Logic and Theoretical Foundation of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University
Dependable Systems Group

Motivation

- We have now the language of propositional logic for describing problems and for deducing valid stuff

Motivation

- We have now the language of propositional logic for describing problems and for deducing valid stuff
- How does this help to accept that some problems are solvable by a computer and others not?

Motivation

- We have now the language of propositional logic for describing problems and for deducing valid stuff
- How does this help to accept that some problems are solvable by a computer and others not?
- How to get our fancy atoms into a computer?

Motivation

- We have now the language of propositional logic for describing problems and for deducing valid stuff
- How does this help to accept that some problems are solvable by a computer and others not?
- How to get our fancy atoms into a computer?
- We will develop models for computation and refine them until we get something sufficient.

Motivation

- We have now the language of propositional logic for describing problems and for deducing valid stuff
- How does this help to accept that some problems are solvable by a computer and others not?
- How to get our fancy atoms into a computer?
- We will develop models for computation and refine them until we get something sufficient.
- Recall: we are only interested in decision problems!

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters

Encoding

Definition

- **alphabet**: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters
- **string, word**: finite length sequence of elements of Σ

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters
- set of all words over Σ of length $n \in \mathbb{N}$: Σ^n

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters
- set of all words over Σ of length $n \in \mathbb{N}$: Σ^n
- set of all words over Σ longer than $n \in \mathbb{N}$: $\Sigma^{\geq n}$

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters
- set of all words over Σ of length $n \in \mathbb{N}$: Σ^n
- set of all words over Σ longer than $n \in \mathbb{N}$: $\Sigma^{\geq n}$
- set of all words over Σ: $\Sigma^* = \bigcup_{n \in \mathbb{N}_0} \Sigma^n$

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \dots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters
- set of all words over Σ of length $n \in \mathbb{N}$: Σ^n
- set of all words over Σ longer than $n \in \mathbb{N}$: $\Sigma^{\geq n}$
- set of all words over Σ: $\Sigma^* = \bigcup_{n \in \mathbb{N}_0} \Sigma^n$
- empty string: ε: $|\varepsilon| = 0$

Encoding

Definition

- alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$ finite set of symbols, letters
- string, word: finite length sequence of elements of Σ
- length of word: $|w|$ number of letters
- set of all words over Σ of length $n \in \mathbb{N}$: Σ^n
- set of all words over Σ longer than $n \in \mathbb{N}$: $\Sigma^{\geq n}$
- set of all words over Σ: $\Sigma^* = \bigcup_{n \in \mathbb{N}_0} \Sigma^n$
- empty string: ε: $|\varepsilon| = 0$
- number of $a \in \Sigma$ in a word $w \in \Sigma^*$: $|w|_a$

Operations on Words

Definition

- concatenation of words u, v: uv

Operations on Words

Definition

- concatenation of words u, v: uv
- repetition: $u^0 = \varepsilon, u^{n+1} = uu^n$

Operations on Words

Definition

- concatenation of words u, v: uv
- repetition: $u^0 = \varepsilon, u^{n+1} = uu^n$
- for a word $w \in \Sigma^n$ for $n \in \mathbb{N}_0$, $w[i]$ denotes the i$^{\text{th}}$ letter of w for all $i \in [n]_0$

Operations on Words

Definition

- concatenation of words u, v: uv
- repetition: $u^0 = \varepsilon, u^{n+1} = uu^n$
- for a word $w \in \Sigma^n$ for $n \in \mathbb{N}_0$, $w[i]$ denotes the i$^{\text{th}}$ letter of w for all $i \in [n]_0$
- denote by $w[i \ldots j]$ the word which is obtained by concatenating the letters $w[i] \ldots w[j]$ for $0 \le i \le j \le |w|$

Properties of the Concatenation

- associative: $(uv)w = u(vw)$

Properties of the Concatenation

- associative: $(uv)w = u(vw)$
- ε is neutral element: $\varepsilon u = u\varepsilon = u$

Properties of the Concatenation

- associative: $(uv)w = u(vw)$
- ε is neutral element: $\varepsilon u = u\varepsilon = u$
- $|xy| = |x| + |y|$

Properties of the Concatenation

- associative: $(uv)w = u(vw)$
- ε is neutral element: $\varepsilon u = u\varepsilon = u$
- $|xy| = |x| + |y|$
- $x^{m+n} = x^m x^n$

Properties of the Concatenation

- associative: $(uv)w = u(vw)$
- ε is neutral element: $\varepsilon u = u\varepsilon = u$
- $|xy| = |x| + |y|$
- $x^{m+n} = x^m x^n$

$\Rightarrow \Sigma^*$ is a free monoid

Properties of the Concatenation

- associative: $(uv)w = u(vw)$
- ε is neutral element: $\varepsilon u = u\varepsilon = u$
- $|xy| = |x| + |y|$
- $x^{m+n} = x^m x^n$

$\Rightarrow \Sigma^*$ is a free monoid

Be careful: concatenation is not commutative!
(washing-machine $\neq$ machine-washing)

Finite Automata and Regular Sets

Intuition for Automata

What is an automaton?

- Input

Intuition for Automata

What is an automaton?

- Input
- States (snapshot of time)

Intuition for Automata

What is an automaton?

- Input
- States (snapshot of time)
- Transitions (changes of states)

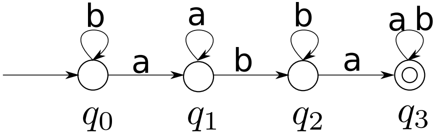
Finite Automata

Definition (Deterministic Finite Automata)

quintuple $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ is DFA iff

- **finite** set of states Q
- input alphabet Σ
- initial/starting state $q_0 \in Q$
- transition **function** $\delta : Q \times \Sigma \to Q$
- final/accepting states $F \subseteq Q$

Visualisation



	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_2
q_3F	q_3	q_3

Acceptance of a DFA

Input: word w and DFA $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$

Does $\mathscr{A}$ accept w?

Acceptance of a DFA

Input: word w and DFA $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$

Does $\mathscr{A}$ accept w?

Definition (Transition function for words)

$\hat{\delta} : Q \times \Sigma^* \to Q$ extension of δ:

- $\hat{\delta}(q, \varepsilon) = q$ for $q \in Q$
- $\hat{\delta}(q, \mathsf{a}x) = \hat{\delta}(\delta(q, \mathsf{a}), x)$ for $q \in Q, x \in \Sigma^*, a \in \Sigma$

Acceptance of a DFA

Input: word w and DFA $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$

Does $\mathscr{A}$ accept w?

