Foundations of Informatics: a Bridging Course

Week 3: Formal Languages and Processes
Part A: Regular Languages
b-it Bonn; 02–06 March 2020

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-19-20/foi/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Organisation

- Schedule:
  - lecture 10:00–12:30 (Mon-Thu)
    - including short break
    - somewhat longer?
  - exercises 14:00–17:00 (Mon-Thu)
    - including short break
    - somewhat shorter?
- First exam on Tuesday, 31 March 2020, 10:00–13:00, at b-it Bonn (room bitmax)
- Second exam on Tuesday, 26 May 2020, 10:00–13:00, at RWTH Aachen University (CS Department, building E3, room 9U10)
- Please ask questions!

Overview of Week 3

1. Regular Languages
   - Formal Languages
   - Finite Automata
   - Regular Expressions
   - Minimisation of Finite Automata
2. Context-Free Languages
   - Context-Free Grammars and Languages
   - Context-Free vs. Regular Languages
   - The Word Problem for Context-Free Languages
   - The Emptiness Problem for Context-Free Languages
   - Closure Properties of Context-Free Languages
   - Pushdown Automata

Literature

- 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)

Outline of Part A

Words and Languages

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

$\Rightarrow$ Data sets = sets of words = formal languages,
data transformations = functions on words

Words and Languages

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

$\Rightarrow$ Data sets = sets of words = formal languages,
data transformations = functions on words

Example A.1

- $\mathit{Java} = \{\text{all valid Java programs}\}$
- $\mathit{Compiler} : \mathit{Java} \to \mathit{Bytecode}$

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Alphabets

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

Definition A.2

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

- $\Sigma, \Gamma, \ldots$ denote alphabets
- $a, b, \ldots$ denote letters

Alphabets

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

Definition A.2

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

- $\Sigma, \Gamma, \ldots$ denote alphabets
- $a, b, \ldots$ denote letters

Example A.3

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

Alphabets

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

Definition A.2

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

- $\Sigma, \Gamma, \ldots$ denote alphabets
- $a, b, \ldots$ denote letters

Example A.3

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

Alphabets

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

Definition A.2

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

- $\Sigma, \Gamma, \ldots$ denote alphabets
- $a, b, \ldots$ denote letters

Example A.3

1. Boolean alphabet $\mathbb{B} := \{0, 1\}$
2. Latin alphabet $\Sigma_{\mathrm{latin}} := \{a, b, c, \ldots, z\}$
3. Keyboard alphabet Σ_{key}

Alphabets

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

Definition A.2

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

- $\Sigma, \Gamma, \ldots$ denote alphabets
- $a, b, \ldots$ denote letters

Example A.3

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

Words

Definition A.4

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

Words

Definition A.4

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

Words

Definition A.4

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

Words

Definition A.4

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

Words

Definition A.4

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

Formal Languages I

Definition A.5

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

Formal Languages I

Definition A.5

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

Example A.6

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

Formal Languages I

Definition A.5

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

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

Formal Languages I

Definition A.5

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

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 Σ_{key}: set of all valid Java programs

Formal Languages II

Seen:

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

Formal Languages II

Seen:

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

Next:

- Description of computations on words

Outline of Part A

Outline of Part A

Example: Pattern Matching

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 recognised
4. Five prefixes: ε, 1, 11, 110, 1101
5. Diagram: on the board

Example: Pattern Matching

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 recognised
4. Five prefixes: ε, 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

Deterministic Finite Automata I

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
- Σ denotes the input alphabet
- $\delta : Q \times \Sigma \to Q$ 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):

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

Deterministic Finite Automata II

Example A.9

Pattern matching (Example A.7):

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

Graphical Representation of DFA:

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

Acceptance by DFA I

Definition A.10

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

$$\delta^* : Q \times \Sigma^* \to Q,$$

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

Acceptance by DFA I

Definition A.10

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

$$\delta^* : Q \times \Sigma^* \to Q,$$

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

Acceptance by DFA II

Definition A.12

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

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

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

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

Acceptance by DFA III

Example A.13

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

Acceptance by DFA III

Example A.13

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

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

on the board

Acceptance by DFA III

Example A.13

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

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

   on the board
3. An automaton which recognises

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

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

Deterministic Finite Automata

Seen:

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

Deterministic Finite Automata

Seen:

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

Next:

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

Outline of Part A

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Operations on Languages

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

Question

Let $\mathfrak{A}_1$, $\mathfrak{A}_2$ be two DFA with $L(\mathfrak{A}_1) = L_1$ and $L(\mathfrak{A}_2) = L_2$.
Can we construct automata which recognise

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

Language Complement

Theorem A.14

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

Language Complement

Theorem A.14

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

Proof.

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

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

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

Language Complement

Theorem A.14

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

Proof.

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

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

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

Example A.15

on the board

Language Intersection I

Theorem A.16

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Language Intersection I

Theorem A.16

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

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 $\mathfrak{A}_1$ and $\mathfrak{A}_2$ accept w

Idea: let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ 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

□

Language Intersection II

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

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

Language Intersection II

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 (for every $w \in \Sigma^*$):

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

Language Intersection II

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 (for every $w \in \Sigma^*$):

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

Thus: $\mathfrak{A}$ 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

Language Union

Theorem A.18

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

Language Union

Theorem A.18

If $L_1, L_2 \subseteq \Sigma^*$ are DFA-recognisable, 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.

Language Union

Theorem A.18

If $L_1, L_2 \subseteq \Sigma^*$ are DFA-recognisable, 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.

Idea: reuse product construction
Construct $\mathfrak{A}$ 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

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

□

Language Concatenation

Definition A.19

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

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

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

Language Concatenation

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
$$L_1 \cdot L_2 = \{101011, 1011, 11\}.$$

Language Concatenation

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
$$L_1 \cdot L_2 = \{w \in \mathbb{B}^* \mid w \text{ has prefix 00 and contains 11}\}.$$

DFA-Recognisability of Concatenation

Conjecture

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

DFA-Recognisability of Concatenation

Conjecture

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

Proof (attempt).

