

# Foundations of Informatics: a Bridging Course

## Week 3: Formal Languages and Semantics

### Part A: Regular Languages

Thomas Noll

Software Modeling and Verification Group (MOVES)



[noll@cs.rwth-aachen.de](mailto:noll@cs.rwth-aachen.de)

<http://cosec.bit.uni-bonn.de/students/teaching/11us/11us-bridgingcourse/>

<http://www-i2.informatik.rwth-aachen.de/i2/b-it11/>

b-it, Bonn, Winter Semester 2011/12

- Schedule:
  - lecture 9:00-10:30, 11:00-12:30 (Mon-Fri)
    - 9:30-11:00, 11:15-12:45?
  - exercises 14:00-14:45, 15:15-16:00 (Mon-Thu)
    - 14:00-15:30?
- Examination at end of week 4 (16 March)
- Please ask questions!

- ① Regular Languages
- ② Context-Free Languages
- ③ Processes and Concurrency

- J.E. Hopcroft, R. Motwani, J.D. Ullmann: *Introduction to Automata Theory, Languages, and Computation*, 2nd ed., Addison-Wesley, 2001
- A. Asteroth, C. Baier: *Theoretische Informatik*, Pearson Studium, 2002 [in German]
- <http://www.jflap.org/>  
(software for experimenting with formal languages and automata)

- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

- Computer systems transform data
- Data encoded as (binary) **words**

⇒ Data sets = sets of words = **formal languages**,  
data transformations = **functions on words**

- Computer systems transform data
- Data encoded as (binary) **words**

⇒ Data sets = sets of words = **formal languages**,  
data transformations = **functions on words**

## Example A.1

*Java* = {all valid Java programs},

*Compiler* : *Java* → *Bytecode*

The atomic elements of words are called symbols (or letters).

## Definition A.2

An **alphabet** is a finite, non-empty set of symbols (“letters”).

$\Sigma, \Gamma, \dots$  denote alphabets

$a, b, \dots$  denote letters



The atomic elements of words are called symbols (or letters).

## Definition A.2

An **alphabet** is a finite, non-empty set of symbols (“letters”).

$\Sigma, \Gamma, \dots$  denote alphabets

$a, b, \dots$  denote letters

## Example A.3

- 1 Boolean alphabet  $\mathbb{B} := \{0, 1\}$

The atomic elements of words are called symbols (or letters).

## Definition A.2

An **alphabet** is a finite, non-empty set of symbols (“letters”).

$\Sigma, \Gamma, \dots$  denote alphabets

$a, b, \dots$  denote letters

## Example A.3

- 1 Boolean alphabet  $\mathbb{B} := \{0, 1\}$
- 2 Latin alphabet  $\Sigma_{\text{latin}} := \{a, b, c, \dots, z\}$

The atomic elements of words are called symbols (or letters).

## Definition A.2

An **alphabet** is a finite, non-empty set of symbols (“letters”).

$\Sigma, \Gamma, \dots$  denote alphabets

$a, b, \dots$  denote letters

## Example A.3

- 1 Boolean alphabet  $\mathbb{B} := \{0, 1\}$
- 2 Latin alphabet  $\Sigma_{\text{latin}} := \{a, b, c, \dots, z\}$
- 3 Keyboard alphabet  $\Sigma_{\text{key}}$

The atomic elements of words are called symbols (or letters).

## Definition A.2

An **alphabet** is a finite, non-empty set of symbols (“letters”).

$\Sigma, \Gamma, \dots$  denote alphabets

$a, b, \dots$  denote letters

## Example A.3

- 1 Boolean alphabet  $\mathbb{B} := \{0, 1\}$
- 2 Latin alphabet  $\Sigma_{\text{latin}} := \{a, b, c, \dots, z\}$
- 3 Keyboard alphabet  $\Sigma_{\text{key}}$
- 4 Morse alphabet  $\Sigma_{\text{morse}} := \{., -, \sqcup\}$

## Definition A.4

- A **word** is a finite sequence of letters from a given alphabet  $\Sigma$ .
- $\Sigma^*$  is the set of all words over  $\Sigma$ .

## Definition A.4

- A **word** is a finite sequence of letters from a given alphabet  $\Sigma$ .
- $\Sigma^*$  is the set of all words over  $\Sigma$ .
- $|w|$  denotes the **length** of a word  $w \in \Sigma^*$ , i.e.,  $|a_1 \dots a_n| := n$ .
- The **empty word** is denoted by  $\varepsilon$ , i.e.,  $|\varepsilon| = 0$ .

## Definition A.4

- A **word** is a finite sequence of letters from a given alphabet  $\Sigma$ .
- $\Sigma^*$  is the set of all words over  $\Sigma$ .
- $|w|$  denotes the **length** of a word  $w \in \Sigma^*$ , i.e.,  $|a_1 \dots a_n| := n$ .
- The **empty word** is denoted by  $\varepsilon$ , i.e.,  $|\varepsilon| = 0$ .
- The **concatenation** of two words  $v = a_1 \dots a_m$  ( $m \in \mathbb{N}$ ) and  $w = b_1 \dots b_n$  ( $n \in \mathbb{N}$ ) is the word

$$v \cdot w := a_1 \dots a_m b_1 \dots b_n$$

(often written as  $vw$ ).

- Thus:  $w \cdot \varepsilon = \varepsilon \cdot w = w$ .

## Definition A.4

- A **word** is a finite sequence of letters from a given alphabet  $\Sigma$ .
- $\Sigma^*$  is the set of all words over  $\Sigma$ .
- $|w|$  denotes the **length** of a word  $w \in \Sigma^*$ , i.e.,  $|a_1 \dots a_n| := n$ .
- The **empty word** is denoted by  $\varepsilon$ , i.e.,  $|\varepsilon| = 0$ .
- The **concatenation** of two words  $v = a_1 \dots a_m$  ( $m \in \mathbb{N}$ ) and  $w = b_1 \dots b_n$  ( $n \in \mathbb{N}$ ) is the word

$$v \cdot w := a_1 \dots a_m b_1 \dots b_n$$

(often written as  $vw$ ).

- Thus:  $w \cdot \varepsilon = \varepsilon \cdot w = w$ .
- A **prefix/suffix**  $v$  of a word  $w$  is an initial/trailing part of  $w$ , i.e.,  $w = vv' / w = v'v$  for some  $v' \in \Sigma^*$ .



## Definition A.4

- A **word** is a finite sequence of letters from a given alphabet  $\Sigma$ .
- $\Sigma^*$  is the set of all words over  $\Sigma$ .
- $|w|$  denotes the **length** of a word  $w \in \Sigma^*$ , i.e.,  $|a_1 \dots a_n| := n$ .
- The **empty word** is denoted by  $\varepsilon$ , i.e.,  $|\varepsilon| = 0$ .
- The **concatenation** of two words  $v = a_1 \dots a_m$  ( $m \in \mathbb{N}$ ) and  $w = b_1 \dots b_n$  ( $n \in \mathbb{N}$ ) is the word

$$v \cdot w := a_1 \dots a_m b_1 \dots b_n$$

(often written as  $vw$ ).

- Thus:  $w \cdot \varepsilon = \varepsilon \cdot w = w$ .
- A **prefix/suffix**  $v$  of a word  $w$  is an initial/trailing part of  $w$ , i.e.,  $w = vv' / w = v'v$  for some  $v' \in \Sigma^*$ .
- If  $w = a_1 \dots a_n$ , then  $w^R := a_n \dots a_1$ .

## Definition A.5

A set of words  $L \subseteq \Sigma^*$  is called a (formal) language over  $\Sigma$ .

## Definition A.5

A set of words  $L \subseteq \Sigma^*$  is called a (formal) language over  $\Sigma$ .

## Example A.6

- 1 over  $\mathbb{B} = \{0, 1\}$ : set of all bit strings containing 1101

## Definition A.5

A set of words  $L \subseteq \Sigma^*$  is called a (formal) language over  $\Sigma$ .