Definition (Transition function for words)

$\hat{\delta} : Q \times \Sigma^* \to Q$ extension of δ:

- $\hat{\delta}(q, \varepsilon) = q$ for $q \in Q$
- $\hat{\delta}(q, \mathsf{a}x) = \hat{\delta}(\delta(q, \mathsf{a}), x)$ for $q \in Q, x \in \Sigma^*, a \in \Sigma$

$\mathscr{A}$ accepts $w \in \Sigma^*$: $\hat{\delta}(q_0, w) \in F$ (otherwise reject)

Language of an Automaton and Regular Set of Words

Definition

$\mathscr{A}$'s language: $L(\mathscr{A}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$

Definition

$A \subseteq \Sigma^*$ regular: $\exists \mathscr{A}$ DFA: $A = L(\mathscr{A})$

Regular Sets

Closure Properties

Lemma

A, B regular sets $\Longrightarrow$

Closure Properties

Lemma

A, B regular sets $\Rightarrow$

- *$A \cup B$ regular*

Closure Properties

Lemma

A, B regular sets $\Rightarrow$

- *$A \cup B$ regular*
- *$A \cap B$ regular*

Closure Properties

Lemma

A, B *regular sets* $\Longrightarrow$

- $A \cup B$ *regular*
- $A \cap B$ *regular*
- $\Sigma^* \backslash A$ *regular*

Closure Properties

Lemma

A, B *regular sets* $\Rightarrow$

- $A \cup B$ *regular*
- $A \cap B$ *regular*
- $\Sigma^* \setminus A$ *regular*
- AB *regular*

Closure Properties

Lemma

A, B regular sets $\Rightarrow$

- $A \cup B$ *regular*
- $A \cap B$ *regular*
- $\Sigma^* \setminus A$ *regular*
- AB *regular*
- A^* *regular*

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept
- $\mathscr{C} = (Q_\cap, \Sigma, \delta_\cap, q_0^\cap, F_\cap)$ with

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept
- $\mathscr{C} = (Q_\cap, \Sigma, \delta_\cap, q_0^\cap, F_\cap)$ with
  - $Q_\cap = Q_A \times Q_B$

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept
- $\mathscr{C} = (Q_\cap, \Sigma, \delta_\cap, q_0^\cap, F_\cap)$ with
  - $Q_\cap = Q_A \times Q_B$
  - $F_\cap = F_A \times F_B$

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept
- $\mathscr{C} = (Q_\cap, \Sigma, \delta_\cap, q_0^\cap, F_\cap)$ with
  - $Q_\cap = Q_A \times Q_B$
  - $F_\cap = F_A \times F_B$
  - $q_0^\cap = (q_0^A, q_0^B)$

Proof of the Regularity of $A \cap B$

- A regular $\Rightarrow \exists \mathscr{A} = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ with $L(\mathscr{A}) = A$
- B regular $\Rightarrow \exists \mathscr{B} = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ with $L(\mathscr{B}) = B$
- plan: walking simultaneously in both automata, accepting if both accept
- $\mathscr{C} = (Q_\cap, \Sigma, \delta_\cap, q_0^\cap, F_\cap)$ with
  - $Q_\cap = Q_A \times Q_B$
  - $F_\cap = F_A \times F_B$
  - $q_0^\cap = (q_0^A, q_0^B)$
  - $\delta_\cap((q, q'), a) = (\delta_A(q, a), \delta_B(q', a))$ (extension as usual)

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_\cap(q_0^\cap, x) \in F_\cap$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_\cap(q_0^\cap, x) \in F_\cap$
- $\Leftrightarrow \hat{\delta}_\cap((q_0^A, q_0^B), x) \in F_A \times F_B$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_\cap(q_0^\cap, x) \in F_\cap$
- $\Leftrightarrow \hat{\delta}_\cap((q_0^A, q_0^B), x) \in F_A \times F_B$
- $\Leftrightarrow (\hat{\delta}_A(q_0^A, x), \hat{\delta}_B(q_0^B, x)) \in F_A \times F_B$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_{\cap}(q_0^{\cap}, x) \in F_{\cap}$
- $\Leftrightarrow \hat{\delta}_{\cap}((q_0^A, q_0^B), x) \in F_A \times F_B$
- $\Leftrightarrow (\hat{\delta}_A(q_0^A, x), \hat{\delta}_B(q_0^B, x)) \in F_A \times F_B$
- $\Leftrightarrow \hat{\delta}_A(q_0^A, x) \in F_A$ and $\hat{\delta}_B(q_0^B, x) \in F_B$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_\cap(q_0^\cap, x) \in F_\cap$
- $\Leftrightarrow \hat{\delta}_\cap((q_0^A, q_0^B), x) \in F_A \times F_B$
- $\Leftrightarrow (\hat{\delta}_A(q_0^A, x), \hat{\delta}_B(q_0^B, x)) \in F_A \times F_B$
- $\Leftrightarrow \hat{\delta}_A(q_0^A, x) \in F_A$ and $\hat{\delta}_B(q_0^B, x) \in F_B$
- $\Leftrightarrow x \in L(\mathscr{A})$ and $x \in L(\mathscr{B})$

Cont. - Regularity of $A \cap B$

Proof of $L(\mathscr{C}) = L(\mathscr{A}) \cap L(\mathscr{B})$:

- $x \in L(\mathscr{C})$
- $\Leftrightarrow \hat{\delta}_{\cap}(q_0^{\cap}, x) \in F_{\cap}$
- $\Leftrightarrow \hat{\delta}_{\cap}((q_0^A, q_0^B), x) \in F_A \times F_B$
- $\Leftrightarrow (\hat{\delta}_A(q_0^A, x), \hat{\delta}_B(q_0^B, x)) \in F_A \times F_B$
- $\Leftrightarrow \hat{\delta}_A(q_0^A, x) \in F_A$ and $\hat{\delta}_B(q_0^B, x) \in F_B$
- $\Leftrightarrow x \in L(\mathscr{A})$ and $x \in L(\mathscr{B})$
- $\Leftrightarrow x \in L(\mathscr{A}) \cap L(\mathscr{B})$

$\square$

Regular Sets are closed under Morphisms

Definition

Σ, Γ alphabets:

- $h : \Sigma^* \longrightarrow \Gamma^*$ (homo)morphism:
  $\forall x, y \in \Sigma^* : h(xy) = h(x)h(y)$

Regular Sets are closed under Morphisms

Definition

Σ, Γ alphabets:

- $h : \Sigma^* \longrightarrow \Gamma^*$ (homo)morphism:
  $\forall x, y \in \Sigma^* : h(xy) = h(x)h(y)$

Lemma

- $h(\varepsilon) = \varepsilon$

Regular Sets are closed under Morphisms

Definition

Σ, Γ alphabets:

- $h : \Sigma^* \longrightarrow \Gamma^*$ (homo)morphism:
  $\forall x, y \in \Sigma^* : h(xy) = h(x)h(y)$

Lemma

- $h(\varepsilon) = \varepsilon$
- *morphism h uniquely determined by values on Σ*

Morphisms and Regular Sets

$h : \Sigma^* \longrightarrow \Gamma^*$ morphism, $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$

Definition

- image of A under h: $h(A) := \{h(x) | \, x \in A\}$
- preimage of B under h: $h^{-1}(B) := \{x | \, h(x) \in B\}$

Morphisms and Regular Sets

$h : \Sigma^* \longrightarrow \Gamma^*$ morphism, $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$

Definition

- image of A under h: $h(A) := \{h(x) \mid x \in A\}$
- preimage of B under h: $h^{-1}(B) := \{x \mid h(x) \in B\}$

Theorem

- $B \subseteq \Gamma^*$ *regular* $\Rightarrow h^{-1}(B)$ *regular*
- $A \subseteq \Sigma^*$ *regular* $\Rightarrow h(A)$ *regular*

Are DFAs what we wanted to describe computation?

