INF2080

1. Introduction and Regular Languages

Daniel Lupp

Universitetet i Oslo

18th January 2018

Department of Informatics

University of Oslo

Details on the Course

- consists of two parts: computability theory (first half of semester) and complexity theory (second half, held by Evgenij Thorstensen)

Details on the Course

- consists of two parts: computability theory (first half of semester) and complexity theory (second half, held by Evgenij Thorstensen)
- closely follows Michael Sipser's book "An Introduction to the Theory of Computation" (3rd International Edition) both in course content and exercises

Details on the Course

- consists of two parts: computability theory (first half of semester) and complexity theory (second half, held by Evgenij Thorstensen)
- closely follows Michael Sipser's book "An Introduction to the Theory of Computation" (3rd International Edition) both in course content and exercises
- prerequisites are INF1080/IN1150 and chapter 0 of the book (very brief and incomplete refresher soon)

Details on the Course

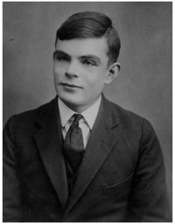
- consists of two parts: computability theory (first half of semester) and complexity theory (second half, held by Evgenij Thorstensen)
- closely follows Michael Sipser's book "An Introduction to the Theory of Computation" (3rd International Edition) both in course content and exercises
- prerequisites are INF1080/IN1150 and chapter 0 of the book (very brief and incomplete refresher soon)
- as always: lectures are useful, but doing exercises yourself is the most important! $\rightarrow$ group exercises

Setup for Computability Theory

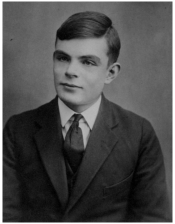
For the first half of the course:

- Thursday lecture: new theory and material
- Friday lecture: new theory and material, but mostly reserved for in-depth discussion, examples, open Q&A

So what's it all about?

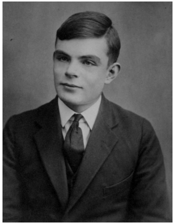


So what's it all about?



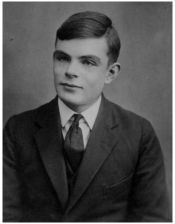
- Alan Turing
  (1912-1954)

So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

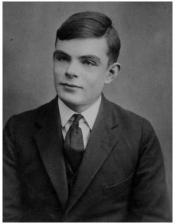
So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines

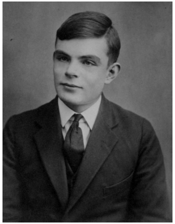
So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines

So what's it all about?

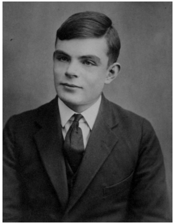


- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines

- Noam Chomsky (1928-)
- "Father" of modern linguistics

So what's it all about?



- Alan Turing (1912-1954)
- "Father" of modern computing
- very interesting (and sad) story

→ Turing machines

- Noam Chomsky (1928-)
- "Father" of modern linguistics
- classification of formal languages

→ Chomsky hierarchy

So what's it all about?

- Automata and formal languages (e.g., programming languages: programs considered as "words" in a language)

So what's it all about?

- Automata and formal languages (e.g., programming languages: programs considered as "words" in a language)
- What is an "algorithm"?

So what's it all about?

- Automata and formal languages (e.g., programming languages: programs considered as "words" in a language)
- What is an "algorithm"?
- Turing machines

So what's it all about?

- Automata and formal languages (e.g., programming languages: programs considered as "words" in a language)
- What is an "algorithm"?
- Turing machines
- Does a "solver" for a given problem always terminate?

So what's it all about?

- Automata and formal languages (e.g., programming languages: programs considered as "words" in a language)
- What is an "algorithm"?
- Turing machines
- Does a "solver" for a given problem always terminate?
- If yes, how expensive is it? ($\rightarrow$ complexity)

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
- $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**: $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- **Power Set**: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
- $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**: $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- **Power Set**: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$, example: $\mathcal{P}(\{0, 1\}) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

The Basics

- **Tuple**: ordered collection of objects
- $(a, a, b) \neq (a, b)$
- **Cartesian product**: $A \times B = \{(a, b) \mid a \in A, b \in B\}$
- **Function**: $f : A \to B$. Assigns to each element $a \in A$ a unique element $f(a) \in B$.