## Example A.6

- ① over  $\mathbb{B} = \{0, 1\}$ : set of all bit strings containing 1101
- ② over  $\Sigma = \{I, V, X, L, C, D, M\}$ : set of all valid roman numbers

## Definition A.5

A set of words  $L \subseteq \Sigma^*$  is called a (formal) language over  $\Sigma$ .

## Example A.6

- ① over  $\mathbb{B} = \{0, 1\}$ : set of all bit strings containing 1101
- ② over  $\Sigma = \{I, V, X, L, C, D, M\}$ : set of all valid roman numbers
- ③ over  $\Sigma_{\text{key}}$ : set of all valid Java programs

## Seen:

- Basic notions: alphabets, words
- Formal languages as sets of words

## Seen:

- Basic notions: alphabets, words
- Formal languages as sets of words

## Open:

- Description of computations on words?

- 1 Formal Languages
- 2 **Finite Automata**
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook



- 1 Formal Languages
- 2 **Finite Automata**
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

## Example A.7 (Pattern 1101)

- ➊ Read Boolean string bit-by-bit
- ➋ Test whether it contains 1101
- ➌ Idea: remember which (initial) part of 1101 has been recognized
- ➍ Five prefixes:  $\epsilon$ , 1, 11, 110, 1101
- ➎ Diagram: on the board

## Example A.7 (Pattern 1101)

- 1 Read Boolean string bit-by-bit
- 2 Test whether it contains 1101
- 3 Idea: remember which (initial) part of 1101 has been recognized
- 4 Five prefixes:  $\epsilon$ , 1, 11, 110, 1101
- 5 Diagram: on the board

What we used:

- finitely many (storage) states
- an initial state
- for every current state and every input symbol: a new state
- a successful state

## Definition A.8

A **deterministic finite automaton (DFA)** is of the form

$$\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where

- $Q$  is a finite set of **states**
- $\Sigma$  denotes the **input alphabet**
- $\delta : Q \times \Sigma \rightarrow Q$  is the **transition function**
- $q_0 \in Q$  is the **initial state**
- $F \subseteq Q$  is the set of **final** (or: **accepting**) **states**

## Example A.9

Pattern matching (Example A.7):

- $Q = \{q_0, \dots, q_4\}$
- $\Sigma = \mathbb{B} = \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q$  on the board
- $F = \{q_4\}$

## Example A.9

Pattern matching (Example A.7):

- $Q = \{q_0, \dots, q_4\}$
- $\Sigma = \mathbb{B} = \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q$  on the board
- $F = \{q_4\}$

## Graphical Representation of DFA:

- states  $\implies$  nodes
- $\delta(q, a) = q' \implies q \xrightarrow{a} q'$
- initial state: incoming edge without source state
- final state(s): double circle

## Definition A.10

Let  $\langle Q, \Sigma, \delta, q_0, F \rangle$  be a DFA. The **extension** of  $\delta : Q \times \Sigma \rightarrow Q$ ,  
$$\delta^* : Q \times \Sigma^* \rightarrow Q,$$

is defined by

$$\delta^*(q, w) := \text{state after reading } w \text{ starting from } q.$$

Formally:

$$\delta^*(q, w) := \begin{cases} q & \text{if } w = \varepsilon \\ \delta^*(\delta(q, a), v) & \text{if } w = av \end{cases}$$

Thus: if  $w = a_1 \dots a_n$  and  $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ , then  $\delta^*(q, w) = q_n$

## Definition A.10

Let  $\langle Q, \Sigma, \delta, q_0, F \rangle$  be a DFA. The **extension** of  $\delta : Q \times \Sigma \rightarrow Q$ ,  
$$\delta^* : Q \times \Sigma^* \rightarrow Q,$$

is defined by

$$\delta^*(q, w) := \text{state after reading } w \text{ starting from } q.$$

Formally:

$$\delta^*(q, w) := \begin{cases} q & \text{if } w = \varepsilon \\ \delta^*(\delta(q, a), v) & \text{if } w = av \end{cases}$$

Thus: if  $w = a_1 \dots a_n$  and  $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ , then  $\delta^*(q, w) = q_n$

## Example A.11

Pattern matching (Example A.9): on the board



## Definition A.12

- $\mathcal{A}$  **accepts**  $w \in \Sigma^*$  if  $\delta^*(q_0, w) \in F$ .
- The **language recognized** (or: **accepted**) by  $\mathcal{A}$  is

$$L(\mathcal{A}) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}.$$

- A language  $L \subseteq \Sigma^*$  is called **DFA-recognizable** if there exists some DFA  $\mathcal{A}$  such that  $L(\mathcal{A}) = L$ .
- Two DFA  $\mathcal{A}_1, \mathcal{A}_2$  are called **equivalent** if

$$L(\mathcal{A}_1) = L(\mathcal{A}_2).$$

## Example A.13

- ① The set of all bit strings containing **1101** is recognized by the automaton from Example A.9.

## Example A.13

- 1 The set of all bit strings containing 1101 is recognized by the automaton from Example A.9.
- 2 Two (equivalent) automata recognizing the language

$$\{w \in \mathbb{B}^* \mid w \text{ contains } 1\} :$$

on the board

## Example A.13

- 1 The set of all bit strings containing **1101** is recognized by the automaton from Example A.9.
- 2 Two (equivalent) automata recognizing the language

$$\{w \in \mathbb{B}^* \mid w \text{ contains } 1\} :$$

on the board

- 3 An automaton which recognizes

$$\{w \in \{0, \dots, 9\}^* \mid \text{value of } w \text{ divisible by } 3\}$$

Idea: test whether sum of digits is divisible by 3 – one state for each residue class (on the board)

## Seen:

- Deterministic finite automata as a model of simple sequential computations
- Recognizability of formal languages by automata

## Seen:

- Deterministic finite automata as a model of simple sequential computations
- Recognizability of formal languages by automata

## Open:

- Composition and transformation of automata?
- Which languages are recognizable, which are not (alternative characterization)?
- Language definition  $\mapsto$  automaton and vice versa?

- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

**Simplest case:** Boolean operations (complement, intersection, union)

## Question

Let  $\mathcal{A}_1, \mathcal{A}_2$  be two DFA with  $L(\mathcal{A}_1) = L_1$  and  $L(\mathcal{A}_2) = L_2$ .

Can we construct automata which recognize

- $\overline{L_1}$  ( $:= \Sigma^* \setminus L_1$ ),
- $L_1 \cap L_2$ , and
- $L_1 \cup L_2$ ?



## Theorem A.14

*If  $L \subseteq \Sigma^*$  is DFA-recognizable, then so is  $\bar{L}$ .*

## Theorem A.14

If  $L \subseteq \Sigma^*$  is DFA-recognizable, then so is  $\bar{L}$ .

## Proof.

Let  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  be a DFA such that  $L(\mathcal{A}) = L$ . Then:

$$w \in \bar{L} \iff w \notin L \iff \delta^*(q_0, w) \notin F \iff \delta^*(q_0, w) \in Q \setminus F.$$

Thus,  $\bar{L}$  is recognized by the DFA  $\langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$ . □

## Theorem A.14

If  $L \subseteq \Sigma^*$  is DFA-recognizable, then so is  $\bar{L}$ .

## Proof.

Let  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  be a DFA such that  $L(\mathcal{A}) = L$ . Then:

$$w \in \bar{L} \iff w \notin L \iff \delta^*(q_0, w) \notin F \iff \delta^*(q_0, w) \in Q \setminus F.$$

Thus,  $\bar{L}$  is recognized by the DFA  $\langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$ . □

## Example A.15

on the board

## Theorem A.16

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cap L_2$ .

## Theorem A.16

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cap L_2$ .