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 a prefix of w is recognised by $\mathfrak{A}_1$, and if $\mathfrak{A}_2$ 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-Recognisability of Concatenation

Conjecture

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

Proof (attempt).

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 a prefix of w is recognised by $\mathfrak{A}_1$, and if $\mathfrak{A}_2$ 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

Language Iteration

Definition A.21

- The *n*th power of a language $L \subseteq \Sigma^*$ is the *n*-fold concatenation of L with itself ($n \in \mathbb{N}$):
$$L^n := \underbrace{L \cdot \ldots \cdot L}_{n \text{ times}} = \{w_1 \ldots w_n \mid \forall i \in \{1, \ldots, n\} : w_i \in L\}.$$
Inductively: $L^0 := \{\varepsilon\}$, $L^{n+1} := L^n \cdot L$
- The iteration (or: Kleene star) of L is
$$L^* := \bigcup_{n \in \mathbb{N}} L^n = \{w_1 \ldots w_n \mid n \in \mathbb{N}, \forall i \in \{1, \ldots, n\} : w_i \in L\}.$$

Language Iteration

Definition A.21

- The *n*th power of a language $L \subseteq \Sigma^*$ is the n-fold concatenation of L with itself ($n \in \mathbb{N}$):
$$L^n := \underbrace{L \cdot \ldots \cdot L}_{n \text{ times}} = \{w_1 \ldots w_n \mid \forall i \in \{1, \ldots, n\} : w_i \in L\}.$$
Inductively: $L^0 := \{\varepsilon\}$, $L^{n+1} := L^n \cdot L$
- The iteration (or: Kleene star) of L is
$$L^* := \bigcup_{n \in \mathbb{N}} L^n = \{w_1 \ldots w_n \mid n \in \mathbb{N}, \forall i \in \{1, \ldots, n\} : w_i \in L\}.$$

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 $v_1, \ldots, v_n$ (i.e., $w = v_1 \cdot \ldots \cdot v_n$) such that $v_i \in L$ for every $1 \leq i \leq n$
- again we would suspect that the iteration of a DFA-recognisable language is DFA-recognisable, but there is no simple (deterministic) construction

Operations on Languages and Automata

Seen:

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

Operations on Languages and Automata

Seen:

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

Next:

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

Outline of Part A

Nondeterministic Finite Automata I

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Nondeterministic Finite Automata I

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 modelling of systems with nondeterministic behaviour
  - communication protocols, multi-agent systems, ...

Nondeterministic Finite Automata II

Definition A.22

A nondeterministic finite automaton (NFA) is of the form

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

where

- Q is a finite set of states
- Σ 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

Nondeterministic Finite Automata II

Definition A.22

A nondeterministic finite automaton (NFA) is of the form

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

where

- Q is a finite set of states
- Σ 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 ($(q, a, q') \in \Delta \iff \delta(q, a) = q'$)

Acceptance by NFA

Definition A.23

- Let $w = a_1 \dots a_n \in \Sigma^*$.
- A w-labelled $\mathfrak{A}$-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-labelled $\mathfrak{A}$-run from q_0 to some $q \in F$
- The language recognised by $\mathfrak{A}$ is

$$L(\mathfrak{A}) := \{w \in \Sigma^* \mid \mathfrak{A} \text{ accepts } w\}.$$

- A language $L \subseteq \Sigma^*$ is called NFA-recognisable 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 $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$, $w \in \Sigma^*$

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

Procedure: Computation of the *reachability set*

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

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

1. *let* $R_{\mathfrak{A}}(\varepsilon) := \{q_0\}$
2. *for* $i := 1, \ldots, n$: *let*
   $R_{\mathfrak{A}}(a_1 \ldots a_i) := \{q \in Q \mid \exists p \in R_{\mathfrak{A}}(a_1 \ldots a_{i-1}) : p \xrightarrow{a_i} q\}$

Output: *"yes" if* $R_{\mathfrak{A}}(w) \cap F \neq \emptyset$, *otherwise "no"*

Remark: this algorithm solves the word problem for NFA

Acceptance Test for NFA

Algorithm A.24 (Acceptance Test for NFA)

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

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

Procedure: Computation of the *reachability set*

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

Iterative procedure for $w = a_1 \ldots a_n$*:*

1. *let* $R_{\mathfrak{A}}(\varepsilon) := \{q_0\}$
2. *for* $i := 1, \ldots, n$*: let*
   $R_{\mathfrak{A}}(a_1 \ldots a_i) := \{q \in Q \mid \exists p \in R_{\mathfrak{A}}(a_1 \ldots a_{i-1}) : p \xrightarrow{a_i} q\}$

Output: *"yes" if* $R_{\mathfrak{A}}(w) \cap F \neq \emptyset$*, otherwise "no"*

Remark: this algorithm solves the word problem for NFA

Example A.25

on the board

NFA-Recognisability of Concatenation

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

NFA-Recognisability of Concatenation

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

Solution: admit empty word ε as transition label

ε-NFA

Definition A.26

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

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

Remarks:

- every NFA is an ε-NFA
- definitions of runs and acceptance: in analogy to NFA

Definition A.26

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

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

Remarks:

- every NFA is an ε-NFA
- definitions of runs and acceptance: in analogy to NFA

Example A.27

on the board

Concatenation and Iteration via ε-NFA

Theorem A.28

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

Concatenation and Iteration via ε-NFA

Theorem A.28

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

Proof (idea).

on the board $\square$

Concatenation and Iteration via ε-NFA

Theorem A.28

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

Proof (idea).

on the board □

Theorem A.29

If $L \subseteq \Sigma^*$ is ε-NFA-recognisable, then so is L^*.