Finite Automata

- computational model of a computer with finite memory

Finite Automata

- computational model of a computer with finite memory
- Takes an input w and decides whether to *accept* or *reject*

Finite Automata

- computational model of a computer with finite memory
- Takes an input w and decides whether to *accept* or *reject*
- Can be used to answer such questions as "Is w a palindrome?" or "Is w a valid program in a given programming language?"

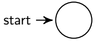
Finite Automata

- computational model of a computer with finite memory
- Takes an input w and decides whether to *accept* or *reject*
- Can be used to answer such questions as "Is w a palindrome?" or "Is w a valid program in a given programming language?"
- usual depicted as a graph for ease of reading:
- nodes represent *states* in which the automaton can be

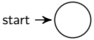
Finite Automata

- computational model of a computer with finite memory
- Takes an input w and decides whether to *accept* or *reject*
- Can be used to answer such questions as "Is w a palindrome?" or "Is w a valid program in a given programming language?"
- usual depicted as a graph for ease of reading:
- nodes represent *states* in which the automaton can be
- edges between nodes represent the transition between states given a parsed input

Finite Automata

- computational model of a computer with finite memory
- Takes an input *w* and decides whether to *accept* or *reject*
- Can be used to answer such questions as "Is *w* a palindrome?" or "Is *w* a valid program in a given programming language?"
- usual depicted as a graph for ease of reading:
- nodes represent *states* in which the automaton can be
- edges between nodes represent the transition between states given a parsed input
- always exactly one start node: 

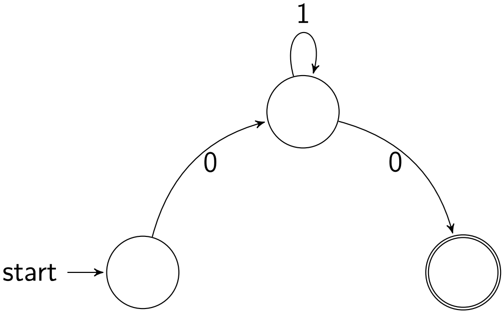
Finite Automata

- computational model of a computer with finite memory
- Takes an input w and decides whether to *accept* or *reject*
- Can be used to answer such questions as "Is w a palindrome?" or "Is w a valid program in a given programming language?"
- usual depicted as a graph for ease of reading:
- nodes represent *states* in which the automaton can be
- edges between nodes represent the transition between states given a parsed input
- always exactly one start node: 

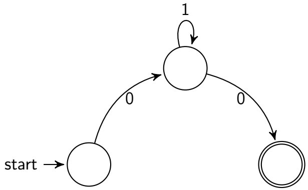
- as well as some accept states: 

Finite Automata

Example:

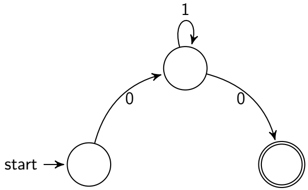


What does it mean for a finite automaton to "accept" an input w?



Finite Automata

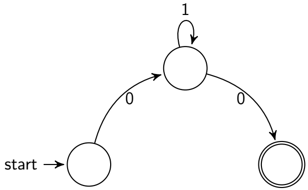
What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;

Finite Automata

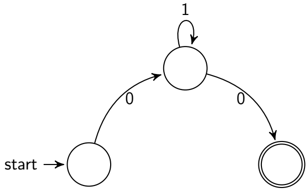
What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;

Finite Automata

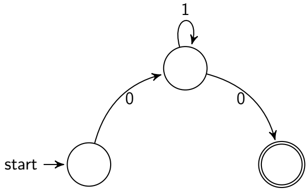
What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.

Finite Automata

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

Deterministic Finite Automata

Definition

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

1. Q is a finite set of *states*

Deterministic Finite Automata

Definition

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

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*

Deterministic Finite Automata

Definition

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

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*
3. $\delta : Q \times \Sigma \to Q$ the transition function

Deterministic Finite Automata

Definition

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

1. Q is a finite set of *states*
2. Σ is a finite set called the *alphabet*
3. $\delta : Q \times \Sigma \to Q$ the transition function
4. q_0 the start state

Deterministic Finite Automata

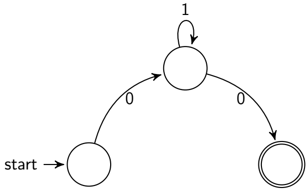
Definition

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

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

Finite Automata

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

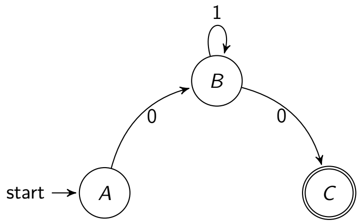
Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