## Proof.

Let  $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$  be DFA such that  $L(\mathcal{A}_i) = L_i$  ( $i = 1, 2$ ). The new automaton  $\mathcal{A}$  has to accept  $w$  iff  $\mathcal{A}_1$  and  $\mathcal{A}_2$  accept  $w$

**Idea:** let  $\mathcal{A}_1$  and  $\mathcal{A}_2$  run in parallel

- use pairs of states  $(q_1, q_2) \in Q_1 \times Q_2$
- start with both components in initial state
- a transition updates both components independently
- for acceptance both components need to be in a final state



Proof (continued).

**Formally:** let the **product automaton**

$$\mathfrak{A} := \langle Q_1 \times Q_2, \Sigma, \delta, (q_0^1, q_0^2), F_1 \times F_2 \rangle$$

be defined by

$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \text{ for every } a \in \Sigma.$$

Proof (continued).

**Formally:** let the **product automaton**

$$\mathfrak{A} := \langle Q_1 \times Q_2, \Sigma, \delta, (q_0^1, q_0^2), F_1 \times F_2 \rangle$$

be defined by

$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \text{ for every } a \in \Sigma.$$

This definition yields

$$\delta^*((q_1, q_2), w) = (\delta_1^*(q_1, w), \delta_2^*(q_2, w)) \quad (*)$$

for every  $w \in \Sigma^*$ .

Proof (continued).

**Formally:** let the **product automaton**

$$\mathfrak{A} := \langle Q_1 \times Q_2, \Sigma, \delta, (q_0^1, q_0^2), F_1 \times F_2 \rangle$$

be defined by

$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \text{ for every } a \in \Sigma.$$

This definition yields

$$\delta^*((q_1, q_2), w) = (\delta_1^*(q_1, w), \delta_2^*(q_2, w)) \quad (*)$$

for every  $w \in \Sigma^*$ .

Thus we have:

$\mathfrak{A}$  accepts  $w$

$$\iff \delta^*((q_0^1, q_0^2), w) \in F_1 \times F_2$$

$$\stackrel{(*)}{\iff} (\delta_1^*(q_0^1, w), \delta_2^*(q_0^2, w)) \in F_1 \times F_2$$

$$\iff \delta_1^*(q_0^1, w) \in F_1 \text{ and } \delta_2^*(q_0^2, w) \in F_2$$

$$\iff \mathfrak{A}_1 \text{ accepts } w \text{ and } \mathfrak{A}_2 \text{ accepts } w$$





Example A.17

on the board

## Theorem A.18

*If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cup L_2$ .*

## Theorem A.18

*If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cup L_2$ .*

## Proof.

Let  $\mathfrak{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$  be DFA such that  $L(\mathfrak{A}_i) = L_i$  ( $i = 1, 2$ ). The new automaton  $\mathfrak{A}$  has to accept  $w$  iff  $\mathfrak{A}_1$  **or**  $\mathfrak{A}_2$  accepts  $w$ .

## Theorem A.18

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cup L_2$ .

### Proof.

Let  $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$  be DFA such that  $L(\mathcal{A}_i) = L_i$  ( $i = 1, 2$ ). The new automaton  $\mathcal{A}$  has to accept  $w$  iff  $\mathcal{A}_1$  or  $\mathcal{A}_2$  accepts  $w$ .

**Idea:** reuse product construction

Construct  $\mathcal{A}$  as before but choose as final states those pairs  $(q_1, q_2) \in Q_1 \times Q_2$  with  $q_1 \in F_1$  or  $q_2 \in F_2$ . Thus the set of final states is given by

$$F := (F_1 \times Q_2) \cup (Q_1 \times F_2).$$



## Definition A.19

The **concatenation** of two languages  $L_1, L_2 \subseteq \Sigma^*$  is given by

$$L_1 \cdot L_2 := \{v \cdot w \in \Sigma^* \mid v \in L_1, w \in L_2\}.$$

**Abbreviations:**  $w \cdot L := \{w\} \cdot L$ ,  $L \cdot w := L \cdot \{w\}$

## Definition A.19

The **concatenation** of two languages  $L_1, L_2 \subseteq \Sigma^*$  is given by

$$L_1 \cdot L_2 := \{v \cdot w \in \Sigma^* \mid v \in L_1, w \in L_2\}.$$

**Abbreviations:**  $w \cdot L := \{w\} \cdot L$ ,  $L \cdot w := L \cdot \{w\}$

## Example A.20

- ① If  $L_1 = \{101, 1\}$  and  $L_2 = \{011, 1\}$ , then  
 $L_1 \cdot L_2 = \{101011, 1011, 11\}.$

## Definition A.19

The **concatenation** of two languages  $L_1, L_2 \subseteq \Sigma^*$  is given by

$$L_1 \cdot L_2 := \{v \cdot w \in \Sigma^* \mid v \in L_1, w \in L_2\}.$$

**Abbreviations:**  $w \cdot L := \{w\} \cdot L$ ,  $L \cdot w := L \cdot \{w\}$

## Example A.20

- ① If  $L_1 = \{101, 1\}$  and  $L_2 = \{011, 1\}$ , then
$$L_1 \cdot L_2 = \{101011, 1011, 11\}.$$
- ② If  $L_1 = 00 \cdot \mathbb{B}^*$  and  $L_2 = 11 \cdot \mathbb{B}^*$ , then
$$L_1 \cdot L_2 = \{w \in \mathbb{B}^* \mid w \text{ has prefix } 00 \text{ and contains } 11\}.$$

## Conjecture

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cdot L_2$ .



# DFA-Recognizability of Concatenation

## Conjecture

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cdot L_2$ .

## Proof (attempt).

Let  $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$  be DFA such that  $L(\mathcal{A}_i) = L_i$  ( $i = 1, 2$ ). The new automaton  $\mathcal{A}$  has to accept  $w$  iff a prefix of  $w$  is recognized by  $\mathcal{A}_1$ , and if  $\mathcal{A}_2$  accepts the remaining suffix.

**Idea:** choose  $Q := Q_1 \cup Q_2$  where each  $q \in F_1$  is identified with  $q_0^2$

**But:** on the board



# DFA-Recognizability of Concatenation

## Conjecture

If  $L_1, L_2 \subseteq \Sigma^*$  are DFA-recognizable, then so is  $L_1 \cdot L_2$ .

## Proof (attempt).

Let  $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$  be DFA such that  $L(\mathcal{A}_i) = L_i$  ( $i = 1, 2$ ). The new automaton  $\mathcal{A}$  has to accept  $w$  iff a prefix of  $w$  is recognized by  $\mathcal{A}_1$ , and if  $\mathcal{A}_2$  accepts the remaining suffix.

**Idea:** choose  $Q := Q_1 \cup Q_2$  where each  $q \in F_1$  is identified with  $q_0^2$

**But:** on the board □

## Conclusion

Required: automata model where the successor state (for a given state and input symbol) is not unique

## Definition A.21

- The  **$n$ th power** of a language  $L \subseteq \Sigma^*$  is the  $n$ -fold concatenation of  $L$  with itself ( $n \in \mathbb{N}$ ):

$$L^n := \underbrace{L \cdot \dots \cdot L}_{n \text{ times}} = \{w_1 \dots w_n \mid \forall i \in \{1, \dots, n\} : w_i \in L\}.$$

Inductively:  $L^0 := \{\varepsilon\}$ ,  $L^{n+1} := L^n \cdot L$

- The **iteration** (or: **Kleene star**) of  $L$  is

$$L^* := \bigcup_{n \in \mathbb{N}} L^n = \{w_1 \dots w_n \mid n \in \mathbb{N}, \forall i \in \{1, \dots, n\} : w_i \in L\}.$$

## Definition A.21

- The  **$n$ th power** of a language  $L \subseteq \Sigma^*$  is the  $n$ -fold concatenation of  $L$  with itself ( $n \in \mathbb{N}$ ):

$$L^n := \underbrace{L \cdot \dots \cdot L}_{n \text{ times}} = \{w_1 \dots w_n \mid \forall i \in \{1, \dots, n\} : w_i \in L\}.$$

