- Monday 13:15–15:00

Exercise Session:

- Monday 15:15–17:00

Office hours: TBD

Ed Discussion forum: All questions about course admin and content

All links on Moodle!

Grading

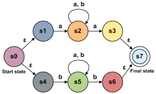
30% — 3 sets of homeworks in groups of 2–4 students

70% — Final exam in June/July

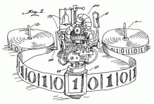
Content

Understanding the limits of computational models

- Part I — What problems can we solve with **constant** memory?
  - Finite Automaton and Regular Languages
  - Non-determinism
  - Non-regular languages



- Part II — What is computable with any computer?
  - Turing Machines
  - Decidability/Undecidability



- Part III — What is computable **efficiently**?
  - Time complexity
  - **P**, **NP**
  - **NP**-completeness

Let's start our journey!

What can we do with limited memory?

Example: Parity

Input: A string s made of symbols M and W

Output: **Yes** if M appears in s an even number of times.
No otherwise

Examples: MWMWMW $\rightarrow$ **No** MWMMMW$\rightarrow$ **Yes**

Computational model: Use just 1 bit of memory

What can algorithm on a 1-bit computer do?

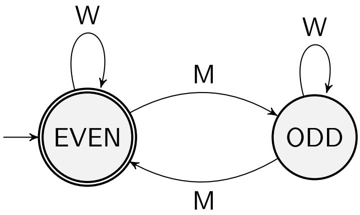
- Initialize memory
- Scan input from left to right
- For every symbol seen, change the memory state based on
  - the current state
  - the current symbol
- Provide an answer based on the final state of memory

$\Longrightarrow$

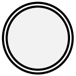
Memory states:
ODD and EVEN

	ODD	EVEN
M	?	?
W	?	?

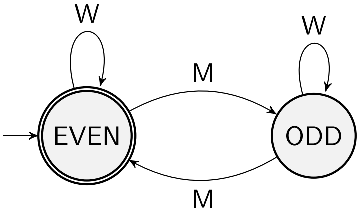
Our first algorithm on a 1-bit computer



State diagram of **Deterministic Finite Automaton (DFA)**

1. Alphabet $\Sigma = \{M, W\}$ (given by problem description)
2. States (memory allowance). In this case 2 but in general any finite number independent of input length
3. Transition function (arrows)
4. Starting state 
5. Accepting state(s) 

How to check if a DFA accepts a string?



Start

	M	W	W	M	W	M	M
E	O	O	O	E	E	O	E

Accepting

	W	M	M	W	W	M
E	E	O	E	E	E	O

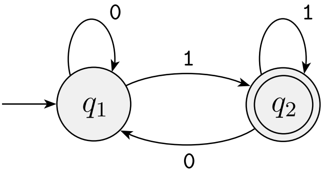
Not Accepting

This DFA accepts even if the input string is empty!

ε

Example 1

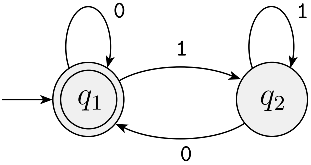
To understand what strings a DFA accepts, try a couple of strings to understand the roles of the states!



What strings does the DFA accept? Strings ending with a 1

Example 2

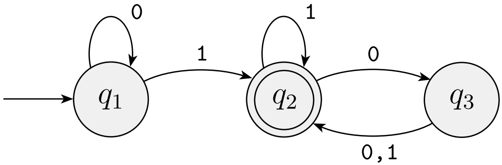
To understand what strings a DFA accepts, try a couple of strings to understand the roles of the states!



What strings does the DFA accept? Strings ending with a 0 plus empty string ε

Example 3

To understand what strings a DFA accepts, try a couple of strings to understand the roles of the states!



What strings does the DFA accept? Strings with at least one 1 that ends with an even number of 0's

Is the concept of DFAs clear? NO!