- DFAs have nice closure properties

Are DFAs what we wanted to describe computation?

- DFAs have nice closure properties
- nicely to visualise

Are DFAs what we wanted to describe computation?

- DFAs have nice closure properties
- nicely to visualise
- can we express every problem by a DFA such that this automaton decides whether a property holds or not?

Are DFAs what we wanted to describe computation?

- DFAs have nice closure properties
- nicely to visualise
- can we express every problem by a DFA such that this automaton decides whether a property holds or not?
- Consider the problem: *Has a word w the same number of* a *and* b*?*

Are DFAs what we wanted to describe computation?

- DFAs have nice closure properties
- nicely to visualise
- can we express every problem by a DFA such that this automaton decides whether a property holds or not?
- Consider the problem: *Has a word w the same number of* a *and* b*?*
- We need something more powerful?

Non-deterministic Finite Automata

Motivation

- What is non-determinism? Why did we call the previous automaton **deterministic finite automaton**?

Motivation

- What is non-determinism? Why did we call the previous automaton **deterministic finite automaton**?
- Formal difference between determinism and non-determinism?

Motivation

- What is non-determinism? Why did we call the previous automaton **deterministic finite automaton**?
- Formal difference between determinism and non-determinism?
- How can we adopt this to finite automata?

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)
- **non-determinism**: input does **not** determine output uniquely (not a function anymore)

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)
- **non-determinism**: input does **not** determine output uniquely (not a function anymore)
  - several choices for a path

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)
- **non-determinism**: input does **not** determine output uniquely (not a function anymore)
  - several choices for a path
  - guessing

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)
- **non-determinism**: input does **not** determine output uniquely (not a function anymore)
  - several choices for a path
  - guessing
  - choosing by random

Informal Answers

- **determinism**: input determines output (transitions can be described as functions)
- **non-determinism**: input does **not** determine output uniquely (not a function anymore)
  - several choices for a path
  - guessing
  - choosing by random
- previous automaton defined by transition-**function** $\Rightarrow$ determinism

Non-Deterministic Finite Automata

Definition (NFA)

quintuple $\mathscr{A} = (Q, \Sigma, \Delta, q_0, F)$ is NFA iff

- **finite** set of states Q

Non-Deterministic Finite Automata

Definition (NFA)

quintuple $\mathscr{A} = (Q, \Sigma, \Delta, q_0, F)$ is NFA iff

- **finite** set of states Q
- finite alphabet Σ

Non-Deterministic Finite Automata

Definition (NFA)

quintuple $\mathscr{A} = (Q, \Sigma, \Delta, q_0, F)$ is NFA iff

- **finite** set of states Q
- finite alphabet Σ
- transition-**relation** $\Delta \subseteq Q \times \Sigma \times Q$

Non-Deterministic Finite Automata

Definition (NFA)

quintuple $\mathscr{A} = (Q, \Sigma, \Delta, q_0, F)$ is NFA iff

- **finite** set of states Q
- finite alphabet Σ
- transition-**relation** $\Delta \subseteq Q \times \Sigma \times Q$
- starting/initial state $q_0 \in Q$

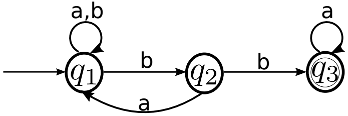
Non-Deterministic Finite Automata

Definition (NFA)

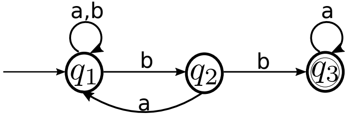
quintuple $\mathscr{A} = (Q, \Sigma, \Delta, q_0, F)$ is NFA iff

- **finite** set of states Q
- finite alphabet Σ
- transition-**relation** $\Delta \subseteq Q \times \Sigma \times Q$
- starting/initial state $q_0 \in Q$
- set of final states $F \subseteq Q$

Example for a NFA

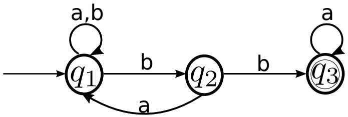


Example for a NFA



transitions:

Example for a NFA



transitions:

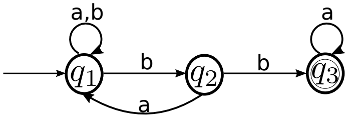
as relation

$(q_1, a, q_1), (q_1, b, q_1), (q_1, b, q_2)$

$(q_2, a, q_1), (q_2, b, q_3)$

(q_3, a, q_3)

Example for a NFA



transitions:

as relation

$(q_1, a, q_1), (q_1, b, q_1), (q_1, b, q_2)$

$(q_2, a, q_1), (q_2, b, q_3)$

(q_3, a, q_3)

as function $\Delta' : Q \times \Sigma \longrightarrow \mathscr{P}(Q)$

$(q_1, a) \mapsto \{q_1\}$

$(q_1, b) \mapsto \{q_1, q_2\}$

$(q_2, a) \mapsto \{q_1\}$

$(q_2, b) \mapsto \{q_3\}$

$(q_3, a) \mapsto \{q_3\}$

+ remaining to the empty set

Acceptance by an NFA

Definition (Extending Δ)

$\hat{\Delta} : \mathscr{P}(Q) \times \Sigma^* \to \mathscr{P}(Q)$ with

Acceptance by an NFA

Definition (Extending Δ)

$\hat{\Delta} : \mathscr{P}(Q) \times \Sigma^* \longrightarrow \mathscr{P}(Q)$ with

- $\hat{\Delta}(A, \varepsilon) = A$

Acceptance by an NFA

Definition (Extending Δ)

$\hat{\Delta} : \mathscr{P}(Q) \times \Sigma^* \longrightarrow \mathscr{P}(Q)$ with

- $\hat{\Delta}(A, \varepsilon) = A$
- $\hat{\Delta}(A, \mathrm{a}x) = \bigcup_{q \in \Delta(A,\mathrm{a})} \hat{\Delta}(q, x)$

Acceptance by an NFA

Definition (Extending Δ)

$\hat{\Delta} : \mathscr{P}(Q) \times \Sigma^* \longrightarrow \mathscr{P}(Q)$ with

- $\hat{\Delta}(A, \varepsilon) = A$
- $\hat{\Delta}(A, \mathsf{a}x) = \bigcup_{q \in \Delta(A,\mathsf{a})} \hat{\Delta}(q, x)$

Definition (Acceptance, Language)

- $\mathcal{N}$ accepts $x \in \Sigma^*$: $\hat{\Delta}(Q_0, x) \cap F \neq \emptyset$