Inductively:  $L^0 := \{\varepsilon\}$ ,  $L^{n+1} := L^n \cdot L$

- The **iteration** (or: **Kleene star**) of  $L$  is

$$L^* := \bigcup_{n \in \mathbb{N}} L^n = \{w_1 \dots w_n \mid n \in \mathbb{N}, \forall i \in \{1, \dots, n\} : w_i \in L\}.$$

## Remarks:

- we always have  $\varepsilon \in L^*$  (since  $L^0 \subseteq L^*$  and  $L^0 = \{\varepsilon\}$ )
- $w \in L^*$  iff  $w = \varepsilon$  or if  $w$  can be decomposed into  $n \geq 1$  subwords  $v_1, \dots, v_n$  (i.e.,  $w = v_1 \cdot \dots \cdot v_n$ ) such that  $v_i \in L$  for every  $1 \leq i \leq n$
- again we would suspect that the iteration of a DFA-recognizable language is DFA-recognizable, but there is no simple (deterministic) construction

## Seen:

- Operations on languages:
  - complement
  - intersection
  - union
  - concatenation
  - iteration
- DFA constructions for:
  - complement
  - intersection
  - union

## Seen:

- Operations on languages:
  - complement
  - intersection
  - union
  - concatenation
  - iteration
- DFA constructions for:
  - complement
  - intersection
  - union

## Open:

- Automata model for (direct implementation of) concatenation and iteration?

- 1 Formal Languages
- 2 **Finite Automata**
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - **Nondeterministic Finite Automata**
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

## Idea:

- for a given state and a given input symbol, several transitions (or none at all) are possible
- an input word generally induces several state sequences (“runs”)
- the word is accepted if at least one accepting run exists



## Idea:

- for a given state and a given input symbol, several transitions (or none at all) are possible
- an input word generally induces several state sequences (“runs”)
- the word is accepted if at least one accepting run exists

## Advantages:

- simplifies representation of languages  
(example:  $\mathbb{B}^* \cdot 1101 \cdot \mathbb{B}^*$ ; on the board)
- yields direct constructions for concatenation and iteration of languages
- more adequate modeling of systems with nondeterministic behaviour  
(communication protocols, multi-agent systems, ...)

## Definition A.22

A **nondeterministic finite automaton (NFA)** is of the form

$$\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$$

where

- $Q$  is a finite set of **states**
- $\Sigma$  denotes the **input alphabet**
- $\Delta \subseteq Q \times \Sigma \times Q$  is the **transition relation**
- $q_0 \in Q$  is the **initial state**
- $F \subseteq Q$  is the set of **final states**

## Definition A.22

A **nondeterministic finite automaton (NFA)** is of the form

$$\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$$

where

- $Q$  is a finite set of **states**
- $\Sigma$  denotes the **input alphabet**
- $\Delta \subseteq Q \times \Sigma \times Q$  is the **transition relation**
- $q_0 \in Q$  is the **initial state**
- $F \subseteq Q$  is the set of **final states**

## Remarks:

- $(q, a, q') \in \Delta$  usually written as  $q \xrightarrow{a} q'$
- every DFA can be considered as an NFA  
 $((q, a, q') \in \Delta \iff \delta(q, a) = q')$

## Definition A.23

- Let  $w = a_1 \dots a_n \in \Sigma^*$ .
- A  $w$ -labeled  $\mathfrak{A}$ -run from  $q_1$  to  $q_2$  is a sequence

$$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots p_{n-1} \xrightarrow{a_n} p_n$$

such that  $p_0 = q_1$ ,  $p_n = q_2$ , and  $(p_{i-1}, a_i, p_i) \in \Delta$  for every  $1 \leq i \leq n$  (we also write:  $q_1 \xrightarrow{w} q_2$ ).

- $\mathfrak{A}$  **accepts**  $w$  if there is a  $w$ -labeled  $\mathfrak{A}$ -run from  $q_0$  to some  $q \in F$
- The **language recognized by**  $\mathfrak{A}$  is
$$L(\mathfrak{A}) := \{w \in \Sigma^* \mid \mathfrak{A} \text{ accepts } w\}.$$
- A language  $L \subseteq \Sigma^*$  is called **NFA-recognizable** if there exists a NFA  $\mathfrak{A}$  such that  $L(\mathfrak{A}) = L$ .
- Two NFA  $\mathfrak{A}_1, \mathfrak{A}_2$  are called **equivalent** if  $L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$ .

# Acceptance Test for NFA

## Algorithm A.24 (Acceptance Test for NFA)

Input:  $NFA \mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$ ,  $w \in \Sigma^*$

Question:  $w \in L(\mathcal{A})$ ?

Procedure: Computation of the *reachability set*

$$R_{\mathcal{A}}(w) := \{q \in Q \mid q_0 \xrightarrow{w} q\}$$

Iterative procedure for  $w = a_1 \dots a_n$ :

① let  $R_{\mathcal{A}}(\varepsilon) := \{q_0\}$

② for  $i := 1, \dots, n$ : let

$$R_{\mathcal{A}}(a_1 \dots a_i) := \{q \in Q \mid \exists p \in R_{\mathcal{A}}(a_1 \dots a_{i-1}) : p \xrightarrow{a_i} q\}$$

Output: “yes” if  $R_{\mathcal{A}}(w) \cap F \neq \emptyset$ , otherwise “no”

**Remark:** this algorithm solves the *word problem* for NFA

# Acceptance Test for NFA

## Algorithm A.24 (Acceptance Test for NFA)

Input:  $NFA \mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$ ,  $w \in \Sigma^*$

Question:  $w \in L(\mathcal{A})$ ?

Procedure: Computation of the **reachability set**

$$R_{\mathcal{A}}(w) := \{q \in Q \mid q_0 \xrightarrow{w} q\}$$

Iterative procedure for  $w = a_1 \dots a_n$ :

① let  $R_{\mathcal{A}}(\varepsilon) := \{q_0\}$

② for  $i := 1, \dots, n$ : let

$$R_{\mathcal{A}}(a_1 \dots a_i) := \{q \in Q \mid \exists p \in R_{\mathcal{A}}(a_1 \dots a_{i-1}) : p \xrightarrow{a_i} q\}$$

Output: “yes” if  $R_{\mathcal{A}}(w) \cap F \neq \emptyset$ , otherwise “no”

**Remark:** this algorithm solves the **word problem** for NFA

## Example A.25

on the board

Definition of NFA looks promising, but... (on the board)

Definition of NFA looks promising, but... (on the board)

**Solution:** admit empty word  $\epsilon$  as transition label



## Definition A.26

A **nondeterministic finite automaton with  $\epsilon$ -transitions ( $\epsilon$ -NFA)** is of the form  $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  where

- $Q$  is a finite set of **states**
- $\Sigma$  denotes the **input alphabet**
- $\Delta \subseteq Q \times \Sigma_\epsilon \times Q$  is the **transition relation** where  $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$
- $q_0 \in Q$  is the **initial state**
- $F \subseteq Q$  is the set of **final states**

## Remarks:

- every NFA is an  $\epsilon$ -NFA
- definitions of runs and acceptance: in analogy to NFA

## Definition A.26

A **nondeterministic finite automaton with  $\epsilon$ -transitions ( $\epsilon$ -NFA)** is of the form  $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  where

- $Q$  is a finite set of **states**
- $\Sigma$  denotes the **input alphabet**
- $\Delta \subseteq Q \times \Sigma_{\epsilon} \times Q$  is the **transition relation** where  $\Sigma_{\epsilon} := \Sigma \cup \{\epsilon\}$
- $q_0 \in Q$  is the **initial state**
- $F \subseteq Q$  is the set of **final states**

## Remarks:

- every NFA is an  $\epsilon$ -NFA
- definitions of runs and acceptance: in analogy to NFA

## Example A.27

on the board

## Theorem A.28

*If  $L_1, L_2 \subseteq \Sigma^*$  are  $\varepsilon$ -NFA-recognizable, then so is  $L_1 \cdot L_2$ .*

## Theorem A.28

If  $L_1, L_2 \subseteq \Sigma^*$  are  $\varepsilon$ -NFA-recognizable, then so is  $L_1 \cdot L_2$ .

