Foundations of Informatics: a Bridging Course

Week 3: Formal Languages and Processes
Part A: Regular Languages
b-it Bonn, 29 February – 4 March 2016

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ws-1516/foi/
Organisation

- **Schedule:**
  - lecture 9:00-10:30, 11:00-12:30 (Mon-Thu)
    - 10:00-11:30, 11:45-13:15?
  - exercises 14:00-14:45, 15:15-16:00 (Mon-Thu)
    - 14:00-15:30?
  - Friday session?

- **Bridging Course exam** on Thursday, 30 March 2016, 10:00-13:00, b-it

- Please ask questions!
Overview of Week 3

1. Regular Languages
2. Context-Free Languages
Literature


• A. Asteroth, C. Baier: *Theoretische Informatik*, Pearson Studium, 2002 [in German]

• http://www.jflap.org/
  (software for experimenting with formal languages and automata)
Formal Languages

Outline of Part A

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Formal Languages

Words and Languages

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

Data sets = sets of words = formal languages,
data transformations = functions on words
Formal Languages

Words and Languages

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

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

Example A.1

\[ \text{Java} = \{ \text{all valid Java programs} \}, \]

Compiler : Java $\rightarrow$ Bytecode
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”).

Σ, Γ, . . . denote alphabets

a, b, . . . denote letters
Formal Languages

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”).

Σ, Γ, . . . denote alphabets

a, b, . . . denote letters

**Example A.3**

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

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 \( \Sigma_{\text{latin}} := \{a, b, c, \ldots, z\} \)
3. Keyboard alphabet \( \Sigma_{\text{key}} \)
Formal Languages

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 \( \Sigma_{\text{latin}} := \{a, b, c, \ldots, z\} \)
3. Keyboard alphabet \( \Sigma_{\text{key}} \)
4. Morse alphabet \( \Sigma_{\text{morse}} := \{\cdot, -, \Box\} \)
Words

Definition A.4

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

#### Words

<table>
<thead>
<tr>
<th>Definition A.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A <strong>word</strong> is a finite sequence of letters from a given alphabet $\Sigma$.</td>
</tr>
<tr>
<td>- $\Sigma^*$ is the set of all words over $\Sigma$.</td>
</tr>
<tr>
<td>- $</td>
</tr>
<tr>
<td>- The <strong>empty word</strong> is denoted by $\varepsilon$, i.e., $</td>
</tr>
</tbody>
</table>
Formal Languages

Words

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., $|a_1 \ldots a_n| := n$.
- The empty word is denoted by $\varepsilon$, i.e., $|\varepsilon| = 0$.
- The concatenation of two words $v = a_1 \ldots a_m$ ($m \in \mathbb{N}$) and $w = b_1 \ldots b_n$ ($n \in \mathbb{N}$) is the word
  \[
  v \cdot w := a_1 \ldots a_m b_1 \ldots b_n
  \]
  (often written as $vw$).
- Thus: $w \cdot \varepsilon = \varepsilon \cdot w = w$. 
Formal Languages

Words

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., $|a_1 \ldots a_n| := n$.
- The empty word is denoted by $\varepsilon$, i.e., $|\varepsilon| = 0$.
- The concatenation of two words $v = a_1 \ldots a_m (m \in \mathbb{N})$ and $w = b_1 \ldots b_n (n \in \mathbb{N})$ is the word
  $$v \cdot w := a_1 \ldots a_m b_1 \ldots b_n$$
  (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 $v' \in \Sigma^*$.
Formal Languages

Words

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., \(|a_1 \ldots a_n| := n\).
- The empty word is denoted by \( \varepsilon \), i.e., \(|\varepsilon| = 0\).
- The concatenation of two words \( v = a_1 \ldots a_m (m \in \mathbb{N}) \) and \( w = b_1 \ldots b_n (n \in \mathbb{N}) \) is the word
  \[
  v \cdot w := a_1 \ldots a_mb_1 \ldots b_n
  \]
  (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 \( w = a_1 \ldots a_n \), then \( w^R := a_n \ldots a_1 \).
Formal Languages

Formal Languages I

Definition A.5

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

<table>
<thead>
<tr>
<th>Example A.6</th>
<th>1. over ( B = {0, 1} ): set of all bit strings containing 1101</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. over ( \Sigma = {I, V, X, L, C, D, M} ): set of all valid roman numbers</td>
</tr>
<tr>
<td></td>
<td>3. over ( \Sigma \text{ key} ): set of all valid Java programs</td>
</tr>
</tbody>
</table>
Formal Languages I