Although state diagrams are easier to grasp intuitively, we need the formal definition, too, for two specific reasons:

- Clarify uncertainties
  - Are 0 accept states allowed?
  - Must have exactly one transition exiting every state for each possible input symbol?
- A formal definition provides notation
  - Good notation helps you think and express your thoughts clearly

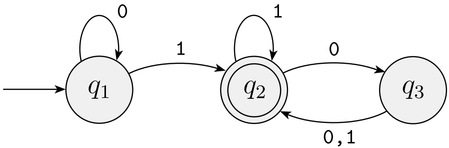
Formal definitions

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

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states. *(allow $F = \emptyset$)*

- $\delta(q, \sigma)$ encodes the state we go to from q when reading $\sigma \in \Sigma$. For a string s we use $\delta(q, s)$ to denote the state obtained by reading all of s starting in state q.
- If A is the set of all strings that machine M accepts, we say that A is the **language of machine** M and write $L(M) = A$. We say that M **recognizes** A or that M **accepts** A. If the machine accepts no strings, it still recognizes one language — namely, the empty language $\emptyset$.

Example



We can describe this DFA M formally by writing $M = (Q, \Sigma, \delta, q_1, F)$, where

- $Q = \{q_1, q_2, q_3\}$,
- $\Sigma = \{0, 1\}$,
- δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

- q_1 is the start state, and
- $F = \{q_2\}$.

M recognizes the language $L(M) =$
$\{w \mid w$ contains at least one 1
and an even number of 0s follow the last 1$\}$

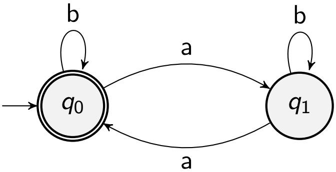
PROVING CORRECTNESS OF AUTOMATA

Induction!

Does your DFA accept the correct language?

$\Sigma = \{a, b\}, \qquad L = \{w \mid \underbrace{w \text{ contains an even number of } a\text{'s}}_{count(w,a) \text{ is even}}\}$

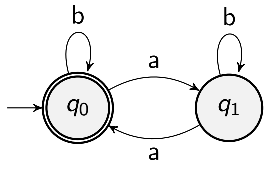
DFA M:



How do you prove that M is correct, that is, M accepts exactly L?

- To prove: For all strings w, M accepts w iff $count(w, a)$ is even
- To prove: For all strings w, $\delta(q_0, w) = q_0$ iff $count(w, a)$ is even

Proof of correctness



To prove: For all strings w, $\delta(q_0, w) = q_0$ iff $count(w, a)$ is even

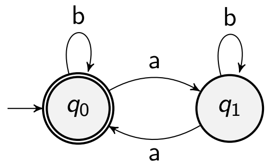
Proof by induction on string length/structure

Base case: Prove the claim for $w = \varepsilon$

Inductive case:

- Assume that claim holds for an arbitrary string x
- Prove the claim for $w = x.\sigma$, where σ is a symbol (either a or b)

Proof of correctness: Base Case

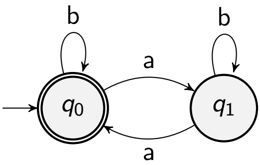


To prove: For all strings w, $\delta(q_0, w) = q_0$ iff $count(w, a)$ is even

Base case: Prove the claim for $w = \varepsilon$

- We have $\delta(q_0, \varepsilon) = q_0$
- The empty string has 0 number of a's, which is an even number
- So the claim holds

Inductive Case

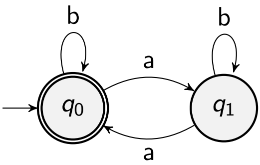


To prove: For all strings w, $\delta(q_0, w) = q_0$ iff $count(w, a)$ is even

Inductive case: Assume that the claim holds for an arbitrary string x:
that is, assume: $\delta(q_0, x) = q_0$ iff $count(x, a)$ is even