Proof (idea).

on the board □

## Theorem A.28

*If  $L_1, L_2 \subseteq \Sigma^*$  are  $\varepsilon$ -NFA-recognizable, then so is  $L_1 \cdot L_2$ .*

Proof (idea).

on the board ☐

## Theorem A.29

*If  $L \subseteq \Sigma^*$  is  $\varepsilon$ -NFA-recognizable, then so is  $L^*$ .*

Proof.

see Theorem A.47 ☐

Syntax diagrams (without recursive calls) can be interpreted as  $\epsilon$ -NFA

## Example A.30

decimal numbers (on the board)

# Types of Finite Automata

- ① DFA (Definition A.8)
- ② NFA (Definition A.22)
- ③  $\epsilon$ -NFA (Definition A.26)

- ① DFA (Definition A.8)
- ② NFA (Definition A.22)
- ③  $\epsilon$ -NFA (Definition A.26)

From the definitions we immediately obtain:

## Corollary A.31

- ① *Every DFA-recognizable language is NFA-recognizable.*
- ② *Every NFA-recognizable language is  $\epsilon$ -NFA-recognizable.*



- 1 DFA (Definition A.8)
- 2 NFA (Definition A.22)
- 3  $\epsilon$ -NFA (Definition A.26)

From the definitions we immediately obtain:

## Corollary A.31

- 1 *Every DFA-recognizable language is NFA-recognizable.*
- 2 *Every NFA-recognizable language is  $\epsilon$ -NFA-recognizable.*

**Goal:** establish reverse inclusions

## Theorem A.32

*Every NFA can be transformed into an equivalent DFA.*

## Theorem A.32

*Every NFA can be transformed into an equivalent DFA.*

## Proof.

**Idea:** let the DFA operate on **sets of states** (“powerset construction”)

- Initial state of DFA  $:= \{\text{initial state of NFA}\}$
- $P \xrightarrow{a} P'$  in DFA iff there exist  $q \in P, q' \in P'$  such that  $q \xrightarrow{a} q'$  in NFA
- $P$  final state in DFA iff it contains some final state of NFA



## Proof (continued).

Let  $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  be a NFA.

**Powerset construction** of  $\mathfrak{A}' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ :

- $Q' := 2^Q := \{P \mid P \subseteq Q\}$
- $\delta' : Q' \times \Sigma \rightarrow Q'$  with
$$q \in \delta'(P, a) \iff \text{there exists } p \in P \text{ such that } (p, a, q) \in \Delta$$
- $q'_0 := \{q_0\}$
- $F' := \{P \subseteq Q \mid P \cap F \neq \emptyset\}$

This yields

$$q_0 \xrightarrow{w} q \text{ in } \mathfrak{A} \iff q \in \delta'^*(\{q_0\}, w) \text{ in } \mathfrak{A}'$$

and thus

$$\mathfrak{A} \text{ accepts } w \iff \mathfrak{A}' \text{ accepts } w$$



## Proof (continued).

Let  $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  be a NFA.

**Powerset construction** of  $\mathfrak{A}' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ :

- $Q' := 2^Q := \{P \mid P \subseteq Q\}$
- $\delta' : Q' \times \Sigma \rightarrow Q'$  with
$$q \in \delta'(P, a) \iff \text{there exists } p \in P \text{ such that } (p, a, q) \in \Delta$$
- $q'_0 := \{q_0\}$
- $F' := \{P \subseteq Q \mid P \cap F \neq \emptyset\}$

This yields

$$q_0 \xrightarrow{w} q \text{ in } \mathfrak{A} \iff q \in \delta'^*(\{q_0\}, w) \text{ in } \mathfrak{A}'$$

and thus

$$\mathfrak{A} \text{ accepts } w \iff \mathfrak{A}' \text{ accepts } w$$



## Example A.33

on the board

## Theorem A.34

*Every  $\varepsilon$ -NFA can be transformed into an equivalent NFA.*

## Theorem A.34

*Every  $\varepsilon$ -NFA can be transformed into an equivalent NFA.*

## Proof (idea).

Let  $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  be a  $\varepsilon$ -NFA. We construct the NFA  $\mathcal{A}'$  by eliminating all  $\varepsilon$ -transitions, adding appropriate direct transitions: if  $p \xrightarrow{\varepsilon}^* q$ ,  $q \xrightarrow{a} q'$ , and  $q' \xrightarrow{\varepsilon}^* r$  in  $\mathcal{A}$ , then  $p \xrightarrow{a} r$  in  $\mathcal{A}'$ . Moreover  $F' := F \cup \{q_0\}$  if  $q_0 \xrightarrow{\varepsilon}^* q \in F$  in  $\mathcal{A}$ , and  $F' := F$  otherwise. □

# From $\varepsilon$ -NFA to NFA

## Theorem A.34

*Every  $\varepsilon$ -NFA can be transformed into an equivalent NFA.*

## Proof (idea).

Let  $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  be a  $\varepsilon$ -NFA. We construct the NFA  $\mathcal{A}'$  by eliminating all  $\varepsilon$ -transitions, adding appropriate direct transitions: if  $p \xrightarrow{\varepsilon}^* q$ ,  $q \xrightarrow{a} q'$ , and  $q' \xrightarrow{\varepsilon}^* r$  in  $\mathcal{A}$ , then  $p \xrightarrow{a} r$  in  $\mathcal{A}'$ . Moreover  $F' := F \cup \{q_0\}$  if  $q_0 \xrightarrow{\varepsilon}^* q \in F$  in  $\mathcal{A}$ , and  $F' := F$  otherwise. □

## Example A.35

on the board



# From $\varepsilon$ -NFA to NFA

## Theorem A.34

*Every  $\varepsilon$ -NFA can be transformed into an equivalent NFA.*

## Proof (idea).

Let  $\mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$  be a  $\varepsilon$ -NFA. We construct the NFA  $\mathcal{A}'$  by eliminating all  $\varepsilon$ -transitions, adding appropriate direct transitions: if  $p \xrightarrow{\varepsilon^*} q$ ,  $q \xrightarrow{a} q'$ , and  $q' \xrightarrow{\varepsilon^*} r$  in  $\mathcal{A}$ , then  $p \xrightarrow{a} r$  in  $\mathcal{A}'$ . Moreover  $F' := F \cup \{q_0\}$  if  $q_0 \xrightarrow{\varepsilon^*} q \in F$  in  $\mathcal{A}$ , and  $F' := F$  otherwise.  $\square$

## Example A.35

on the board

## Corollary A.36

*All types of finite automata recognize the same class of languages.*

## Seen:

- Definition of  $\epsilon$ -NFA
- Determinization of ( $\epsilon$ -)NFA

## Seen:

- Definition of  $\epsilon$ -NFA
- Determinization of ( $\epsilon$ -)NFA

## Open:

- More decidability results

- 1 Formal Languages
- 2 **Finite Automata**
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - **More Decidability Results**
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

## Definition A.37

The **word problem for DFA** is specified as follows:

Given a DFA  $\mathfrak{A}$  and a word  $w \in \Sigma^*$ , decide whether

$$w \in L(\mathfrak{A}).$$

## Definition A.37

The **word problem for DFA** is specified as follows:

Given a DFA  $\mathfrak{A}$  and a word  $w \in \Sigma^*$ , decide whether

$$w \in L(\mathfrak{A}).$$

As we have seen (Def. A.10, Alg. A.24, Thm. A.34):

## Theorem A.38

*The word problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

# The Emptiness Problem

## Definition A.39

The **emptiness problem for DFA** is specified as follows:

Given a DFA  $\mathcal{A}$ , decide whether  $L(\mathcal{A}) = \emptyset$ .