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
Formal Languages

Formal Languages I

Definition A.5

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

Example A.6

1. over $B = \{0, 1\}$: 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

Formal Languages I

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
Formal Languages

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

Open:
- Description of computations on words?
Finite Automata

Outline of Part A

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Finite Automata

Outline of Part A

Formal Languages

Finite Automata

Deterministic Finite Automata
Operations on Languages and Automata
Nondeterministic Finite Automata
More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
# Finite Automata

## 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: $\varepsilon$, 1, 11, 110, 1101
5. Diagram: on the board
Finite Automata

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: $\varepsilon$, 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
Finite Automata

Deterministic Finite Automata I

Definition A.8

A deterministic finite automaton (DFA) 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
- \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function
- \( q_0 \in Q \) is the initial state
- \( F \subseteq Q \) is the set of final (or: accepting) states
Finite Automata

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
- $F = \{q_4\}$
Finite Automata

Deterministic Finite Automata II

Example A.9

Pattern matching (Example A.7):
- \( Q = \{q_0, \ldots, q_4\} \)
- \( \Sigma = \{0, 1\} \)
- \( \delta : Q \times \Sigma \rightarrow Q \) on the board
- \( F = \{q_4\} \)

Graphical Representation of DFA:
- states \( \rightarrow \) nodes
- \( \delta(q, a) = q' \rightarrow q \xrightarrow{a} q' \)
- initial state: incoming edge without source state
- final state(s): double circle
Finite Automata

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 \( \delta^*(q, w) = q_n \)
Finite Automata

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 \( q \overset{a_1}{\to} q_1 \overset{a_2}{\to} \ldots \overset{a_n}{\to} q_n \), then \( \delta^*(q, w) = q_n \)

Example A.11

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

Acceptance by DFA II

Definition A.12

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

- A language $L \subseteq \Sigma^*$ is called DFA-recognisable 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 recognised by the automaton from Example A.9.
Finite Automata

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

\[ \{ w \in \mathbb{B}^* | w \text{ contains 1} \} : \]

on the board
Finite Automata

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
   \[ \{ w \in \{0,1\}^* \mid w \text{ contains 1} \} : \]
   on the board
3. An automaton which recognises
   \[ \{ w \in \{0,\ldots,9\}^* \mid \text{value of } w \text{ divisible by 3} \} \]
   Idea: test whether sum of digits is divisible by 3 – one state for each residue class (on the board)
Finite Automata

Deterministic Finite Automata

Seen:

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

Deterministic Finite Automata

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

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

Outline of Part A

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Finite Automata

Operations on Languages

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

Question

Let $A_1$, $A_2$ be two DFA with $L(A_1) = L_1$ and $L(A_2) = L_2$. Can we construct automata which recognise

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

Language Complement

Theorem A.14

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

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

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

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

Example A.15 on the board

22 of 67 Foundations of Informatics/Formal Languages and Processes, Part A
Thomas Noll
b-it Bonn, 29 February – 4 March 2016
Finite Automata

Language Complement

Theorem A.14

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

Proof.

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

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

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

\( \square \)
Finite Automata

Language Complement

Theorem A.14

If $L \subseteq \Sigma^*$ is DFA-recognisable, 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 recognised by the DFA $\langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$. 

Example A.15

on the board
Finite Automata

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 $A_i = \langle Q_i, \Sigma, \delta_i, q_{i0}, F_i \rangle$ be DFA such that $L(A_i) = L_i$ ($i = 1, 2$). The new automaton $A$ has to accept $w$ iff $A_1$ and $A_2$ accept $w$.

Idea:

- Let $A_1$ and $A_2$ 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
**Finite Automata**

**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 \( \mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle \) 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
Language Intersection II

Proof (continued).

Formally: let the product automaton

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

be defined by

\[ \delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \]

for every \( a \in \Sigma \).
Finite Automata

Language Intersection II

Proof (continued).

Formally: let the product automaton

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

be defined by

\[ \delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \]

for every \( a \in \Sigma \).

This definition yields (for every \( w \in \Sigma^* \)):

\[ \delta^*(((q_1, q_2), w) = (\delta_1^*(q_1, w), \delta_2^*(q_2, w)) \quad (\ast) \]
Finite Automata

Language Intersection II

Proof (continued).

Formally: let the product automaton

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

be defined by

\[ \delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a)) \]

for every \( a \in \Sigma \).