Proof.

see Theorem A.46 □

Types of Finite Automata

1. DFA (Definition A.8)
2. NFA (Definition A.22)
3. ε-NFA (Definition A.26)

Types of Finite Automata

1. DFA (Definition A.8)
2. NFA (Definition A.22)
3. ε-NFA (Definition A.26)

From the definitions we immediately obtain:

Corollary A.30

1. *Every DFA-recognisable language is NFA-recognisable.*
2. *Every NFA-recognisable language is ε-NFA-recognisable.*

Types of Finite Automata

1. DFA (Definition A.8)
2. NFA (Definition A.22)
3. ε-NFA (Definition A.26)

From the definitions we immediately obtain:

Corollary A.30

1. *Every DFA-recognisable language is NFA-recognisable.*
2. *Every NFA-recognisable language is ε-NFA-recognisable.*

Goal: establish reverse inclusions

From NFA to DFA I

Theorem A.31

Every NFA can be transformed into an equivalent DFA.

From NFA to DFA I

Theorem A.31

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 := {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

□

From NFA to DFA II

Proof (continued).

Let $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$ a NFA. Powerset construction of $\mathfrak{A}' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$:

- $Q' := 2^Q := \{P \mid P \subseteq Q\}$
- $\delta' : Q' \times \Sigma \to Q'$ with $q \in \delta'(P, a) \iff$ there exists $p \in P$ 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

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

□

From NFA to DFA II

Proof (continued).

Let $\mathfrak{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle$ a NFA. Powerset construction of $\mathfrak{A}' = \langle Q', \Sigma, \delta', q_0', F' \rangle$:

- $Q' := 2^Q := \{P \mid P \subseteq Q\}$
- $\delta' : Q' \times \Sigma \to Q'$ with $q \in \delta'(P, a) \iff$ there exists $p \in P$ 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.32

on the board

From ε-NFA to NFA

Theorem A.33

Every ε-NFA can be transformed into an equivalent NFA.

From ε-NFA to NFA

Theorem A.33

Every ε-NFA can be transformed into an equivalent NFA.

Proof (idea).

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

From ε-NFA to NFA

Theorem A.33

Every ε-NFA can be transformed into an equivalent NFA.

Proof (idea).

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

Example A.34

on the board

From ε-NFA to NFA

Theorem A.33

Every ε-NFA can be transformed into an equivalent NFA.

Proof (idea).

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

Example A.34

on the board

Corollary A.35

All types of finite automata recognise the same class of languages.

Nondeterministic Finite Automata

Seen:

- Definition of ε-NFA
- Determinisation of (ε-)NFA

Seen:

- Definition of ε-NFA
- Determinisation of (ε-)NFA

Next:

- More decidablity results

Outline of Part A

The Word Problem Revisited

Definition A.36

The word problem for DFA is specified as follows:

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

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

The Word Problem Revisited

Definition A.36

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.33):

Theorem A.37

The word problem for DFA (NFA, ε-NFA) is decidable.

The Emptiness Problem

Definition A.38

The emptiness problem for DFA is specified as follows:

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

The Emptiness Problem

Definition A.38

The emptiness problem for DFA is specified as follows:

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

Theorem A.39

The emptiness problem for DFA (NFA, ε-NFA) is decidable.

Proof.

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

The Emptiness Problem

Definition A.38

The emptiness problem for DFA is specified as follows:

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

Theorem A.39

The emptiness problem for DFA (NFA, ε-NFA) is decidable.

Proof.

It holds that $L(\mathfrak{A}) \neq \emptyset$ iff in $\mathfrak{A}$ 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.40

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.40

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.41

The equivalence problem for DFA (NFA, ε-NFA) is decidable.

Proof.

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

The Equivalence Problem

Definition A.40

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.41

The equivalence problem for DFA (NFA, ε-NFA) is decidable.

Proof.

$$\begin{aligned} & L(\mathfrak{A}_1) = L(\mathfrak{A}_2) \\ \iff \quad & L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1) \end{aligned}$$

The Equivalence Problem

Definition A.40

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.41

The equivalence problem for DFA (NFA, ε-NFA) is decidable.

Proof.

$$\begin{aligned} & L(\mathfrak{A}_1) = L(\mathfrak{A}_2) \\ \iff & L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1) \\ \iff & (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset \end{aligned}$$

The Equivalence Problem

Definition A.40

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.41

The equivalence problem for DFA (NFA, ε-NFA) is decidable.

Proof.

$$
\begin{aligned}
& L(\mathfrak{A}_1) = L(\mathfrak{A}_2) \\
\iff\; & L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2) \text{ and } L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1) \\
\iff\; & (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset \\
\iff\; & \underbrace{\underbrace{\underbrace{(L(\mathfrak{A}_1) \cap \underbrace{\overline{L(\mathfrak{A}_2)}}_{\text{DFA-recognisable (Thm. A.14)}})}_{\text{DFA-recognisable (Thm. A.16)}} \cup \underbrace{(L(\mathfrak{A}_2) \cap \underbrace{\overline{L(\mathfrak{A}_1)}}_{\text{DFA-recognisable (Thm. A.14)}})}_{\text{DFA-recognisable (Thm. A.16)}}}_{\text{DFA-recognisable (Thm. A.18)}} = \emptyset}_{\text{decidable (Thm. A.39)}}
\end{aligned}
$$

□

Seen:

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

Finite Automata

Seen:

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

Next:

- Non-algorithmic description of languages

Outline of Part A

Outline of Part A

An Example

Example A.42

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

An Example

Example A.42

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:

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

Syntax of Regular Expressions

Definition A.43

The set of regular expressions over Σ is inductively defined by:

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

Syntax of Regular Expressions

Definition A.43

The set of regular expressions over Σ is inductively defined by:

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

Notation:

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

Semantics of Regular Expressions

Definition A.44

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

$$
\begin{aligned}
L(\emptyset) &:= \emptyset \\
L(\varepsilon) &:= \{\varepsilon\} \\
L(a) &:= \{a\} \\
L(\alpha + \beta) &:= L(\alpha) \cup L(\beta) \\
L(\alpha \cdot \beta) &:= L(\alpha) \cdot L(\beta) \\
L(\alpha^*) &:= (L(\alpha))^*
\end{aligned}
$$

Semantics of Regular Expressions

Definition A.44

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

$$
\begin{aligned}
L(\emptyset) &:= \emptyset \\
L(\varepsilon) &:= \{\varepsilon\} \\
L(a) &:= \{a\} \\
L(\alpha + \beta) &:= L(\alpha) \cup L(\beta) \\
L(\alpha \cdot \beta) &:= L(\alpha) \cdot L(\beta) \\
L(\alpha^*) &:= (L(\alpha))^*
\end{aligned}
$$

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

Example A.45

1. $\{aa\}$ is regular since

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

Example A.45

1. $\{aa\}$ is regular since

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

2. $\{a, b\}^*$ is regular since

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

Regular Languages

Example A.45

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

Outline of Part A

Theorem A.46 (Kleene's Theorem)

To each regular expression there corresponds an ε-NFA, and vice versa.

Theorem A.46 (Kleene's Theorem)

To each regular expression there corresponds an ε-NFA, and vice versa.

Proof.

$\Rightarrow$: by induction over the given regular expression α, we construct an ε-NFA $\mathfrak{A}_\alpha$ with exactly one final state q_f and without transitions into the initial/leaving the final state:

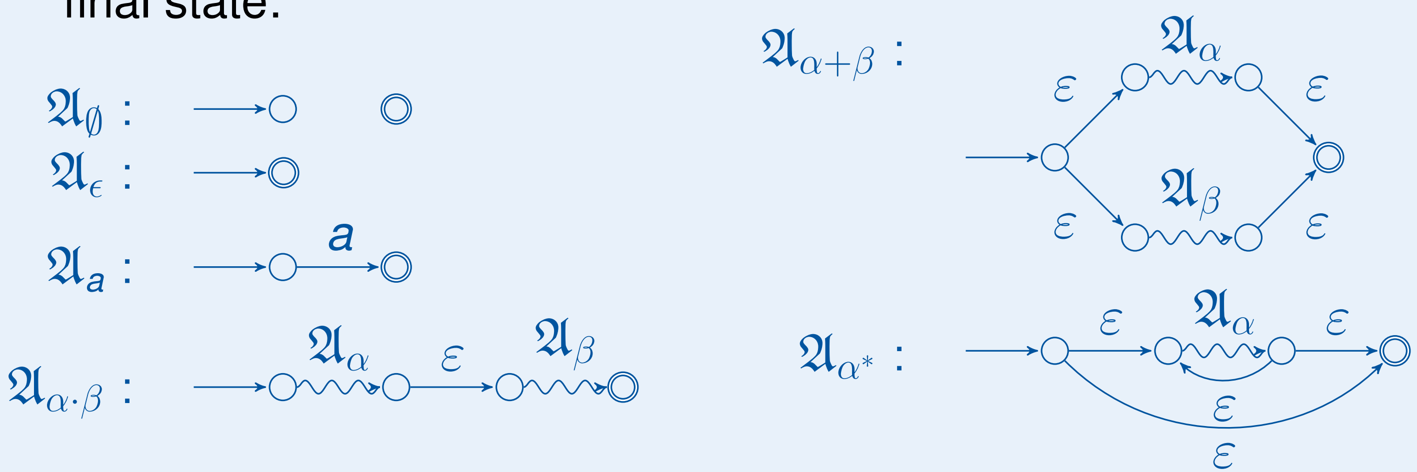
Regular Languages and Finite Automata I

Theorem A.46 (Kleene's Theorem)

To each regular expression there corresponds an ε-NFA, and vice versa.

Proof.

$\Rightarrow$: by induction over the given regular expression α, we construct an ε-NFA $\mathfrak{A}_\alpha$ with exactly one final state q_f and without transitions into the initial/leaving the final state:

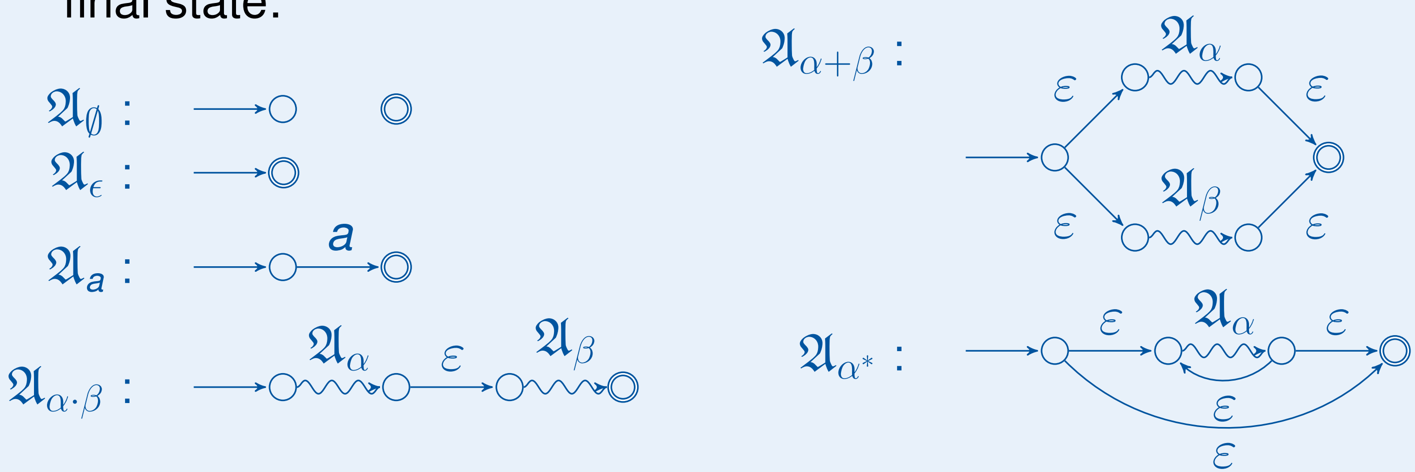


Theorem A.46 (Kleene's Theorem)

To each regular expression there corresponds an ε-NFA, and vice versa.