Example:



For $w = w_1 w_2 w_3 w_4 w_5 = 01110$,

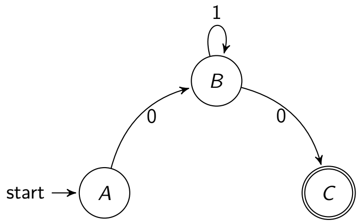
Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

Example:



For $w = w_1 w_2 w_3 w_4 w_5 = 01110$,
$s = s_0 s_1 s_2 s_3 s_4 s_5 = ABBBBC$

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

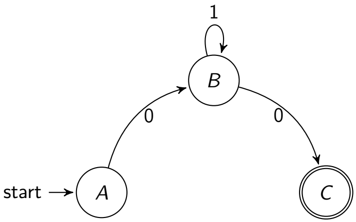
Definition

A language L is a *regular language* if there exists a DFA M that accepts each word in L, i.e., $L = \{w \mid M \text{ accepts } w\}$.

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Definition

A language L is a *regular language* if there exists a DFA M that accepts each word in L, i.e., $L = \{w \mid M \text{ accepts } w\}$.



describes

$$L_1 = \{w \mid w \text{ starts and ends with } 0, \text{ with only 1's in between}\}.$$

$\rightarrow L_1$ is regular

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.
- Concatanation: $L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$

Regular languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.
- Concatanation: $L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$
- Kleene star: $L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, \text{ each } x_i \in L_1\}$

Regular Languages

Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Proof idea: We multitask! Construct "product" automaton that runs both DFA's in parallel: $(Q_1 \times Q_2, \Sigma, \delta, F)$ where

- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$

Regular Languages

Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Proof idea: We multitask! Construct "product" automaton that runs both DFA's in parallel: $(Q_1 \times Q_2, \Sigma, \delta, F)$ where

- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$
- $F = \{(s_1, s_2) \mid s_1$ or s_2 is an accepting state$\}$ for union,
- $F = \{(s_1, s_2) \mid s_1$ and s_2 is an accepting state$\}$ for intersection

Regular Languages

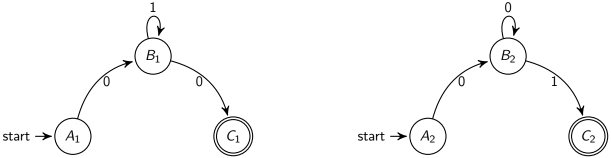
Theorem

The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Proof idea: We multitask! Construct "product" automaton that runs both DFA's in parallel: $(Q_1 \times Q_2, \Sigma, \delta, F)$ where

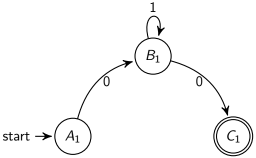
- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$
- $F = \{(s_1, s_2) \mid s_1 \text{ or } s_2 \text{ is an accepting state}\}$ for union,
- $F = \{(s_1, s_2) \mid s_1 \text{ and } s_2 \text{ is an accepting state}\}$ for intersection

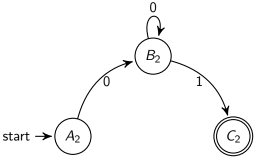
Union of regular languages: Example



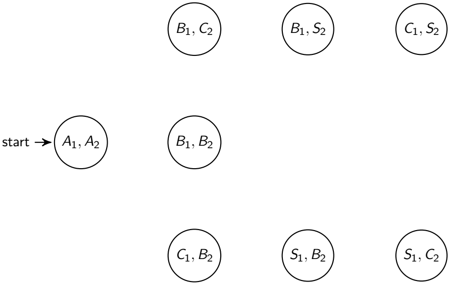
with implicit sink states S_1, S_2.

Union of regular languages: Example

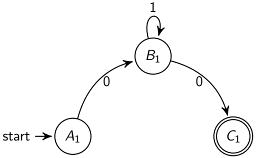


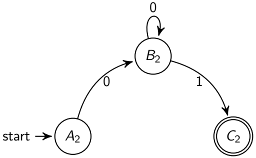


with implicit sink states S_1, S_2.