This definition yields (for every \( w \in \Sigma^* \)):

\[ \delta^*((q_1, q_2), w) = (\delta_1^*(q_1, w), \delta_2^*(q_2, w)) \quad (\ast) \]

Thus we have:

- \( \mathcal{A} \) accepts \( w \)
  \[ \iff \delta^*((q_0^1, q_0^2), w) \in F_1 \times F_2 \]
  \[ \iff (\delta_1^*(q_0^1, w), \delta_2^*(q_0^2, w)) \in F_1 \times F_2 \]
  \[ \iff \delta_1^*(q_0^1, w) \in F_1 \text{ and } \delta_2^*(q_0^2, w) \in F_2 \]
  \[ \iff \mathcal{A}_1 \text{ accepts } w \text{ and } \mathcal{A}_2 \text{ accepts } w \]

Example A.17

on the board
Finite Automata

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 $A_i = \langle Q_i, \Sigma, \delta_i, q_i^0, F_i \rangle$ be DFA such that $L(A_i) = L_i$ ($i = 1, 2$). The new automaton $A$ has to accept $w$ iff $A_1$ or $A_2$ accepts $w$.

Idea: reuse product construction. Construct $A$ 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 $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$.
**Theorem A.18**

If \( L_1, L_2 \subseteq \Sigma^* \) are DFA-recognisable, 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 \( w \).
Finite Automata

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 \( \mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle \) 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

\[
F := (F_1 \times Q_2) \cup (Q_1 \times F_2).
\]
Finite Automata

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:

\[
w \cdot L := \{ w \} \cdot L, \quad L \cdot w := L \cdot \{ w \}
\]
Finite Automata

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

**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 $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$ 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 recognised 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-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 $\mathcal{A}_i = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle$ 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 recognised 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
The $n$th power of a language $L \subseteq \Sigma^*$ is the $n$-fold concatenation of $L$ with itself ($n \in \mathbb{N}$):

$$L^n := L \cdots \cdot L = \{w_1 \ldots w_n \mid \forall i \in \{1, \ldots, n\} : w_i \in L\}.$$ 

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

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

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

Language Iteration

Definition A.21

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

\[
L^n := \underbrace{L \cdots L}_{n \text{ times}} = \{ w_1 \ldots w_n | \forall i \in \{1, \ldots, n\} : w_i \in L \}.
\]

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

• The iteration (or: Kleene star) of \( L \) is

\[
L^* := \bigcup_{n \in \mathbb{N}} L^n = \{ w_1 \ldots w_n | n \in \mathbb{N}, \forall i \in \{1, \ldots, n\} : w_i \in L \}.
\]

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
Finite Automata

Operations on Languages and Automata

Seen:

- Operations on languages:
  - complement
  - intersection
  - union
  - concatenation
  - iteration

- DFA constructions for:
  - complement
  - intersection
  - union
Finite Automata

Operations on Languages and Automata

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?
Finite Automata

Outline of Part A

Formal Languages

Finite Automata
   Deterministic Finite Automata
   Operations on Languages and Automata
   Nondeterministic Finite Automata
   More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
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: $B^* \cdot 1101 \cdot B^*$; 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, ...
Finite Automata

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: $B^* \cdot 1101 \cdot B^*$; 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, ...)

31 of 67 Foundations of Informatics/Formal Languages and Processes, Part A
Thomas Noll
b-it Bonn, 29 February – 4 March 2016
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
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 \(((q, a, q') \in \Delta \iff \delta(q, a) = q')\)
Finite Automata

Acceptance by NFA

Definition A.23

- Let $w = a_1 \ldots a_n \in \Sigma^*$.
- A $w$-labelled $\mathcal{A}$-run from $q_1$ to $q_2$ is a sequence
  $$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots p_{n-1} \xrightarrow{a_n} p_n$$
  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: $q_1 \xrightarrow{w} q_2$).
- $\mathcal{A}$ accepts $w$ if there is a $w$-labelled $\mathcal{A}$-run from $q_0$ to some $q \in F$.
- The language recognised by $\mathcal{A}$ is
  $$L(\mathcal{A}) := \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}.$$
- A language $L \subseteq \Sigma^*$ is called NFA-recognisable if there exists a NFA $\mathcal{A}$ such that $L(\mathcal{A}) = L$.
- Two NFA $\mathcal{A}_1, \mathcal{A}_2$ are called equivalent if $L(\mathcal{A}_1) = L(\mathcal{A}_2)$. 
Finite Automata

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 \( w = a_1 \ldots a_n \):

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

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

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

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 $w = a_1 \ldots a_n$:

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

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
Finite Automata

NFA-Recognisability of Concatenation

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

NFA-Recognisability of Concatenation

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

**Solution:** admit empty word $\varepsilon$ as transition label
Finite Automata

\( \varepsilon \)-NFA

**Definition A.26**

A nondeterministic finite automaton with \( \varepsilon \)-transitions (\( \varepsilon \)-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
- \( \Delta \subseteq Q \times \Sigma_{\varepsilon} \times Q \) 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 \( \varepsilon \)-NFA
- definitions of runs and acceptance: in analogy to NFA
Finite Automata