Proof.

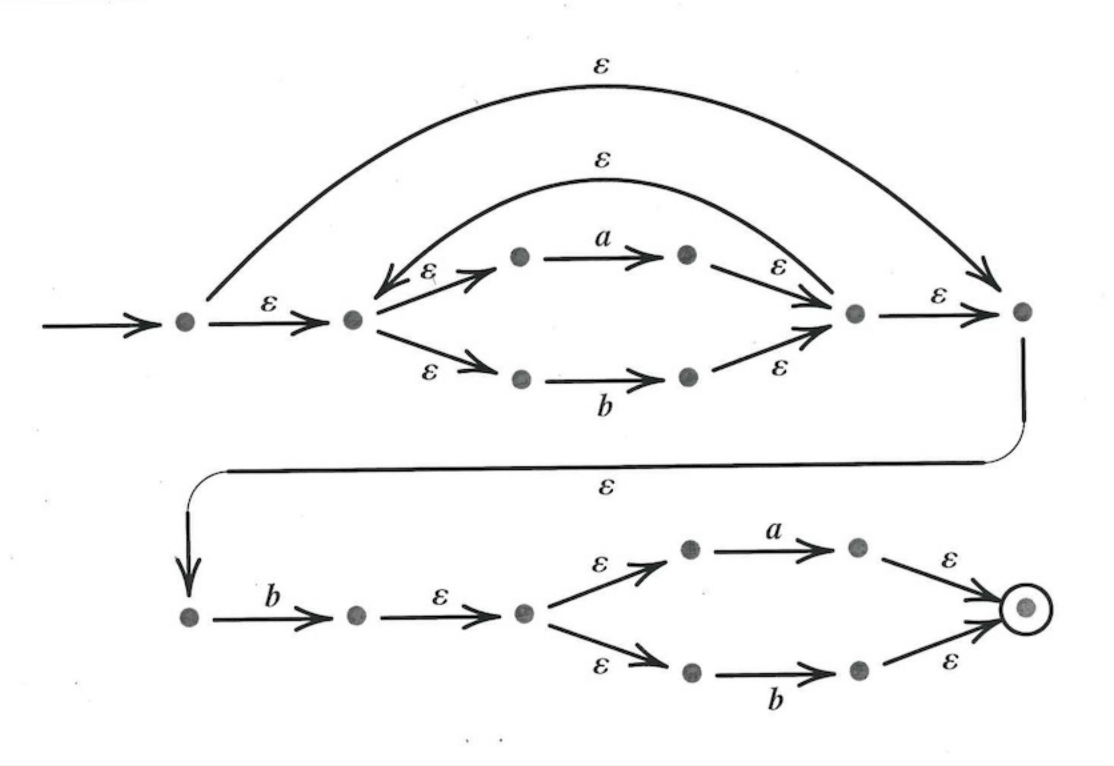
$\Rightarrow$: by induction over the given regular expression α, we construct an ε-NFA $\mathfrak{A}_\alpha$ with exactly one final state q_f and without transitions into the initial/leaving the final state:



$\Leftarrow$: by solving a regular equation system (details omitted) □

Example A.47

For the regular expression $(a + b)^* \cdot b \cdot (a + b)$, we obtain the following ε-NFA:



Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Regular Languages and Finite Automata III

Corollary A.48

The following properties are equivalent:

- L *is regular*
- L *is DFA-recognisable*
- L *is NFA-recognisable*
- L *is* ε*-NFA-recognisable*

Implementation of Pattern Matching

Algorithm A.49 (Pattern Matching)

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

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

Procedure:
1. let $\beta := (a_1 + \ldots + a_n)^* \cdot \alpha$ (for $\Sigma = \{a_1, \ldots, a_n\}$)
2. determine ε-NFA $\mathfrak{A}_\beta$ for β
3. eliminate ε-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`
`[`*Chars*`]`	one of *Chars*; ranges possible (“`0-9`”)
`[^`*Chars*`]`	none of *Chars*
`\\`, `\.`, `\[`, etc.	`\`, `.`, `[`, etc.
`"`*Text*`"`	*Text* without interpretation of `.`, `[`, `\`, etc.
`^`α	α at beginning of line
α`$`	α at end of line
α`?`	zero or one α
α`*`	zero or more α
α`+`	one or more α
α`{`*n*, *m*`}`	between *n* and *m* times α (“, *m*” optional)
`(`α`)`	α
$\alpha_1\alpha_2$	concatenation
$\alpha_1 \mid \alpha_2$	alternative

Regular Expressions

Seen:

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Regular Expressions

Seen:

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

Next:

- “Optimisation” of finite automata

Outline of Part A

Motivation

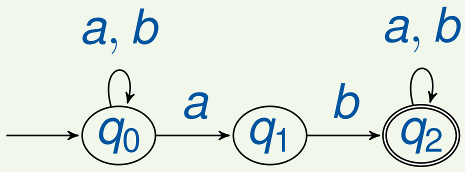
Goal: space-efficient implementation of regular languages