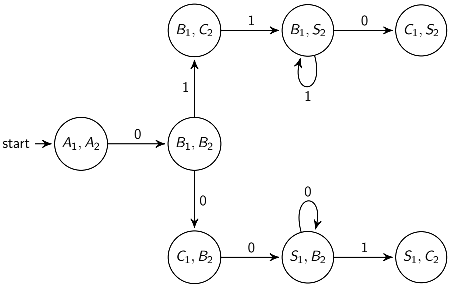


Union of regular languages: Example

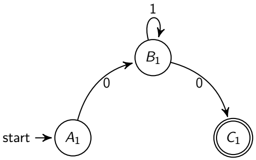


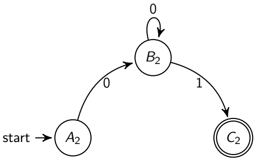


with implicit sink states S_1, S_2.

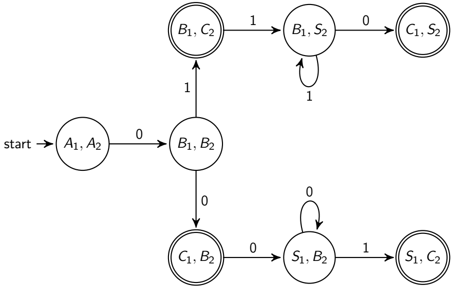


Union of regular languages: Example





with implicit sink states S_1, S_2.



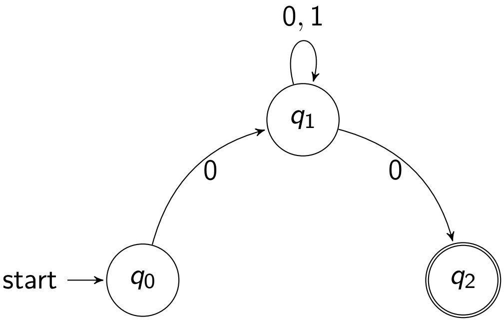
Nondeterministic Finite Automata

So far, the transition function δ gave for a given state and input symbol precisely one following state. $\rightarrow$ determinism

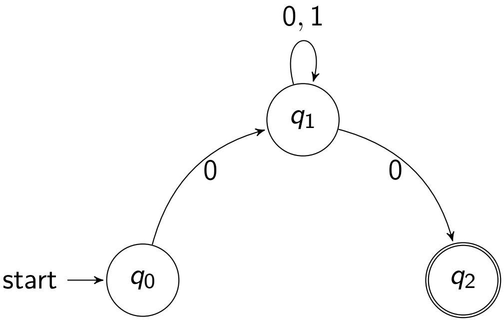
Nondeterministic Finite Automata

So far, the transition function δ gave for a given state and input symbol precisely one following state. $\rightarrow$ determinism
Now we allow for multiple possible "next" states.$\rightarrow$ nondeterminism

NFA - An example

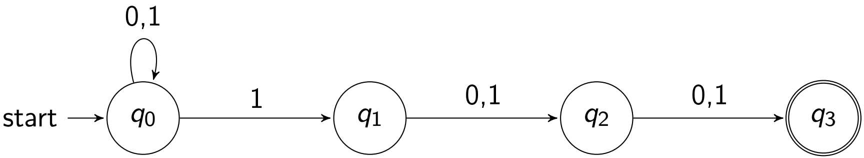


NFA - An example

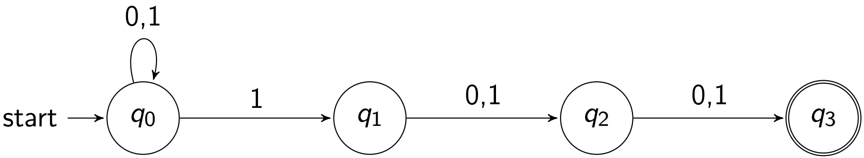


Language consists of all 0,1 sequences starting and ending with 0.

NFA - Another example

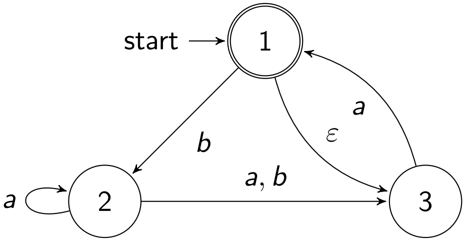


NFA - Another example



Language consists of all 0,1 sequences with a 1 in the third position from the end.

NFA - an example with empty transitions



Nondeterministic Finite Automata

Definition

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, and
4. $F \subseteq Q$ is the set of accepting states.