\( \varepsilon \)-NFA

Definition A.26

A nondeterministic finite automaton with \( \varepsilon \)-transitions (\( \varepsilon \)-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
- \( \Delta \subseteq Q \times \Sigma \varepsilon \times Q \) 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 \( \varepsilon \)-NFA
- definitions of runs and acceptance: in analogy to NFA

Example A.27

on the board
Finite Automata

Concatenation and Iteration via $\varepsilon$-NFA

**Theorem A.28**

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

**Proof (idea).**
Finite Automata

Concatenation and Iteration via $\varepsilon$-NFA

Theorem A.28

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

Proof (idea).

on the board
# Finite Automata

## Concatenation and Iteration via $\varepsilon$-NFA

### Theorem A.28

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

### Proof (idea).

on the board

### Theorem A.29

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

### Proof.

see Theorem A.47
Finite Automata

Syntax Diagrams as $\varepsilon$-NFA

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

Example A.30

decimal numbers (on the board)
Finite Automata

Types of Finite Automata

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

From the definitions we immediately obtain:

Corollary A.31

1. Every DFA-recognisable language is NFA-recognisable.
2. Every NFA-recognisable language is $\varepsilon$-NFA-recognisable.

Goal: establish reverse inclusions
Finite Automata

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.31

1. Every DFA-recognisable language is NFA-recognisable.
2. Every NFA-recognisable language is ε-NFA-recognisable.
Finite Automata

Types of Finite Automata

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

From the definitions we immediately obtain:

Corollary A.31

1. Every DFA-recognisable language is NFA-recognisable.
2. Every NFA-recognisable language is $\varepsilon$-NFA-recognisable.

Goal: establish reverse inclusions
Finite Automata

From NFA to DFA I

Theorem A.32

Every NFA can be transformed into an equivalent DFA.
Finite Automata

From NFA to DFA I

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 := \{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
Finite Automata

From NFA to DFA II

Proof (continued).

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

Powerset construction of $\mathcal{A}' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$:
- $Q' := 2^Q := \{ P \mid P \subseteq Q \}$
- $\delta' : Q' \times \Sigma \rightarrow Q'$ 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

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

and thus

$\mathcal{A}$ accepts $w \iff \mathcal{A}' \text{ accepts } w$
Finite Automata

From NFA to DFA II

Proof (continued).

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

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

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

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

and thus

$\mathcal{A}$ accepts $w \iff \mathcal{A}'$ accepts $w$

Example A.33

on the board
Finite Automata

From $\varepsilon$-NFA to NFA

**Theorem A.34**

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

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.
Finite Automata

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 $p \overset{\varepsilon}{\longrightarrow}^* q$, $q \overset{a}{\longrightarrow} q'$, and $q' \overset{\varepsilon}{\longrightarrow}^* r$ in $\mathcal{A}$, then $p \overset{a}{\longrightarrow} r$ in $\mathcal{A}'$. Moreover $F' := F \cup \{q_0\}$ if $q_0 \overset{\varepsilon}{\longrightarrow}^* q \in F$ in $\mathcal{A}$, and $F' := F$ otherwise.

**Example A.35**

on the board
Finite Automata

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

**Corollary A.36**

*All types of finite automata recognise the same class of languages.*
Finite Automata

Nondeterministic Finite Automata

Seen:
- Definition of $\varepsilon$-NFA
- Determinisation of ($\varepsilon$-)NFA
Finite Automata

Nondeterministic Finite Automata

Seen:
- Definition of $\varepsilon$-NFA
- Determinisation of $(\varepsilon)$-NFA

Open:
- More decidablity results
Finite Automata

Outline of Part A

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Finite Automata

The Word Problem Revisited

Definition A.37

The \textit{word problem for DFA} is specified as follows:

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

$$w \in L(A).$$
Finite Automata

The Word Problem Revisited

Definition A.37

The word problem for DFA is specified as follows:

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

$$w \in L(A).$$

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.
Finite Automata

The Emptiness Problem

Definition A.39

The emptiness problem for DFA is specified as follows:

Given a DFA $A$, decide whether $L(A) = \emptyset$. 
Finite Automata

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).
Finite Automata

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)
**Finite Automata**

**The Equivalence Problem**

<table>
<thead>
<tr>
<th>Definition A.41</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The equivalence problem for DFA</strong> is specified as follows:</td>
</tr>
<tr>
<td><strong>Given two DFA</strong> $A_1, A_2$, decide whether</td>
</tr>
</tbody>
</table>
| $L(A_1) = L(A_2)$.
### The Equivalence Problem