Acceptance by an NFA

Definition (Extending Δ)

$\hat{\Delta} : \mathscr{P}(Q) \times \Sigma^* \longrightarrow \mathscr{P}(Q)$ with

- $\hat{\Delta}(A, \varepsilon) = A$
- $\hat{\Delta}(A, \mathsf{a}x) = \bigcup_{q \in \Delta(A,\mathsf{a})} \hat{\Delta}(q, x)$

Definition (Acceptance, Language)

- $\mathcal{N}$ accepts $x \in \Sigma^*$: $\hat{\Delta}(Q_0, x) \cap F \neq \emptyset$
- language $L(\mathcal{N}) = \{x \in \Sigma^* | \mathcal{N} \text{ accepts } x\}$

Properties of NFA and DFA

- each DFA is an NFA

Properties of NFA and DFA

- each DFA is an NFA
- $\forall x, y \in \Sigma^* \forall A \subseteq Q : \hat{\Delta}(A, xy) = \hat{\Delta}(\hat{\Delta}(A, x), y)$

Properties of NFA and DFA

- each DFA is an NFA
- $\forall x, y \in \Sigma^* \forall A \subseteq Q : \hat{\Delta}(A, xy) = \hat{\Delta}(\hat{\Delta}(A, x), y)$
- $\hat{\Delta}$ commutes with $\cup$: $\forall n \in \mathbb{N} \forall A_1, \ldots, A_n \subseteq Q$:

$$\hat{\Delta}\left(\bigcup_{i \in [n]} A_i, x\right) = \bigcup_{i \in [n]} \hat{\Delta}(A_i, x)$$

The Subset Construction

Did we build something more powerful?

- Only if there exists an NFA accepting a language that cannot be accepted by a DFA, we build something more powerful . . .

Did we build something more powerful?

- Only if there exists an NFA accepting a language that cannot be accepted by a DFA, we build something more powerful . . .
- . . . and we can have a look if NFAs are sufficient for the definition of *computation*

Did we build something more powerful?

- Only if there exists an NFA accepting a language that cannot be accepted by a DFA, we build something more powerful ...
- ... and we can have a look if NFAs are sufficient for the definition of *computation*
- we will show in the following that NFAs are not more powerful $\sim$ guessing does not bring any advantage in finite automata

Subset Construction (Power Set Construction)

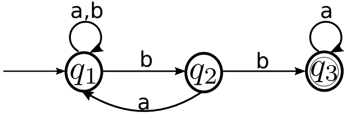
Input: NFA $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$

Output: DFA $\mathcal{A} = (\mathbb{Q}, \Sigma, \delta, \{q_0\}, \mathcal{F})$

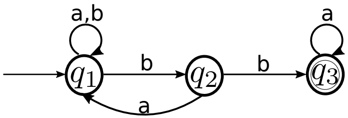
Construction:

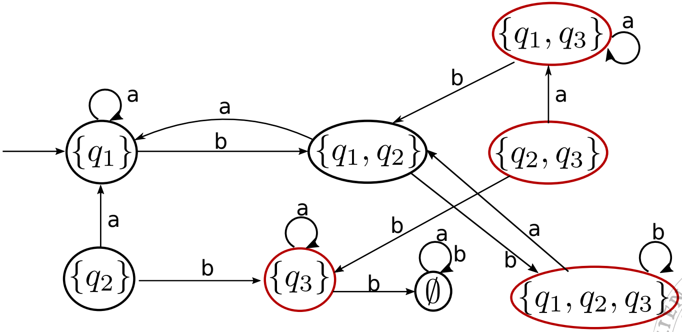
- $\mathbb{Q} = \mathcal{P}(Q)$
- $\delta : \mathbb{Q} \times \Sigma \to \mathbb{Q}, (R, a) \mapsto \{q' \in Q \mid \exists q \in R : (q, a, q') \in \Delta\}$
- $\mathcal{F} = \{S \in \mathbb{Q} \mid S \cap F \neq \emptyset\}$

Example



Example





Connection: DFA and NFA

Lemma

The constructed DFA is equivalent to the given NFA.

Connection: DFA and NFA

Lemma

The constructed DFA is equivalent to the given NFA.

Theorem

NFA and DFA are equivalent!

Pattern Matching - Regular Expressions

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?
How to formalize this? Can we use P as a stencil?

Problem: Given a text T and a set of words P, does T contain the words?

How to formalize this? Can we use P as a stencil?

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?

How to formalize this? Can we use P as a stencil?

In this tutorial we learn something about patterns.

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?

How to formalize this? Can we use P as a stencil?

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?

How to formalize this? Can we use P as a stencil?

Now we are getting into details on patterns.

Application for NFA/DFA

Problem: Given a text T and a set of words P, does T contain the words?

How to formalize this? Can we use P as a stencil?

Now we are getting into details on patterns.

This does not match!

Important Problems

- Does a string x match a pattern α?

Important Problems

- Does a string x match a pattern α?
- Is every set of words representable by pattern?

Important Problems

- Does a string x match a pattern α?
- Is every set of words representable by pattern?
- Are two patterns α, β equivalent, i.e. $L(\alpha) = L(\beta)$?

Important Problems

- Does a string x match a pattern α?
- Is every set of words representable by pattern?
- Are two patterns α, β equivalent, i.e. $L(\alpha) = L(\beta)$?
- Is there a reason that we called sets accepted by DFAs *regular*?

Formal Approach to Regular Expressions

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)

Formal Approach to Regular Expressions

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)
- ε is the empty word/string

Formal Approach to Regular Expressions

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)
- ε is the empty word/string
- atomic pattern every element from $\Sigma \cup \{\varepsilon, \emptyset\}$

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)
- ε is the empty word/string
- atomic pattern every element from $\Sigma \cup \{\varepsilon, \emptyset\}$
- operations: $+, \cdot, {}^*$

Formal Approach to Regular Expressions

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)
- ε is the empty word/string
- atomic pattern every element from $\Sigma \cup \{\varepsilon, \emptyset\}$
- operations: $+, \cdot, ^*$
- compound patterns for given patterns α, β: $\alpha + \beta$, $\alpha \cdot \beta$, α^*

Formal Approach to Regular Expressions

Definition

- new symbol $\emptyset$ for *the nothing* (this is not(!) ε)
- ε is the empty word/string
- atomic pattern every element from $\Sigma \cup \{\varepsilon, \emptyset\}$
- operations: $+, \cdot, ^*$
- compound patterns for given patterns α, β: $\alpha + \beta$, $\alpha \cdot \beta$, α^*
- Precedences for operators: $\cdot, ^* > +$