- Need to show the claim for $x.\sigma$, where σ is symbol in $\{a, b\}$
- $\delta(q_0, x)$ can be q_0 or q_1, and σ can be a or b.
- Gives four cases to consider. Let us consider the case $\delta(q_0, x) = q_0$ and $\sigma = b$, rest three are similar

Inductive Case



To prove: For all strings w, $\delta(q_0, w) = q_0$ iff $count(w, a)$ is even

Case $\delta(q_0, x) = q_0$ and $\sigma = b$:

- By Induction Hypothesis, $count(x, a)$ is even
- To prove $\delta(q_0, x.b) = q_0$ iff $count(x.b, a)$ is even
- By definition of δ:

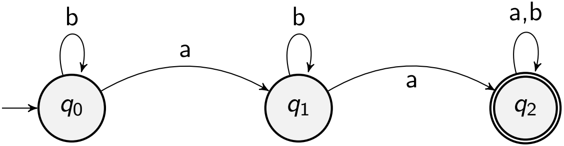
$$\delta(q_0, x.b) = \delta(\delta(q_0, x), b) = \delta(q_0, b) = q_0$$

- Adding b to a string does not change the number of a's it contains, so $count(x.b, a)$ equals $count(x, a)$, which is even in this case. QED

Another Example

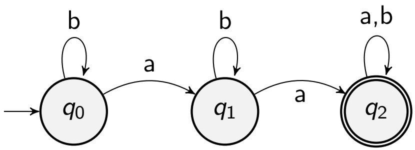
What language does this DFA accept?

DFA M:



Claim: $L(M) = \{w \mid\ w \text{ contains at least two } a\text{'s}\}$.

Proof of correctness



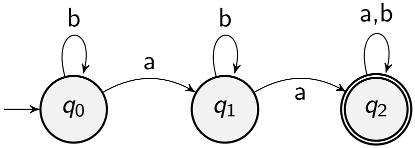
To prove: For all strings w, $\delta(q_0, w) = q_2$ iff $count(w, a)$ is at least 2

Proof by induction on string w

Base case: Prove the claim for $w = \varepsilon$

- We have $\delta(q_0, \varepsilon) = q_0$
- $count(w, a) = 0$
- So the claim holds

Inductive case



Induction hypothesis, $\delta(q_0, x) = q_2$ iff $count(x, a)$ is at least 2

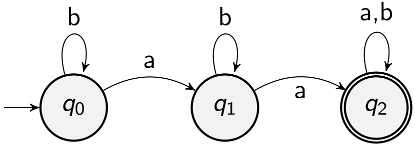
To prove, for $\sigma \in \{a, b\}, \delta(q_0, x.\sigma) = q_2$ iff $count(x.\sigma, a)$ is at least 2.

Case $\delta(q_0, x) = q_0$ and $\sigma = a$:

- In this case, by induction hypothesis, $count(x, a) < 2$
- To prove: $\delta(q_0, x.a) = q_2$ iff $count(x.a, a)$ is at least 2
- **The proof fails!!**
  - $count(x, a) = 1$ and $\delta(q_0, x) = q_0$ is consistent with the assumptions
  - In such a case, $count(x.a, a) = 2$ but $\delta(q_0, x.a) = \delta(q_0, a) = q_1$
  - Claim does not hold.

How to fix the proof?