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

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

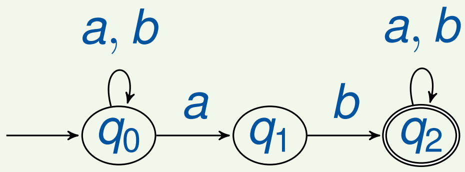
Example A.50

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

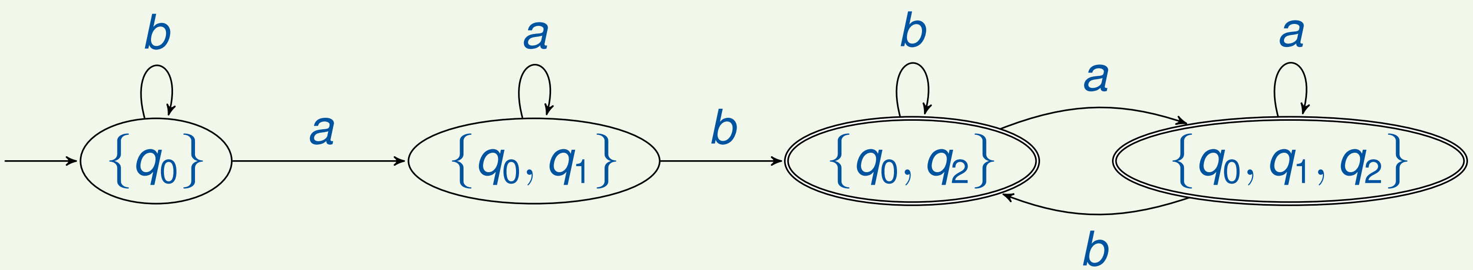


Example A.50

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

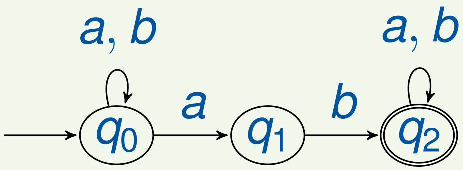


Powerset construction yields DFA $\mathfrak{A}$:

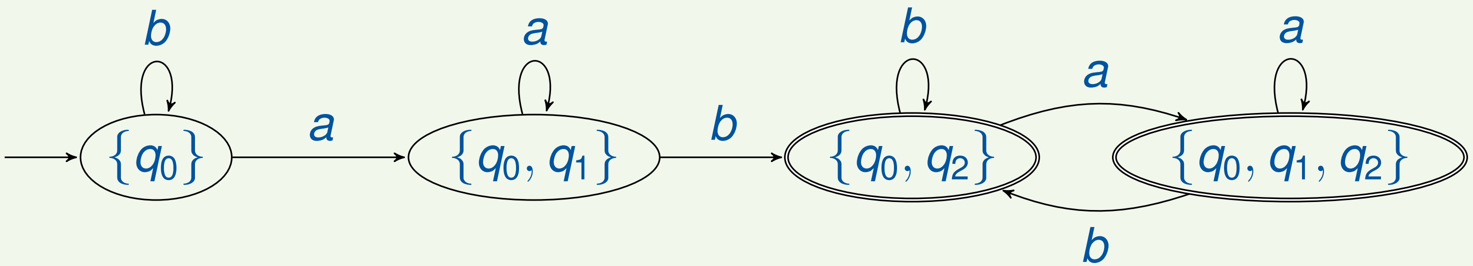


Example A.50

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



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

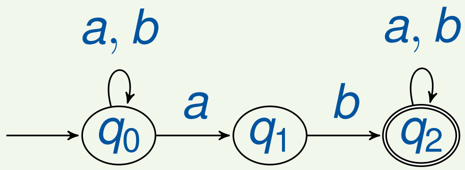


Observation: $\{q_0, q_2\}$ and $\{q_0, q_1, q_2\}$ are equivalent

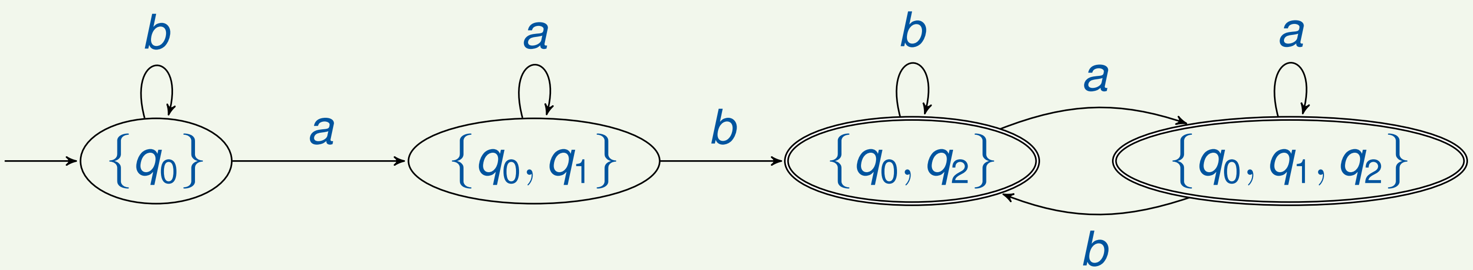
State Equivalence

Example A.50

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



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



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

Definition A.51

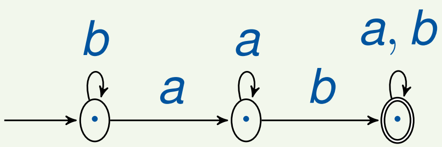
Given DFA $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, states $p, q \in Q$ are equivalent if

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

Minimisation: merging of equivalent states

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

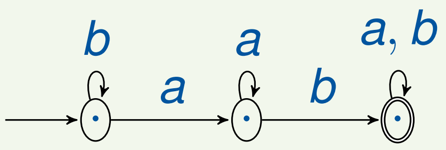
DFA after state merging:



Minimisation: 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)

Computing State (In-)Equivalence

Lemma A.54

Inductive characterisation of state inequivalence:

- $w = \varepsilon$: $p \in F, q \notin F \implies p, q$ *inequivalent (by* ε*)*
- $w = av$: 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 characterisation of state inequivalence:

- $w = \varepsilon$: $p \in F, q \notin F \implies p, q$ *inequivalent (by* ε*)*
- $w = av$: 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 $Q \times Q$

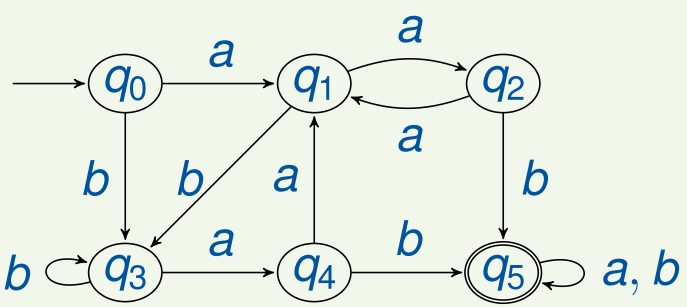
1. *mark every pair* (p, q) *with* $p \in F, q \notin F$ *by* ε
2. *for every unmarked pair* (p, q) *and every* $a \in \Sigma$*:*
   if $(\delta(p, a), \delta(q, a))$ *marked by* v*, then mark* (p, q) *by* av
3. *repeat until no change*

Output: all equivalent (= unmarked) pairs of states

Minimisation Example

Example A.56

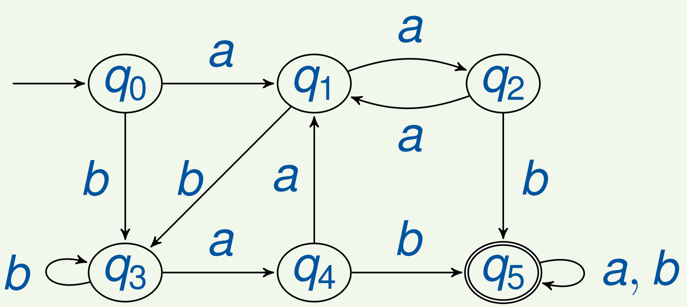
Given DFA:



Minimisation Example

Example A.56

Given DFA:



Equivalence matrix:

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X					
q_1	X	X				
q_2	X	X	X			
q_3	X	X	X	X		
q_4	X	X	X	X	X	
q_5	X	X	X	X	X	X

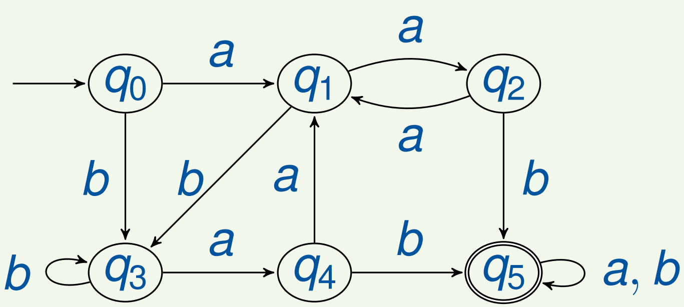
Remarks:

- entries (q_i, q_i) not needed as always equivalent
- entries (q_i, q_j) with $i > j$ not needed due to symmetry

Minimisation Example

Example A.56

Given DFA:



Equivalence matrix:

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X					ε
q_1	X	X				ε
q_2	X	X	X			ε
q_3	X	X	X	X		ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

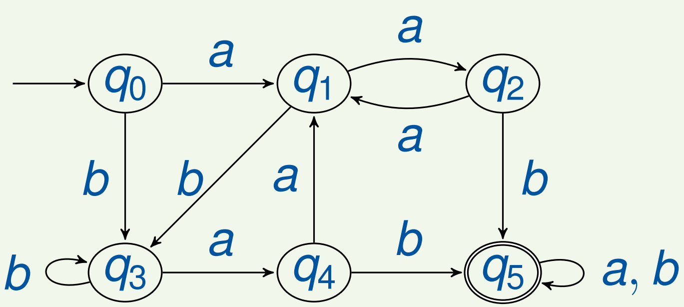
Algorithm A.55:

1. Mark every pair (p, q) with $p \in F, q \notin F$ by ε

Minimisation Example

Example A.56

Given DFA:



Equivalence matrix:

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X					ε
q_1	X	X				ε
q_2	X	X	X			ε
q_3	X	X	X	X		ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

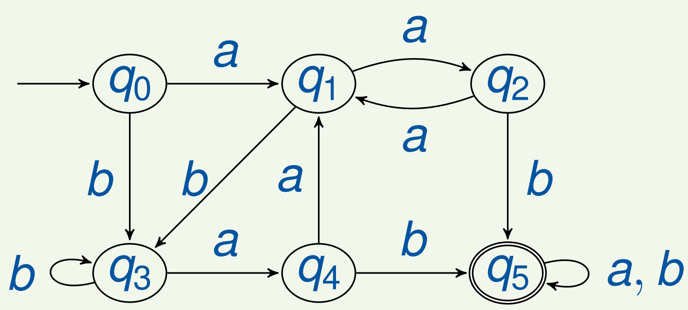
Algorithm A.55:

2. If $(\delta(p, a), \delta(q, a))$ marked by ε, then mark (p, q) by a (not applicable)

Minimisation Example

Example A.56

Given DFA:



Equivalence matrix:

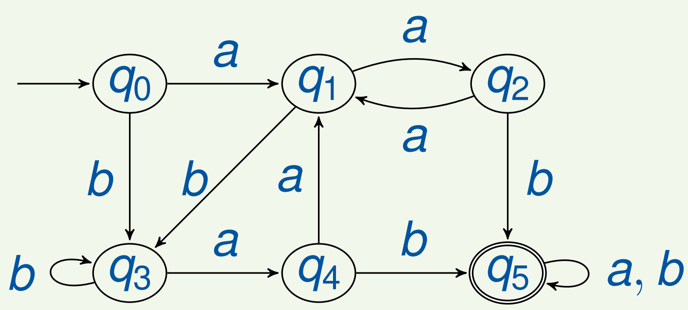
	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X		b		b	ε
q_1	X	X	b		b	ε
q_2	X	X	X	b		ε
q_3	X	X	X	X	b	ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

Algorithm A.55:

2. If $(\delta(p, b), \delta(q, b))$ marked by ε, then mark (p, q) by b

Example A.56

Given DFA:



Equivalence matrix:

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X	ab	b	ab	b	ε
q_1	X	X	b		b	ε
q_2	X	X	X	b		ε
q_3	X	X	X	X	b	ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

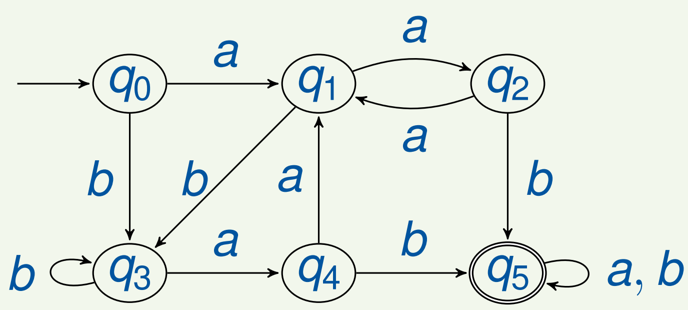
Algorithm A.55:

2. If $(\delta(p, a), \delta(q, a))$ marked by $c \in \{a, b\}$, then mark (p, q) by ac

Minimisation Example

Example A.56

Given DFA:



Equivalence matrix:

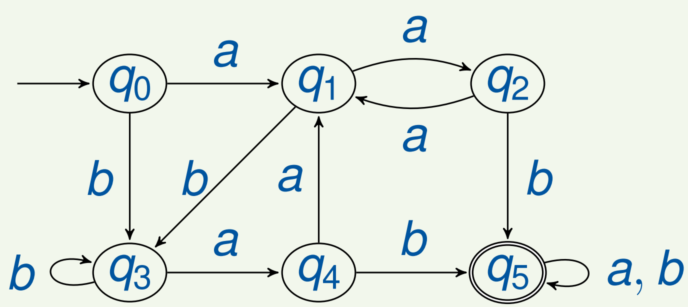
	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X	ab	b	ab	b	ε
q_1	X	X	b		b	ε
q_2	X	X	X	b		ε
q_3	X	X	X	X	b	ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

Algorithm A.55:

2. If $(\delta(p, b), \delta(q, b))$ marked by $c \in \{a, b\}$, then mark (p, q) by bc (not applicable)

Example A.56

Given DFA:



Equivalence matrix:

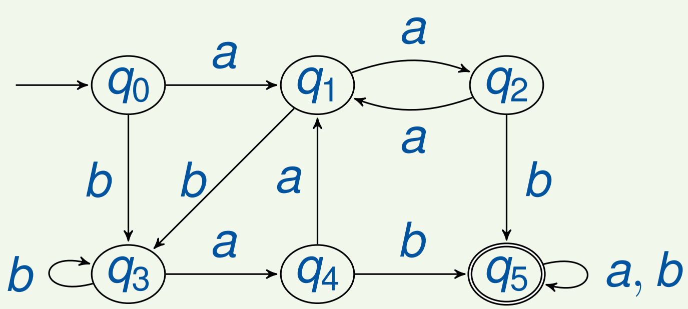
	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X	ab	b	ab	b	ε
q_1	X	X	b	✓	b	ε
q_2	X	X	X	b	✓	ε
q_3	X	X	X	X	b	ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

Algorithm A.55:

3. No further changes $\Longrightarrow$ $(q_1, q_3), (q_2, q_4)$ equivalent

Example A.56

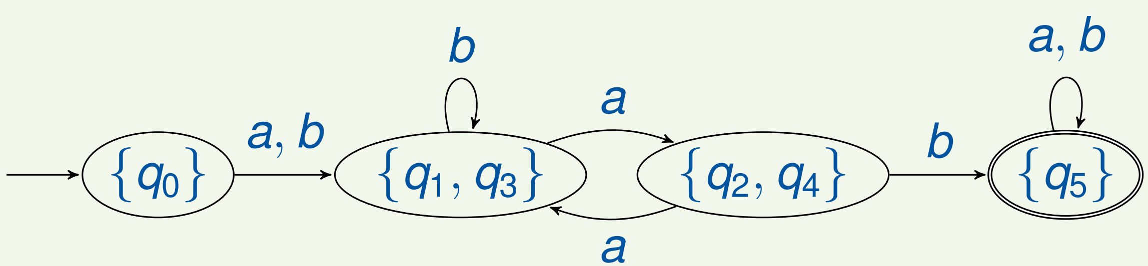
Given DFA:



Equivalence matrix:

	q_0	q_1	q_2	q_3	q_4	q_5
q_0	X	ab	b	ab	b	ε
q_1	X	X	b	$\checkmark$	b	ε
q_2	X	X	X	b	$\checkmark$	ε
q_3	X	X	X	X	b	ε
q_4	X	X	X	X	X	ε
q_5	X	X	X	X	X	X

Resulting minimal DFA:



Correctness of Minimisation

Theorem A.57

For every DFA $\mathfrak{A}$,

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

Correctness of Minimisation

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)

Outline of Part A

Outlook

- Pumping Lemma (to prove non-regularity of languages)
  - can be used to show that $\{a^n b^n \mid n \geq 1\}$ is not regular
- More language operations (homomorphisms, ...)
- Construction of scanners for compilers