Matching - Semantics of Regular Expressions

Definition

$L(\alpha) = \{x \in \Sigma^* | \, x \text{ matches } \alpha\}$ defined by

Matching - Semantics of Regular Expressions

Definition

$L(\alpha) = \{x \in \Sigma^* | \, x \text{ matches } \alpha\}$ defined by

- $\forall \mathsf{a} \in \Sigma : \; L(\mathsf{a}) = \{\mathsf{a}\}$

Matching - Semantics of Regular Expressions

Definition

$L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}$ defined by

- $\forall \mathsf{a} \in \Sigma: \ L(\mathsf{a}) = \{\mathsf{a}\}$
- $L(\emptyset) = \emptyset$

Matching - Semantics of Regular Expressions

Definition

$L(\alpha) = \{x \in \Sigma^* | \, x \text{ matches } \alpha\}$ defined by

- $\forall \mathsf{a} \in \Sigma : \; L(\mathsf{a}) = \{\mathsf{a}\}$
- $L(\emptyset) = \emptyset$
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$

Matching - Semantics of Regular Expressions

Definition

$L(\alpha) = \{x \in \Sigma^* | \, x \text{ matches } \alpha\}$ defined by

- $\forall \mathsf{a} \in \Sigma : \; L(\mathsf{a}) = \{\mathsf{a}\}$
- $L(\emptyset) = \emptyset$
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha \cdot \beta) = L(\alpha\beta) = L(\alpha)L(\beta)$

Definition

$L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}$ defined by

- $\forall \mathsf{a} \in \Sigma: \ L(\mathsf{a}) = \{\mathsf{a}\}$
- $L(\emptyset) = \emptyset$
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha \cdot \beta) = L(\alpha\beta) = L(\alpha)L(\beta)$
- $L(\alpha^*) = \bigcup_{i \in \mathbb{N}_0} L(\alpha)^i$

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*

Simplifying Regular Expressions

Lemma

- *$+$ is associative, commutative, idempotent*
- *$\emptyset$ is neutral element w.r.t. $+$*

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*
- ε *is neutral element w.r.t.* $\cdot$

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*
- ε *is neutral element w.r.t.* $\cdot$
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ *and* $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*
- ε *is neutral element w.r.t.* $\cdot$
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ *and* $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$
- $\emptyset\alpha = \alpha\emptyset = \emptyset$

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*
- ε *is neutral element w.r.t.* $\cdot$
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ *and* $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$
- $\emptyset\alpha = \alpha\emptyset = \emptyset$
- $\varepsilon + \alpha^+ = \alpha^*$

Simplifying Regular Expressions

Lemma

- $+$ *is associative, commutative, idempotent*
- $\emptyset$ *is neutral element w.r.t.* $+$
- $\cdot$ *is associative*
- ε *is neutral element w.r.t.* $\cdot$
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ *and* $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$
- $\emptyset\alpha = \alpha\emptyset = \emptyset$
- $\varepsilon + \alpha^{+} = \alpha^{*}$

The proof is left to the reader.

Connection between Regularity, RegExp, Finite Automata

Theorem

$A \subseteq \Sigma^* \Rightarrow$ *following statements are equivalent:*

1. *A is regular*
2. *exists DFA $\mathcal{A}$ such that $L(\mathcal{A}) = A$*
3. *exists NFA $\mathcal{N}$ such that $L(\mathcal{N}) = A$*
4. *exists regular expression β such that $L(\beta) = A$*

Connection between Regularity, RegExp, Finite Automata

Theorem

$A \subseteq \Sigma^* \Rightarrow$ following statements are equivalent:

1. *A is regular*
2. *exists DFA $\mathcal{A}$ such that $L(\mathcal{A}) = A$*
3. *exists NFA $\mathcal{N}$ such that $L(\mathcal{N}) = A$*
4. *exists regular expression β such that $L(\beta) = A$*

This theorem is important!

Remarks on the Proof of the Theorem

- how to prove that n statements are equivalent?

Remarks on the Proof of the Theorem

- how to prove that n statements are equivalent?
- we can do that by proving $\frac{n(n+1)}{2}$ equivalences, i.e. $n(n+1)$ implications

Remarks on the Proof of the Theorem

- how to prove that n statements are equivalent?
- we can do that by proving $\frac{n(n+1)}{2}$ equivalences, i.e. $n(n+1)$ implications
- smarter: ring-proof
  - prove that the i^{th} statement implies the $(i+1)^{\text{th}}$
  - prove that the n^{th} statement implies the 1^{st}

Remarks on the Proof of the Theorem

- how to prove that n statements are equivalent?
- we can do that by proving $\frac{n(n+1)}{2}$ equivalences, i.e. $n(n+1)$ implications
- smarter: ring-proof
  - prove that the i^{th} statement implies the $(i+1)^{\text{th}}$
  - prove that the n^{th} statement implies the 1^{st}
- we have already proven 1. $\longrightarrow$ 2., 2. $\longrightarrow$ 3.

Remarks on the Proof of the Theorem

- how to prove that n statements are equivalent?
- we can do that by proving $\frac{n(n+1)}{2}$ equivalences, i.e. $n(n+1)$ implications
- smarter: ring-proof
  - prove that the i^{th} statement implies the $(i+1)^{\text{th}}$
  - prove that the n^{th} statement implies the 1^{st}
- we have already proven 1. $\to$ 2., 2. $\to$ 3.
- it remains to show: 3. $\to$ 4. and 4. $\to$ 1.

NFA $\Rightarrow$ RegExp

$\mathcal{A} = (Q, \Sigma, \Delta, S, F)$ NFA, $X \subseteq Q$, $q, q' \in Q$

NFA $\Rightarrow$ RegExp

$\mathscr{A} = (Q, \Sigma, \Delta, S, F)$ NFA, $X \subseteq Q$, $q, q' \in Q$

- regular expression for $\mathscr{A}$ is $\alpha = +_{i\in[n], j\in[m]} \alpha^{Q}_{s_i f_j}$

NFA $\Rightarrow$ RegExp

$\mathcal{A} = (Q, \Sigma, \Delta, S, F)$ NFA, $X \subseteq Q$, $q, q' \in Q$

- regular expression for $\mathcal{A}$ is $\alpha = \boldsymbol{+}_{i \in [n], j \in [m]} \alpha^{Q}_{s_i f_j}$
- induction basis: $a_1, \ldots, a_k \in \Sigma$ with $v \in \Delta(q, a_i)$
  - $q \neq q'$: $\alpha^{\emptyset}_{qq'} = \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1, \\ \emptyset & \text{otherwise} \end{cases}$

NFA $\Rightarrow$ RegExp

$\mathscr{A} = (Q, \Sigma, \Delta, S, F)$ NFA, $X \subseteq Q$, $q, q' \in Q$

- regular expression for $\mathscr{A}$ is $\alpha = \mathbf{+}_{i \in [n], j \in [m]} \alpha^{Q}_{s_i f_j}$
- induction basis: $a_1, \ldots, a_k \in \Sigma$ with $v \in \Delta(q, a_i)$
  - $q \neq q'$: $\alpha^{\emptyset}_{qq'} = \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1, \\ \emptyset & \text{otherwise} \end{cases}$
  - $q = q'$: $\alpha^{\emptyset}_{qq'} = \begin{cases} a_1 + \cdots + a_k + \varepsilon & \text{if } k \geq 1, \\ \emptyset + \varepsilon & \text{otherwise} \end{cases}$