Stronger claim



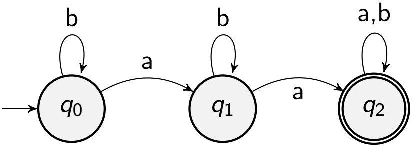
For all strings w, $\delta(q_0, w) = \begin{cases} q_0 & \text{if } count(w, a) = 0 \\ q_1 & \text{if } count(w, a) = 1 \\ q_2 & \text{if } count(w, a) \geq 2 \end{cases}$

The claim is stronger than the original claim:

- If we prove this, it follows that $\delta(q_0, w) = q_2$ iff $count(w, a) \geq 2$

Instead of just saying "strings in L lead to a final state and strings not in L lead to a non-final state", we have strengthened the claim by identifying, for each state, which strings lead to that state

Correctness Proof



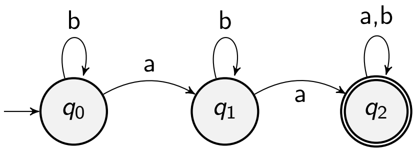
To prove: For all strings w, $\delta(q_0, w) = \begin{cases} q_0 & \text{if } count(w, a) = 0 \\ q_1 & \text{if } count(w, a) = 1 \\ q_2 & \text{if } count(w, a) \geq 2 \end{cases}$

Proof by induction on string w

Base case: Prove the claim for $w = \varepsilon$

- We have $\delta(q_0, \varepsilon) = q_0$
- $count(w, a) = 0$
- So the claim holds

Inductive case



Assume that $\delta(q_0, x)$ equals q_0 if $count(x, a) = 0$, equals q_1 if $count(x, a) = 1$ and equals q_2 if $count(x, a) \geq 2$

Prove that, for each $\sigma \in \{a, b\}$, $\delta(q_0, x.\sigma)$ equals q_0 if $count(x.\sigma, a) = 0$, equals q_1 if $count(x.\sigma, a) = 1$ and equals q_2 if $count(x.\sigma, a) \geq 2$

Proof by cases: $\delta(q_0, x)$ can be q_0 or q_1 or q_2, and σ can be a or b

Case $\delta(q_0, x) = q_0$ and $\sigma = a$:

- $count(x, a) = 0$ by induction hypothesis and so $count(x.a, a) = 1$
- $\delta(q_0, x.a) = \delta(\delta(q_0, x), a) = \delta(q_0, a) = q_1$.
- So claim holds

Remaining five cases are similar

Recipe for Proving Correctness of Automata

Given a language L described by a mathematical constraint and a DFA $M = (\{q_0, q_1, \ldots, q_n\}, \Sigma, \delta, q_0, F)$, to prove that $L(M) = L$:

- Find a precise descriptions of the sets T_0, $T_1, \ldots,$ T_n of strings that take the machine from initial state to the corresponding states
- Language L should match sets corresponding to accepting states
- Prove by induction on string w:

  For all w, $\delta(q_0, w) = q_i$ if w is in set T_i, for $i = 0, 1, \ldots, n$

- Base case
  - Prove claim for $w = \varepsilon$
- Inductive case
  - Assume: $\delta(q_0, x) = q_i$ if $x \in T_i$ for $i = 0, 1, \ldots, n$
  - To prove: for each $\sigma \in \Sigma, \delta(q_0, x.\sigma) = q_i$ if $x.\sigma \in T_i$ for $i = 0, 1, \ldots, n$.
  - Proof by case analysis on what σ is and what $\delta(q_0, x)$ is.

REGULAR LANGUAGES AND OPERATIONS

Regular Languages

- Σ^* set of all strings composed of symbols from Σ
  - Includes the empty string ε
- $L(M) \subseteq \Sigma^*$ set of strings accepted by M
- L is a regular language if there is a DFA such that $L = L(M)$

Regular expressions for pattern matching in documents (supported by all modern programming languages and editors)

So DFAs not only beautiful theory but of practical importance!!

Are all languages regular? If not which ones are?

New languages from old

- Complement
  - $\overline{L} = \{w \in \Sigma^* : w \text{ is } \textbf{not} \text{ in } L\}$
- Union
  - $L_1 \cup L_2 = \{w \in \Sigma^* : w \in L_1 \textbf{ or } w \in L_2\}$
- Intersection
  - $L_1 \cap L_2 = \{w \in \Sigma^* : w \in L_1 \textbf{ and } w \in L_2\}$
- Concatenation
  - $L_1 \circ L_2 = \{w \in \Sigma^* : w = w_1.w_2,\ w_1 \in L_1 \textbf{ and } w_2 \in L_2\}$

Complement

If L is regular is its complement $\bar{L} = \{w \in \Sigma^* : w \text{ is } \textbf{not} \text{ in } L\}$ regular?