# The Emptiness Problem

## Definition A.39

The **emptiness problem for DFA** is specified as follows:

Given a DFA  $\mathcal{A}$ , decide whether  $L(\mathcal{A}) = \emptyset$ .

## Theorem A.40

*The emptiness problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

It holds that  $L(\mathcal{A}) \neq \emptyset$  iff in  $\mathcal{A}$  some final state is reachable from the initial state (simple graph-theoretic problem). □



# The Emptiness Problem

## Definition A.39

The **emptiness problem for DFA** is specified as follows:

Given a DFA  $\mathcal{A}$ , decide whether  $L(\mathcal{A}) = \emptyset$ .

## Theorem A.40

*The emptiness problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

It holds that  $L(\mathcal{A}) \neq \emptyset$  iff in  $\mathcal{A}$  some final state is reachable from the initial state (simple graph-theoretic problem). □

**Remark:** important result for formal verification  
(unreachability of bad [= final] states)

# The Equivalence Problem

## Definition A.41

The **equivalence problem for DFA** is specified as follows:

Given two DFA  $\mathfrak{A}_1, \mathfrak{A}_2$ , decide whether

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2).$$

# The Equivalence Problem

## Definition A.41

The **equivalence problem for DFA** is specified as follows:

Given two DFA  $\mathfrak{A}_1, \mathfrak{A}_2$ , decide whether

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2).$$

## Theorem A.42

*The equivalence problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$$

# The Equivalence Problem

## Definition A.41

The **equivalence problem for DFA** is specified as follows:

Given two DFA  $\mathfrak{A}_1, \mathfrak{A}_2$ , decide whether

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2).$$

## Theorem A.42

*The equivalence problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$$

$$\iff L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1)$$

# The Equivalence Problem

## Definition A.41

The **equivalence problem for DFA** is specified as follows:

Given two DFA  $\mathfrak{A}_1, \mathfrak{A}_2$ , decide whether

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2).$$

## Theorem A.42

*The equivalence problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$$

$$\iff L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1)$$

$$\iff (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset$$

# The Equivalence Problem

## Definition A.41

The **equivalence problem for DFA** is specified as follows:

Given two DFA  $\mathfrak{A}_1, \mathfrak{A}_2$ , decide whether

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2).$$

## Theorem A.42

*The equivalence problem for DFA (NFA,  $\varepsilon$ -NFA) is **decidable**.*

## Proof.

$$L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$$

$$\iff L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1)$$

$$\iff (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset$$

$$\iff (L(\mathfrak{A}_1) \cap \underbrace{\overline{L(\mathfrak{A}_2)}}_{\text{DFA-recognizable (Thm. A.14)}}) \cup (L(\mathfrak{A}_2) \cap \underbrace{\overline{L(\mathfrak{A}_1)}}_{\text{DFA-recognizable (Thm. A.14)}}) = \emptyset$$

$$\underbrace{\hspace{10em}}_{\text{DFA-recognizable (Thm. A.16)}} \quad \underbrace{\hspace{10em}}_{\text{DFA-recognizable (Thm. A.16)}}$$

$$\underbrace{\hspace{10em}}_{\text{DFA-recognizable (Thm. A.18)}}$$

$$\underbrace{\hspace{10em}}_{\text{decidable (Thm. A.40)}}$$

□

## Seen:

- Decidability of word problem
- Decidability of emptiness problem
- Decidability of equivalence problem

## Seen:

- Decidability of word problem
- Decidability of emptiness problem
- Decidability of equivalence problem

## Open:

- Non-algorithmic description of languages



- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

## Example A.43

Consider the set of all words over  $\Sigma := \{a, b\}$  which

- 1 start with one or three  $a$  symbols
- 2 continue with a (potentially empty) sequence of blocks, each containing at least one  $b$  and exactly two  $a$ 's
- 3 conclude with a (potentially empty) sequence of  $b$ 's

Corresponding **regular expression**:

$$(a + aaa)(\underbrace{bb^*ab^*ab^*}_{b \text{ before } a's} + \underbrace{b^*abb^*ab^*}_{b \text{ between } a's} + \underbrace{b^*ab^*abb^*}_{b \text{ after } a's})^*b^*$$

## Definition A.44

The set of **regular expressions** over  $\Sigma$  is inductively defined by:

- $\emptyset$  and  $\varepsilon$  are regular expressions
- every  $a \in \Sigma$  is a regular expression
- if  $\alpha$  and  $\beta$  are regular expressions, then so are
  - $\alpha + \beta$
  - $\alpha \cdot \beta$
  - $\alpha^*$

## Definition A.44

The set of **regular expressions** over  $\Sigma$  is inductively defined by:

- $\emptyset$  and  $\varepsilon$  are regular expressions
- every  $a \in \Sigma$  is a regular expression
- if  $\alpha$  and  $\beta$  are regular expressions, then so are
  - $\alpha + \beta$
  - $\alpha \cdot \beta$
  - $\alpha^*$

## Notation:

- $\cdot$  can be omitted
- $*$  binds stronger than  $\cdot$ ,  $\cdot$  binds stronger than  $+$
- $\alpha^+$  abbreviates  $\alpha \cdot \alpha^*$

## Definition A.45

Every regular expression  $\alpha$  defines a language  $L(\alpha)$ :

$$L(\emptyset) := \emptyset$$

$$L(\varepsilon) := \{\varepsilon\}$$

$$L(a) := \{a\}$$

$$L(\alpha + \beta) := L(\alpha) \cup L(\beta)$$

$$L(\alpha \cdot \beta) := L(\alpha) \cdot L(\beta)$$

$$L(\alpha^*) := (L(\alpha))^*$$

## Definition A.45

Every regular expression  $\alpha$  defines a language  $L(\alpha)$ :

$$L(\emptyset) := \emptyset$$

$$L(\varepsilon) := \{\varepsilon\}$$

$$L(a) := \{a\}$$

$$L(\alpha + \beta) := L(\alpha) \cup L(\beta)$$

$$L(\alpha \cdot \beta) := L(\alpha) \cdot L(\beta)$$

$$L(\alpha^*) := (L(\alpha))^*$$

A language  $L$  is called **regular** if it is definable by a regular expression, i.e., if  $L = L(\alpha)$  for some regular expression  $\alpha$ .

## Example A.46

①  $\{aa\}$  is regular since

$$L(a \cdot a) = L(a) \cdot L(a) = \{a\} \cdot \{a\} = \{aa\}$$

## Example A.46

- ①  $\{aa\}$  is regular since

$$L(a \cdot a) = L(a) \cdot L(a) = \{a\} \cdot \{a\} = \{aa\}$$

- ②  $\{a, b\}^*$  is regular since

$$L((a + b)^*) = (L(a + b))^* = (L(a) \cup L(b))^* = (\{a\} \cup \{b\})^* = \{a, b\}^*$$



## Example A.46

- ①  $\{aa\}$  is regular since

$$L(a \cdot a) = L(a) \cdot L(a) = \{a\} \cdot \{a\} = \{aa\}$$

- ②  $\{a, b\}^*$  is regular since

$$L((a + b)^*) = (L(a + b))^* = (L(a) \cup L(b))^* = (\{a\} \cup \{b\})^* = \{a, b\}^*$$

- ③ The set of all words over  $\{a, b\}$  containing  $abb$  is regular since

$$L((a + b)^* \cdot a \cdot b \cdot b \cdot (a + b)^*) = \{a, b\}^* \cdot \{abb\} \cdot \{a, b\}^*$$

## Theorem A.47 (Kleene's Theorem)

*To each regular expression there corresponds an  $\varepsilon$ -NFA, and vice versa.*

## Theorem A.47 (Kleene's Theorem)

*To each regular expression there corresponds an  $\varepsilon$ -NFA, and vice versa.*