NFA $\Longrightarrow$ RegExp

$\mathcal{A} = (Q, \Sigma, \Delta, S, F)$ NFA, $X \subseteq Q$, $q, q' \in Q$

- regular expression for $\mathcal{A}$ is $\alpha = \boldsymbol{+}_{i \in [n], j \in [m]} \alpha^{Q}_{s_i f_j}$
- induction basis: $a_1, \ldots, a_k \in \Sigma$ with $v \in \Delta(q, a_i)$
  - $q \neq q'$: $\alpha^{\emptyset}_{qq'} = \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1, \\ \emptyset & \text{otherwise} \end{cases}$
  - $q = q'$: $\alpha^{\emptyset}_{qq'} = \begin{cases} a_1 + \cdots + a_k + \varepsilon & \text{if } k \geq 1, \\ \emptyset + \varepsilon & \text{otherwise} \end{cases}$
- induction step: $X \neq \emptyset$, $r \in X$
  - $\alpha^{X}_{qq'} = \alpha^{X \setminus \{r\}}_{qq'} + \alpha^{X \setminus \{r\}}_{qr} (\alpha^{X \setminus \{r\}}_{rr})^* \alpha^{X \setminus \{r\}}_{rq'}$

Regular Expression $\longrightarrow$ Regular Set

For this proof we need an extension of NFAs (which is not an extension): transitions without reading anything

For this proof we need an extension of NFAs (which is not an extension): transitions without reading anything

Definition (NFA with ε-Transistion)

$\mathcal{N} = (Q, \Sigma, \Delta, Q_0, F)$ with Q, Σ, Q_0, F as in the *normal* NFA and

$$\Delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times Q$$

ε-transitions

Lemma

NFA with ε-transitions is ***not*** *more powerful than an NFA*

- M_ε accepts $x \in \Sigma^*$ if there exists $y \in (\Sigma \cup \{\varepsilon\})^*$ such that
  - x is obtained by deleting all occurrences of ε in y
  - y is accepted under the ordinary definition

ε-transitions

Lemma

*NFA with ε-transitions is **not** more powerful than an NFA*

- M_ε accepts $x \in \Sigma^*$ if there exists $y \in (\Sigma \cup \{\varepsilon\})^*$ such that
  - x is obtained by deleting all occurrences of ε in y
  - y is accepted under the ordinary definition

formally: deleting ε is morphism $h : (\Sigma \cup \{\varepsilon\})^* \to \Sigma$ with

- $h(a) = a$ for $a \in \Sigma$ and $h(\varepsilon) = \varepsilon$
- image and preimage of regular sets are regular

Regular Expression $\longrightarrow \varepsilon$-NFA

Thompson-Construction for regular expression r

- $r = \varepsilon \rightsquigarrow q \xrightarrow{\varepsilon} f \in F$

Regular Expression $\longrightarrow \varepsilon$-NFA

Thompson-Construction for regular expression r

- $r = \varepsilon \rightsquigarrow q \xrightarrow{\varepsilon} f \in F$
- $r = \mathsf{a} \rightsquigarrow q \xrightarrow{\mathsf{a}} f \in F$

Regular Expression $\longrightarrow \varepsilon$-NFA

Thompson-Construction for regular expression r

- $r = \varepsilon \rightsquigarrow q \xrightarrow{\varepsilon} f \in F$
- $r = \mathsf{a} \rightsquigarrow q \xrightarrow{\mathsf{a}} f \in F$
- $r = \alpha + \beta \rightsquigarrow q \xrightarrow{\varepsilon} q_0^{\mathcal{A}}, q \xrightarrow{\varepsilon} q_0^{\mathcal{B}}, q_f^{\mathcal{A}} \xrightarrow{\varepsilon} f \in F, q_f^{\mathcal{B}} \xrightarrow{\varepsilon} f \in F$
  with $\mathcal{A}$ ε-NFA for α, $\mathcal{B}$ ε-NFA for β

Regular Expression $\longrightarrow \varepsilon$-NFA

Thompson-Construction for regular expression r

- $r = \varepsilon \rightsquigarrow q \xrightarrow{\varepsilon} f \in F$
- $r = \mathsf{a} \rightsquigarrow q \xrightarrow{\mathsf{a}} f \in F$
- $r = \alpha + \beta \rightsquigarrow q \xrightarrow{\varepsilon} q_0^{\mathscr{A}}, q \xrightarrow{\varepsilon} q_0^{\mathscr{B}}, q_f^{\mathscr{A}} \xrightarrow{\varepsilon} f \in F, q_f^{\mathscr{B}} \xrightarrow{\varepsilon} f \in F$
  with $\mathscr{A}$ ε-NFA for α, $\mathscr{B}$ ε-NFA for β
- $r = \alpha \cdot \beta \rightsquigarrow q_0^{\mathscr{A}} \longrightarrow q_f^{\mathscr{A}} = q_0^{\mathscr{B}} \longrightarrow q_f^{\mathscr{B}}$

Regular Expression $\longrightarrow \varepsilon$-NFA

Thompson-Construction for regular expression r

- $r = \varepsilon \leadsto q \xrightarrow{\varepsilon} f \in F$
- $r = \mathsf{a} \leadsto q \xrightarrow{\mathsf{a}} f \in F$
- $r = \alpha + \beta \leadsto q \xrightarrow{\varepsilon} q_0^{\mathscr{A}}, q \xrightarrow{\varepsilon} q_0^{\mathscr{B}}, q_f^{\mathscr{A}} \xrightarrow{\varepsilon} f \in F, q_f^{\mathscr{B}} \xrightarrow{\varepsilon} f \in F$
  with $\mathscr{A}$ ε-NFA for α, $\mathscr{B}$ ε-NFA for β
- $r = \alpha \cdot \beta \leadsto q_0^{\mathscr{A}} \longrightarrow q_f^{\mathscr{A}} = q_0^{\mathscr{B}} \longrightarrow q_f^{\mathscr{B}}$
- $r = \alpha^* \leadsto q \xrightarrow{\varepsilon} q_0^{\mathscr{A}} \longrightarrow q_f^{\mathscr{A}} \xrightarrow{\varepsilon} f \in F, f \xrightarrow{\varepsilon} q, q_f^{\mathscr{A}} \xrightarrow{\varepsilon} q_0^{\mathscr{A}}$

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal
- NFAs and DFAs are equivalent

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal
- NFAs and DFAs are equivalent
- why keeping on talking about these automata?

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal
- NFAs and DFAs are equivalent
- why keeping on talking about these automata?
- they are elegant and small for handling problems like syntax-checks (compiler construction)

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal
- NFAs and DFAs are equivalent
- why keeping on talking about these automata?
- they are elegant and small for handling problems like syntax-checks (compiler construction)
- unfortunately, determinism is better for implementing but DFAs may have too many states

Why considering finite automata?