NFA

- First notice that DFA's are special cases of NFA's.

NFA

- First notice that DFA's are special cases of NFA's.
- DFA's accept regular languages, but what languages do NFA's accept?

NFA

- First notice that DFA's are special cases of NFA's.
- DFA's accept regular languages, but what languages do NFA's accept?
- As it turns out: regular languages! In other words, in a sense, DFA=NFA.

DFA=NFA

Theorem

Every NFA $N = (Q, \Sigma, \delta, q_0, F)$ *has an equivalent DFA* $M = (Q', \Sigma, \delta', q_0', F')$.

DFA=NFA

Theorem

Every NFA $N = (Q, \Sigma, \delta, q_0, F)$ *has an equivalent DFA* $M = (Q', \Sigma, \delta', q_0', F')$.

Proof:

- Each state in the NFA has multiple possible following states. We need to simultaneously keep track of all these possible following states in *one* state in the DFA.

DFA=NFA

Theorem

Every NFA $N = (Q, \Sigma, \delta, q_0, F)$ *has an equivalent DFA* $M = (Q', \Sigma, \delta', q'_0, F')$.

Proof:

- Each state in the NFA has multiple possible following states. We need to simultaneously keep track of all these possible following states in *one* state in the DFA.
- Since the "set of possible following states" in the NFA could be any subset of the state set Q, the DFA's state set Q' must be $\mathcal{P}(Q)$.

DFA=NFA

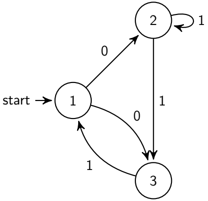
Theorem

Every NFA $N = (Q, \Sigma, \delta, q_0, F)$ *has an equivalent DFA* $M = (Q', \Sigma, \delta', q_0', F')$.

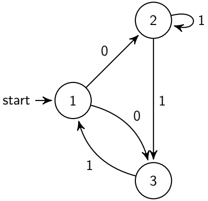
Proof:

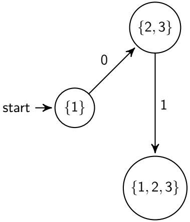
- Each state in the NFA has multiple possible following states. We need to simultaneously keep track of all these possible following states in *one* state in the DFA.
- Since the "set of possible following states" in the NFA could be any subset of the state set Q, the DFA's state set Q' must be $\mathcal{P}(Q)$.
- Let us first assume there are no ε transitions.

DFA=NFA: Example



DFA=NFA: Example





NFA=DFA

Then $q_0' = \{q_0\}$.
What about the transition function δ'?

NFA=DFA

Then $q_0' = \{q_0\}$.
What about the transition function δ'?

- a state R in the DFA M corresponds to a set of states in the NFA N. So for an input w_i at state R, we need to consider *all* possible following states to the set of possible states R.

NFA=DFA

Then $q'_0 = \{q_0\}$.

What about the transition function δ'?

- a state R in the DFA M corresponds to a set of states in the NFA N. So for an input w_i at state R, we need to consider *all* possible following states to the set of possible states R. Or, more formally,

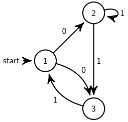
$$\begin{aligned}\delta'(R, w_i) &= \bigcup_{r \in R} \delta(r, w_i) \\ &= \{q \in Q \mid q \in \delta(r, w_i) \text{ for some } r \in R\}\end{aligned}$$

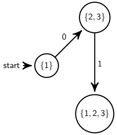
NFA=DFA

Then $q_0' = \{q_0\}$.
What about the transition function δ'?

- a state R in the DFA M corresponds to a set of states in the NFA N. So for an input w_i at state R, we need to consider *all* possible following states to the set of possible states R. Or, more formally,

$$\begin{aligned}\delta'(R, w_i) &= \bigcup_{r \in R} \delta(r, w_i) \\ &= \{q \in Q \mid q \in \delta(r, w_i) \text{ for some } r \in R\}\end{aligned}$$





NFA=DFA

- Since an NFA accepts an input if *any* of the possible computations ends in an accept state, $F' = \{R \subseteq Q \mid R \text{ contains a state } r \in F\}$.

NFA=DFA

- Since an NFA accepts an input if *any* of the possible computations ends in an accept state, $F' = \{R \subseteq Q \mid R \text{ contains a state } r \in F\}$.
- Almost done! Now we need to adjust what we did in order to take ε transitions into account. To that end, let
  $E(R) = \{q \mid q \text{ can be reached from } R \text{ with 0 or more } \varepsilon \text{ transitions}\}$ for $R \subseteq Q$.