### Proof.

- $\Rightarrow$  using induction over the given regular expression  $\alpha$ , we construct an  $\varepsilon$ -NFA  $\mathfrak{A}_\alpha$
- with exactly one final state  $q_f$
  - without transitions into the initial state
  - without transitions leaving the final state
- (on the board)
- $\Leftarrow$  by solving a regular equation system (details omitted)



## Corollary A.48

*The following properties are equivalent:*

- $L$  is regular
- $L$  is DFA-recognizable
- $L$  is NFA-recognizable
- $L$  is  $\varepsilon$ -NFA-recognizable

## Algorithm A.49 (Pattern Matching)

Input: *regular expression  $\alpha$  and  $w \in \Sigma^*$*

Question: *does  $w$  contain some  $v \in L(\alpha)$ ?*

Procedure:

- 1 let  $\beta := (a_1 + \dots + a_n)^* \cdot \alpha$  (for  $\Sigma = \{a_1, \dots, a_n\}$ )
- 2 determine  $\varepsilon$ -NFA  $\mathfrak{A}_\beta$  for  $\beta$
- 3 eliminate  $\varepsilon$ -transitions
- 4 apply powerset construction to obtain DFA  $\mathfrak{A}$
- 5 let  $\mathfrak{A}$  run on  $w$

Output: “yes” if  $\mathfrak{A}$  passes through some final state, otherwise “no”

**Remark:** in UNIX/LINUX implemented by `grep` and `lex`

# Regular Expressions in UNIX (grep, flex, ...)

Syntax	Meaning
printable character	this character
\n, \t, \123, etc.	newline, tab, octal representation, etc.
.	any character except \n
[ <i>Chars</i> ]	one of <i>Chars</i> ; ranges possible ("0-9")
[^ <i>Chars</i> ]	none of <i>Chars</i>
\\, \., \[, etc.	\, ., [, etc.
" <i>Text</i> "	<i>Text</i> without interpretation of ., [, \, etc.
^ $\alpha$	$\alpha$ at beginning of line
$\alpha$ \$	$\alpha$ at end of line
$\alpha$ ?	zero or one $\alpha$
$\alpha$ *	zero or more $\alpha$
$\alpha$ +	one or more $\alpha$
$\alpha\{n, m\}$	between $n$ and $m$ times $\alpha$ (" , $m$ " optional)
( $\alpha$ )	$\alpha$
$\alpha_1\alpha_2$	concatenation
$\alpha_1 \alpha_2$	alternative

## Seen:

- Definition of regular expressions
- Equivalence of regular and DFA-recognizable languages

- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook



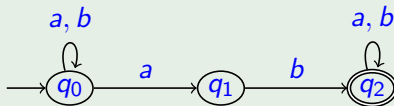
Goal: space-efficient implementation of regular languages

Given: DFA  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$

Wanted: DFA  $\mathcal{A}_{min} = \langle Q', \Sigma, \delta', q'_0, F' \rangle$  such that  $L(\mathcal{A}_{min}) = L(\mathcal{A})$  and  $|Q'|$  **minimal**

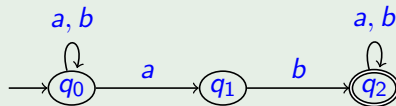
## Example A.50

NFA for accepting  $(a + b)^* ab(a + b)^*$ :

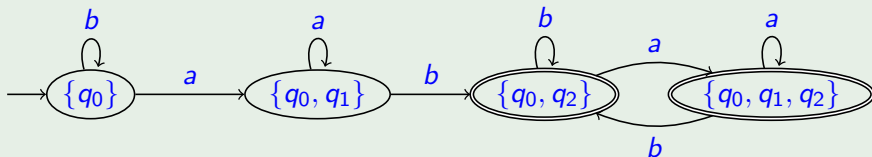


## Example A.50

NFA for accepting  $(a + b)^*ab(a + b)^*$ :

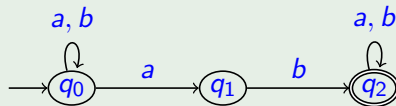


Powerset construction yields DFA  $\mathcal{A}$ :

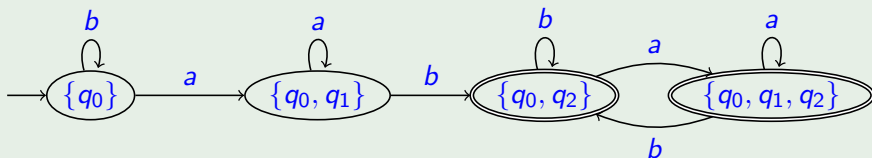


## Example A.50

NFA for accepting  $(a + b)^*ab(a + b)^*$ :



Powerset construction yields DFA  $\mathcal{A}$ :

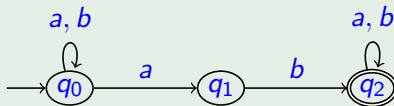


**Observation:**  $\{q_0, q_2\}$  and  $\{q_0, q_1, q_2\}$  are **equivalent**

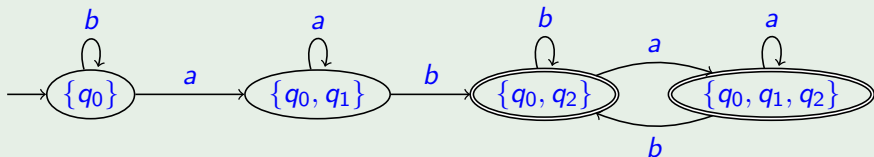
# State Equivalence

## Example A.50

NFA for accepting  $(a + b)^*ab(a + b)^*$ :



Powerset construction yields DFA  $\mathcal{A}$ :



**Observation:**  $\{q_0, q_2\}$  and  $\{q_0, q_1, q_2\}$  are **equivalent**

## Definition A.51

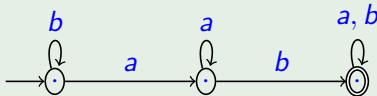
Given DFA  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ , states  $p, q \in Q$  are **equivalent** if

$$\forall w \in \Sigma^* : \delta^*(p, w) \in F \iff \delta^*(q, w) \in F.$$

Minimization: **merging** of equivalent states

Example A.52 (cf. Example A.50)

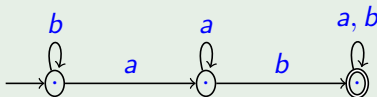
DFA after state merging:



Minimization: **merging** of equivalent states

Example A.52 (cf. Example A.50)

DFA after state merging:



Problem: **identification** of equivalent states

Approach: iterative computation of inequivalent states by refinement

Corollary A.53

$p, q \in Q$  are **inequivalent** if there exists  $w \in \Sigma^*$  such that

$$\delta^*(p, w) \in F \text{ and } \delta^*(q, w) \notin F$$

(or vice versa, i.e.,  $p$  and  $q$  can be distinguished by  $w$ )

## Lemma A.54

*Inductive characterization of state inequivalence:*

- $w = \varepsilon: p \in F, q \notin F \implies p, q$  inequivalent (by  $\varepsilon$ )
- $w = av: p', q'$  inequivalent (by  $v$ ),  $p \xrightarrow{a} p', q \xrightarrow{a} q' \implies p, q$  inequivalent (by  $w$ )



# Computing State (In-)Equivalence

## Lemma A.54

*Inductive characterization of state inequivalence:*

- $w = \varepsilon$ :  $p \in F, q \notin F \implies p, q$  inequivalent (by  $\varepsilon$ )
- $w = av$ :  $p', q'$  inequivalent (by  $v$ ),  $p \xrightarrow{a} p', q \xrightarrow{a} q' \implies p, q$  inequivalent (by  $w$ )