- we know that DFAs are not powerful enough for our goal
- NFAs and DFAs are equivalent
- why keeping on talking about these automata?
- they are elegant and small for handling problems like syntax-checks (compiler construction)
- unfortunately, determinism is better for implementing but DFAs may have too many states
- before we build a better automata model, let's try to handle the state explosion

DFA State Minimization

State Explosion and Handling it

- Subset Construction $\Rightarrow$ DFA has exponentially many states
  - NFA with $|Q| = n \in \mathbb{N} \Rightarrow$ DFA has $|\mathscr{P}(Q)| = 2^n$ states

State Explosion and Handling it

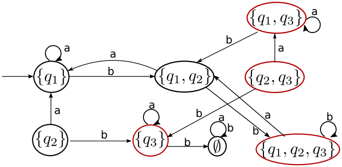
- Subset Construction $\Rightarrow$ DFA has exponentially many states
  - NFA with $|Q| = n \in \mathbb{N} \Rightarrow$ DFA has $|\mathscr{P}(Q)| = 2^n$ states
- Are they all necessary?

State Explosion and Handling it

- Subset Construction $\Rightarrow$ DFA has exponentially many states
  - NFA with $|Q| = n \in \mathbb{N} \Rightarrow$ DFA has $|\mathscr{P}(Q)| = 2^n$ states
- Are they all necessary?
  - Unfortunately, sometimes **yes**, but not always

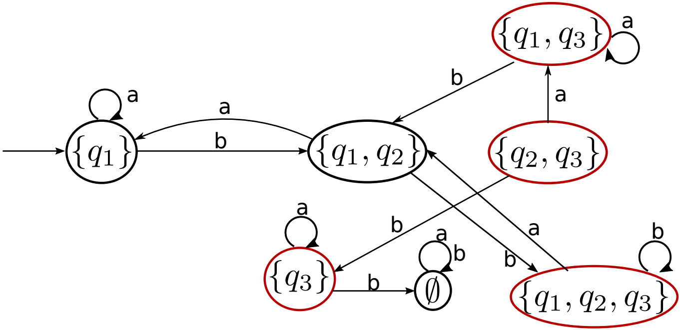
State Explosion and Handling it

- Subset Construction $\Rightarrow$ DFA has exponentially many states
  - NFA with $|Q| = n \in \mathbb{N} \Rightarrow$ DFA has $|\mathscr{P}(Q)| = 2^n$ states
- Are they all necessary?
  - Unfortunately, sometimes **yes**, but not always
- Have a look at the constructed DFA discussed here:

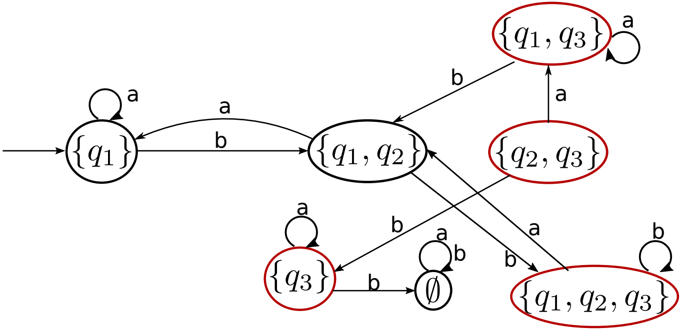


- no incoming arrow $\Rightarrow$ state is not reachable
- here: $\{q_2\}, \{q_1, q_2, q_3\}$

Cont.



Cont.



Can we reduce more? Exists something like a similar "behaviour" of states?

Equivalency of States

Assume: $q_1, q_2 \in Q$ and whatever you read starting in them you either reach with both a final state or with none

Equivalency of States

Assume: $q_1, q_2 \in Q$ and whatever you read starting in them you either reach with both a final state or with none

Definition

states $q_1, q_2 \in Q$ equivalent ($q_1 \approx q_2$):

$$\forall x \in \Sigma^* : \hat{\delta}(q_1, x) \in F \Leftrightarrow \hat{\delta}(q_2, x) \in F$$

Equivalency of States

Assume: $q_1, q_2 \in Q$ and whatever you read starting in them you either reach with both a final state or with none

Definition

states $q_1, q_2 \in Q$ equivalent ($q_1 \approx q_2$):

$$\forall x \in \Sigma^* : \hat{\delta}(q_1, x) \in F \Leftrightarrow \hat{\delta}(q_2, x) \in F$$

- $[p] = \{q \in Q | p \approx q\}$ equivalence class of p

Equivalency of States

Assume: $q_1, q_2 \in Q$ and whatever you read starting in them you either reach with both a final state or with none

Definition

states $q_1, q_2 \in Q$ equivalent ($q_1 \approx q_2$):

$$\forall x \in \Sigma^* : \hat{\delta}(q_1, x) \in F \Leftrightarrow \hat{\delta}(q_2, x) \in F$$

- $[p] = \{q \in Q | p \approx q\}$ equivalence class of p

- $\approx$ is equivalence relation (reflexive, symmetric, transitive)

Equivalency of States

Assume: $q_1, q_2 \in Q$ and whatever you read starting in them you either reach with both a final state or with none

Definition

states $q_1, q_2 \in Q$ equivalent ($q_1 \approx q_2$):

$$\forall x \in \Sigma^* : \hat{\delta}(q_1, x) \in F \Leftrightarrow \hat{\delta}(q_2, x) \in F$$

- $[p] = \{q \in Q | p \approx q\}$ equivalence class of p

- $\approx$ is equivalence relation (reflexive, symmetric, transitive)
- $\approx$ equivalence relation $\Rightarrow$ factorisation possible

Quotient Automaton

Definition

quotient automaton $\mathcal{A}/{\approx} = (Q', \Sigma, \delta', q'_0, F')$ for DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q' = \{[q] \mid q \in Q\}$
- $\delta' : Q' \times \Sigma \rightarrow Q'$ with $\delta'([q], a) = [\delta(q, a)]$
- $q'_0 = [q_0]$
- $F' = \{[q] \mid q \in F\}$

Some Properties

$p, q \in Q, x \in \Sigma^*$

- $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$

Some Properties

$p, q \in Q, x \in \Sigma^*$

- $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$
- $[p] = [q] \Rightarrow [\delta(p, a)] = [\delta(q, a)]$

Some Properties

$p, q \in Q, x \in \Sigma^*$

- $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$
- $[p] = [q] \Rightarrow [\delta(p, a)] = [\delta(q, a)]$
- $p \in F \Leftrightarrow [p] \in F'$

Some Properties

$p, q \in Q, x \in \Sigma^*$

- $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$
- $[p] = [q] \Rightarrow [\delta(p, a)] = [\delta(q, a)]$
- $p \in F \Leftrightarrow [p] \in F'$
- $\hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$

Some Properties

$p, q \in Q, x \in \Sigma^*$

- $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$
- $[p] = [q] \Rightarrow [\delta(p, a)] = [\delta(q, a)]$
- $p \in F \Leftrightarrow [p] \in F'$
- $\hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$

Theorem

For every DFA $\mathcal{A}$: $L(\mathcal{A}) = L(\mathcal{A}/\approx)$

Minimized DFA

Notice: applying the quotient construction twice **does not** reduce the states!