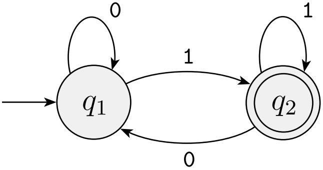
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts L
- Let $M' = (Q, \Sigma, \delta, q_0, \bar{F} = Q \setminus F)$ be the DFA where accepting states are complemented

- $w \in L(M) \iff w \notin L(M')$

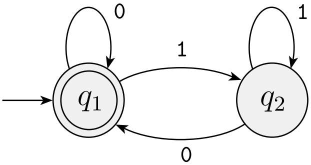
Theorem: $L(M') = \bar{L}$

Hence complement of a regular language is regular

Example



is the complement of



Union

- $L_1 = L(M_1), M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
- $L_2 = L(M_2), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

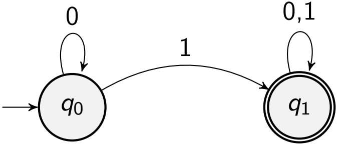
If M_1's alphabet Σ_1 is different from M_2's alphabet Σ_2 then first extend them to the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ before taking the union

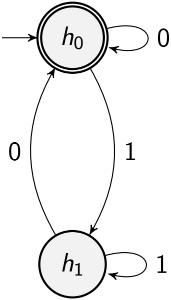
The union is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

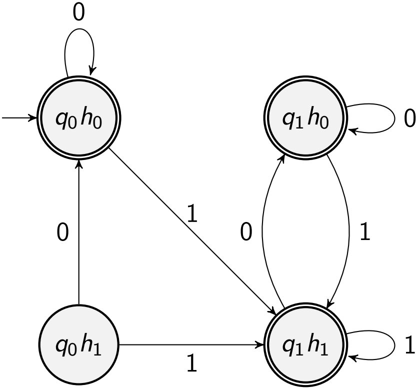
- $Q = Q_1 \times Q_2$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(q_1, q_2) : q_1 \in F_1 \textbf{ or } q_2 \in F_2\}$

Run M_1 and M_2 in parallel and accept if one of them does

Example







The union

Intersection

- $L_1 = L(M_1), M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
- $L_2 = L(M_2), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

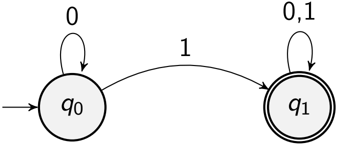
If M_1's alphabet Σ_1 is different from M_2's alphabet Σ_2 then first extend them to the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ before taking the intersection

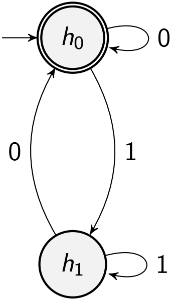
The union is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

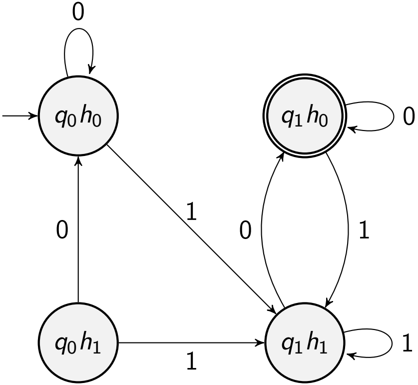
- $Q = Q_1 \times Q_2$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(q_1, q_2) : q_1 \in F_1 \textbf{ and } q_2 \in F_2\}$

Run M_1 and M_2 in parallel and accept if both of them do

Example







The intersection

Concatenation?