## Algorithm A.55 (State Equivalence for DFA)

Input: DFA  $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$

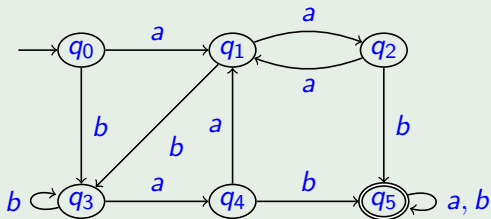
Procedure: Computation of “*equivalence matrix*” over  $Q \times Q$

- 1 mark every pair  $(p, q)$  with  $p \in F, q \notin F$  by  $\varepsilon$
- 2 for every unmarked pair  $(p, q)$  and every  $a \in \Sigma$ :  
if  $(\delta(p, a), \delta(q, a))$  marked by  $v$ , then mark  $(p, q)$  by  $av$
- 3 repeat until no change

Output: all equivalent (= unmarked) pairs of states

## Example A.56

Given DFA:

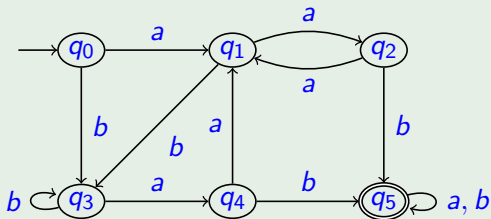


Equivalence matrix: on the board

# Minimization Example

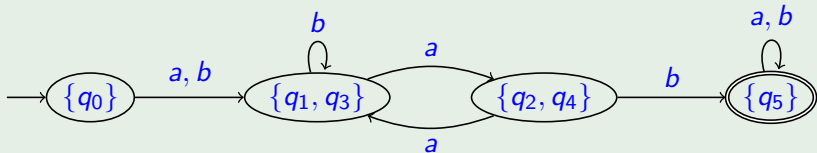
## Example A.56

Given DFA:



Equivalence matrix: on the board

Resulting minimal DFA:



## Theorem A.57

For every DFA  $\mathfrak{A}$ ,

$$L(\mathfrak{A}) = L(\mathfrak{A}_{min})$$

## Theorem A.57

For every DFA  $\mathfrak{A}$ ,

$$L(\mathfrak{A}) = L(\mathfrak{A}_{min})$$

**Remark:** the minimal DFA is **unique**, in the following sense:

$$\forall \text{DFA } \mathfrak{A}, \mathfrak{B} : L(\mathfrak{A}) = L(\mathfrak{B}) \implies \mathfrak{A}_{min} \approx \mathfrak{B}_{min}$$

where  $\approx$  refers to automata isomorphism (= identity up to naming of states)

- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook

**Observation:** a language  $L$  is DFA-recognizable (and thus regular) if the membership of a word  $w$  can be tested by **symbol-wise reading** of  $w$ , using a **bounded memory**

**Observation:** a language  $L$  is DFA-recognizable (and thus regular) if the membership of a word  $w$  can be tested by **symbol-wise reading** of  $w$ , using a **bounded memory**

**Conjecture:** languages of the form  $\{a^n b^n \mid n \in \mathbb{N}\}$  are not regular since the test for membership requires the capability of comparing the number of  $a$  symbols to the number of  $b$  symbols (which can grow arbitrarily large)



# The Pumping Lemma I

## Theorem A.58 (Pumping Lemma for Regular Languages)

If  $L$  is regular, then there exists  $n \geq 1$  (called *pumping index*) such that any  $w \in L$  with  $|w| \geq n$  can be decomposed as  $w = xyz$  where

- $y \neq \varepsilon$  and
- for every  $i \geq 0$ ,  $xy^iz \in L$

## Proof (idea).

Let  $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  be a DFA such that  $L(\mathfrak{A}) = L$ . Choose  $n := |Q|$ , and let  $w \in L$ .

Then:

- $w = a_1 \dots a_k$  with  $k \geq n$
- $\implies$  the accepting run visits  $k + 1 \geq n + 1$  states:  
 $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} q_k$
- $\implies$  some state in  $Q$  occurs (at least) twice:  
there exist  $1 \leq i < j \leq k$  such that  $q_i = q_j$

Choose  $y := a_{i+1} \dots a_j$  to be the substring which is read between the two visits of  $q$ . Clearly,  $y \neq \varepsilon$ . Moreover the cycle can be omitted or repeated such that  $xz \in L$ ,  $xyz \in L$ ,  $xy^2z \in L$ , ... □

# The Pumping Lemma II

**Remark:** Pumping Lemma states a **necessary condition** for regularity  
 $\Rightarrow$  can only be used to show the **non-regularity** of a language

# The Pumping Lemma II

**Remark:** Pumping Lemma states a **necessary condition** for regularity  
 $\Rightarrow$  can only be used to show the **non-regularity** of a language

## Example A.59

- ①  $L := \{a^k b^k \mid k \in \mathbb{N}\}$  is not regular. Proof by contradiction:  
Assume that  $L$  is regular, and let  $n$  be a pumping index. Consider  $w := a^n b^n$ . Since  $|w| \geq n$ , it can be decomposed as  $w = xyz$  with  $y \neq \varepsilon$ . The following cases are possible:
- $y \in L(a^+)$ : then  $xy^2z \notin L$  (more  $as$  than  $bs$ )
  - $y \in L(b^+)$ : then  $xy^2z \notin L$  (less  $as$  than  $bs$ )
  - $y \in L(a^+b^+)$ : then  $xy^2z \notin L$  ( $a$  follows  $b$ )

# The Pumping Lemma II

**Remark:** Pumping Lemma states a **necessary condition** for regularity  
 $\implies$  can only be used to show the **non-regularity** of a language

## Example A.59

- ①  $L := \{a^k b^k \mid k \in \mathbb{N}\}$  is not regular. Proof by contradiction:  
Assume that  $L$  is regular, and let  $n$  be a pumping index. Consider  $w := a^n b^n$ . Since  $|w| \geq n$ , it can be decomposed as  $w = xyz$  with  $y \neq \varepsilon$ . The following cases are possible:
  - $y \in L(a^+)$ : then  $xy^2z \notin L$  (more  $a$ s than  $b$ s)
  - $y \in L(b^+)$ : then  $xy^2z \notin L$  (less  $a$ s than  $b$ s)
  - $y \in L(a^+b^+)$ : then  $xy^2z \notin L$  ( $a$  follows  $b$ )
- ② Similarly: the set of all arithmetic expressions is not regular

# The Pumping Lemma II

**Remark:** Pumping Lemma states a **necessary condition** for regularity  
 $\implies$  can only be used to show the **non-regularity** of a language

## Example A.59

- ①  $L := \{a^k b^k \mid k \in \mathbb{N}\}$  is not regular. Proof by contradiction:  
Assume that  $L$  is regular, and let  $n$  be a pumping index. Consider  $w := a^n b^n$ . Since  $|w| \geq n$ , it can be decomposed as  $w = xyz$  with  $y \neq \varepsilon$ . The following cases are possible:
  - $y \in L(a^+)$ : then  $xy^2z \notin L$  (more  $a$ s than  $b$ s)
  - $y \in L(b^+)$ : then  $xy^2z \notin L$  (less  $a$ s than  $b$ s)
  - $y \in L(a^+b^+)$ : then  $xy^2z \notin L$  ( $a$  follows  $b$ )
- ② Similarly: the set of all arithmetic expressions is not regular

## Conclusion

Finite automata are **too weak** for defining the syntax of programming languages (consider  $a = "("$ ,  $b = ")"$ )!

## Seen:

- Necessary condition for regularity of languages
- Counterexamples

## Seen:

- Necessary condition for regularity of languages
- Counterexamples

## Open:

- More expressive formalisms for describing languages?

- 1 Formal Languages
- 2 Finite Automata
  - Deterministic Finite Automata
  - Operations on Languages and Automata
  - Nondeterministic Finite Automata
  - More Decidability Results
- 3 Regular Expressions
- 4 Minimization of DFA
- 5 The Pumping Lemma
- 6 Outlook



- More **language operations** (homomorphisms, ...)
- Construction of **scanners** for compilers