Minimized DFA

Notice: applying the quotient construction twice **does not** reduce the states!

Definition

$\mathscr{A}_{\text{min}}$ **minimized DFA** for $\mathscr{A}$:

1. eliminate all non-reachable states
2. build the quotient automaton

Minimized DFA

Notice: applying the quotient construction twice **does not** reduce the states!

Definition

$\mathscr{A}_{\mathrm{min}}$ **minimized DFA** for $\mathscr{A}$:

1. eliminate all non-reachable states
2. build the quotient automaton

there exists a nice algorithm for detecting the equivalent states

Minimization Algorithm

DFA $\mathscr{A}$ with no unreachable states

Minimization Algorithm

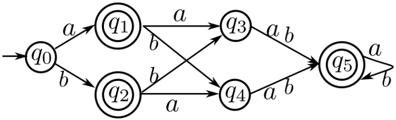
DFA $\mathscr{A}$ with no unreachable states

1. write down a table with all pairs $\{p, q\}$
2. **mark** $\{p, q\}$ if $p \in F$ and $q \notin F$ and vice versa
3. **Repeat Until** no changes
   - $\exists \{p, q\}$ unmarked $\exists a \in \Sigma$:
     $\{\delta(p, a), \delta(q, a)\}$ marked $\Rightarrow$ **mark** $\{p, q\}$

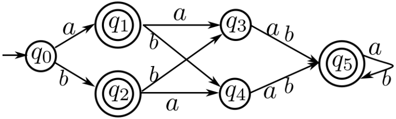
Remarks

- $\{p, q\}$ marked $\Rightarrow$ not equivalent
- every pair may be checked several times
- only stop if really nothing changes
- ≥ 1 mark in each iteration $\Rightarrow$ at most $\binom{|Q|}{2}$ iterations

Example

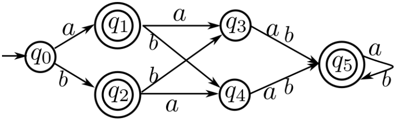


Example



	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1		-	-	-	-	-
q_2			-	-	-	-
q_3				-	-	-
q_4					-	-
q_5						-

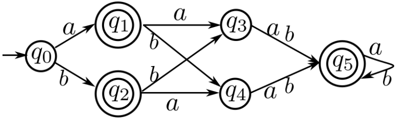
Example



Step 2: 1 final state, 1 non-final state

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×			×	×	-

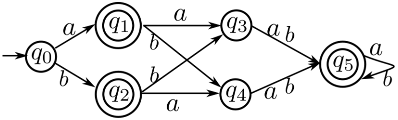
Example



Step 3: for marking $\{q_0, q_3\}$,
$\{\delta(q_0, a/b), \delta(q_3, a/b)\} = \{q_1/q_2, q_5\}$ needs to be marked $\Rightarrow$ nothing

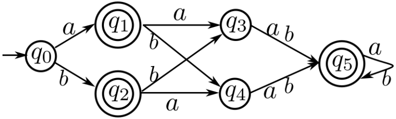
	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×			×	×	-

Example



Step 3: for marking $\{q_0, q_4\}$,
$\{\delta(q_0, a/b), \delta(q_4, a/b)\} = \{q_1/q_2, q_5\}$ needs to $\Rightarrow$ nothing

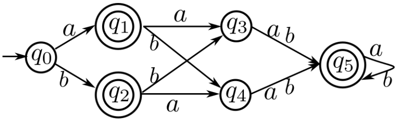
	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×			×	×	-



Step 3: for marking $\{q_1, q_2\}$,
$\{\delta(q_1, a/b), \delta(q_2, a/b)\} = \{q_3/q_4, q_4/q_3\}$ needs to be marked $\Rightarrow$ nothing

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×			×	×	-

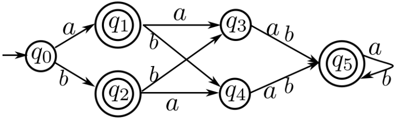
Example



Step 3: for marking $\{q_1, q_5\}$,
$\{\delta(q_1, a/b), \delta(q_5, a/b)\} = \{q_3/q_4, q_5\}$ needs to be marked $\Rightarrow$ mark

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	$\times$	-	-	-	-	-
q_2	$\times$		-	-	-	-
q_3		$\times$	$\times$	-	-	-
q_4		$\times$	$\times$		-	-
q_5	$\times$	$\times$		$\times$	$\times$	-

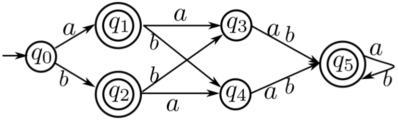
Example



Step 3: for marking $\{q_2, q_5\}$,
$\{\delta(q_2, a/b), \delta(q_5, a/b)\} = \{q_4/q_3, q_5\}$ needs to be marked $\Rightarrow$ mark

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×	×	×	×	×	-

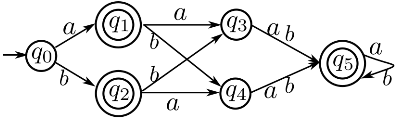
Example



Step 3: for marking $\{q_3, q_4\}$, $\{\delta(q_3, a/b), \delta(q_4, a/b)\} = \{q_5, q_5\}$ needs to be marked $\Rightarrow$ nothing

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3		×	×	-	-	-
q_4		×	×		-	-
q_5	×	×	×	×	×	-

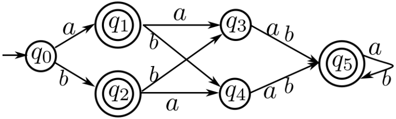
Example



Step 3: for marking $\{q_0, q_3\}$,
$\{\delta(q_0, a/b), \delta(q_3, a/b)\} = \{q_1/q_2, q_5\}$ needs to be marked $\Rightarrow$ mark

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3	×	×	×	-	-	-
q_4		×	×		-	-
q_5	×	×	×	×	×	-

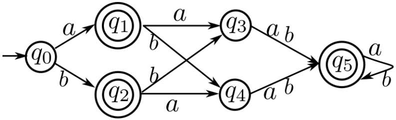
Example



Step 3: for marking $\{q_0, q_4\}$,
$\{\delta(q_0, a/b), \delta(q_4, a/b)\} = \{q_1/q_2, q_5\}$ needs to be marked $\Rightarrow$ mark

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3	×	×	×	-	-	-
q_4	×	×	×		-	-
q_5	×	×	×	×	×	-

Example



no more changes $\Rightarrow$ q_1, q_2 and q_3, q_4 are equivalent

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	-	-	-	-	-	-
q_1	×	-	-	-	-	-
q_2	×		-	-	-	-
q_3	×	×	×	-	-	-
q_4	×	×	×		-	-
q_5	×	×	×	×	×	-

Example