NFA=DFA

- Since an NFA accepts an input if *any* of the possible computations ends in an accept state, $F' = \{R \subseteq Q \mid R \text{ contains a state } r \in F\}$.
- Almost done! Now we need to adjust what we did in order to take ε transitions into account. To that end, let
  $E(R) = \{q \mid q \text{ can be reached from } R \text{ with 0 or more } \varepsilon \text{ transitions}\}$ for $R \subseteq Q$.
- Then $q'_0 = E(\{q_0\})$.

NFA=DFA

- Since an NFA accepts an input if *any* of the possible computations ends in an accept state, $F' = \{R \subseteq Q \mid R \text{ contains a state } r \in F\}$.
- Almost done! Now we need to adjust what we did in order to take ε transitions into account. To that end, let
  $E(R) = \{q \mid q \text{ can be reached from } R \text{ with 0 or more } \varepsilon \text{ transitions}\}$ for $R \subseteq Q$.
- Then $q'_0 = E(\{q_0\})$.
- Transition function δ':

$$\begin{aligned}\delta'(R, w_i) &= \bigcup_{r \in R} E(\delta(r, w_i)) \\ &= \{q \in Q \mid q \in E(\delta(r, w_i)) \text{ for some } r \in R\}\end{aligned}$$

NFA=DFA

In other words, we have just proven:

Theorem

A language is regular iff (if and only if) there exists an NFA that accepts it.

Concatenation of regular languages

- So what about concatanation of languages? Given two regular languages L_1, L_2, is $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ a regular language?

Concatanation of regular languages

- So what about concatanation of languages? Given two regular languages L_1, L_2, is $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ a regular language?
- an automaton gets an input w and must "guess" how to split it $w = w_1 w_2$ in order to check L_1 and L_2 membership.

Concatanation of regular languages

- So what about concatanation of languages? Given two regular languages L_1, L_2, is $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ a regular language?
- an automaton gets an input w and must "guess" how to split it $w = w_1 w_2$ in order to check L_1 and L_2 membership.
- Example: $L_1 = \{w \mid w \text{ begins and ends with } 1\}$ and $L_2 = \{w \mid w \text{ begins and ends with } 0\}$ How to split the word $w = 10101010$? Multiple options

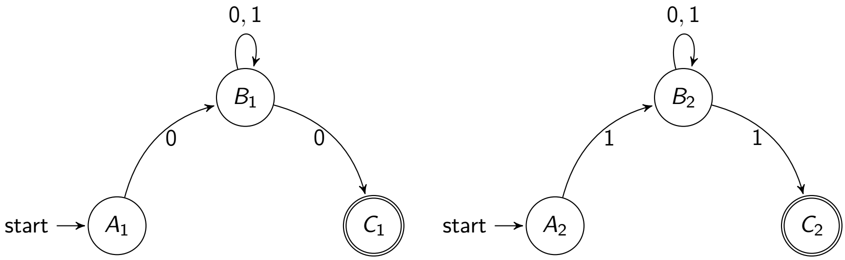
Concatenation of regular languages

- So what about concatanation of languages? Given two regular languages L_1, L_2, is $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ a regular language?
- an automaton gets an input w and must "guess" how to split it $w = w_1w_2$ in order to check L_1 and L_2 membership.
- Example: $L_1 = \{w \mid w \text{ begins and ends with } 1\}$ and $L_2 = \{w \mid w \text{ begins and ends with } 0\}$ How to split the word $w = 10101010$? Multiple options
- This "guessing" is done by using nondeterminism: NFA's try out all computational branches and accept if one leads to an accepting state.

Concatenation of regular languages

- So what about concatanation of languages? Given two regular languages L_1, L_2, is $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ a regular language?
- an automaton gets an input w and must "guess" how to split it $w = w_1 w_2$ in order to check L_1 and L_2 membership.
- Example: $L_1 = \{w \mid w$ begins and ends with $1\}$ and $L_2 = \{w \mid w$ begins and ends with $0\}$ How to split the word $w = 10101010$? Multiple options
- This "guessing" is done by using nondeterminism: NFA's try out all computational branches and accept if one leads to an accepting state.
- for concatenation: after having M_1 parse a part of the input that led to an accept state, nondeterministically "hop" into the second automaton M_2.

Concatanation of regular languages



Concatanation of regular languages