<table>
<thead>
<tr>
<th>Definition A.41</th>
</tr>
</thead>
<tbody>
<tr>
<td>The equivalence problem for DFA is specified as follows:</td>
</tr>
<tr>
<td>Given two DFA $A_1$, $A_2$, decide whether $L(A_1) = L(A_2)$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem A.42</th>
</tr>
</thead>
<tbody>
<tr>
<td>The equivalence problem for DFA (NFA, $\varepsilon$-NFA) is <strong>decidable</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L(A_1) = L(A_2)$</td>
</tr>
</tbody>
</table>
## Finite Automata

### The Equivalence Problem

#### Definition A.41

The equivalence problem for DFA is specified as follows:

Given two DFA $A_1, A_2$, decide whether

\[ L(A_1) = L(A_2). \]

#### Theorem A.42

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

#### Proof.

\[
L(A_1) = L(A_2) \iff L(A_1) \subseteq L(A_2) \text{ and } L(A_2) \subseteq L(A_1)
\]
Finite Automata

The Equivalence Problem

Definition A.41
The equivalence problem for DFA is specified as follows:
Given two DFA $A_1$, $A_2$, decide whether
$$L(A_1) = L(A_2).$$

Theorem A.42
The equivalence problem for DFA (NFA, $\varepsilon$-NFA) is decidable.

Proof.
$$L(A_1) = L(A_2)$$
$$\iff L(A_1) \subseteq L(A_2) \text{ and } L(A_2) \subseteq L(A_1)$$
$$\iff (L(A_1) \setminus L(A_2)) \cup (L(A_2) \setminus L(A_1)) = \emptyset$$
Finite Automata

The Equivalence Problem

Definition A.41

The equivalence problem for DFA is specified as follows:

Given two DFA $A_1, A_2$, decide whether

$$L(A_1) = L(A_2).$$

Theorem A.42

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

Proof.

$$L(A_1) = L(A_2)$$

$\iff L(A_1) \subseteq L(A_2)$ and $L(A_2) \subseteq L(A_1)$

$\iff (L(A_1) \setminus L(A_2)) \cup (L(A_2) \setminus L(A_1)) = \emptyset$

$\iff (L(A_1) \cap \overline{L(A_2)}) \cup (L(A_2) \cap \overline{L(A_1)}) = \emptyset$  

DFA-recognisable (Thm. A.14)  

DFA-recognisable (Thm. A.14)

DFA-recognisable (Thm. A.16)  

DFA-recognisable (Thm. A.16)

DFA-recognisable (Thm. A.18)

decidable (Thm. A.40)
Finite Automata

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

Open:
- Non-algorithmic description of languages
Regular Expressions

Outline of Part A

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Regular Expressions

An Example

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:

\[
(a + aaa)(bb^{*}ab^{*}ab^{*} + b^{*}abb^{*}ab^{*} + b^{*}ab^{*}abb^{*})^{*}b^{*}
\]

b before \( a \)'s

b between \( a \)'s

b after \( a \)'s
Regular Expressions

Syntax of Regular Expressions

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
  - $\alpha + \beta$
  - $\alpha \cdot \beta$
  - $\alpha^*$
Regular Expressions

Syntax of Regular Expressions

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 $\alpha \cdot \alpha^*$
**Semantics of Regular Expressions**

**Definition A.45**

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

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

Semantics of Regular Expressions

Definition A.45

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

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

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

Regular Languages

Example A.46

1. \( \{ aa \} \) is regular since

\[
L(a \cdot a) = L(a) \cdot L(a) = \{ a \} \cdot \{ a \} = \{ aa \}
\]
Regular Expressions

Regular Languages

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

\[ L((a + b)^*) = (L(a + b))^* = (L(a) \cup L(b))^* = (\{a\} \cup \{b\})^* = \{a, b\}^* \]
Regular Expressions

Regular Languages

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 \textit{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 $A_\alpha$

- 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-recognisable
- $L$ is NFA-recognisable
- $L$ is $\varepsilon$-NFA-recognisable
### Implementation of Pattern Matching

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

**Output:** “yes” if $A$ passes through some final state, otherwise “no”

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

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>printable character</td>
<td>this character</td>
</tr>
<tr>
<td>\n, \t, \123, etc.</td>
<td>newline, tab, octal representation, etc.</td>
</tr>
<tr>
<td>.</td>
<td>any character except \n</td>
</tr>
<tr>
<td>[Chars]</td>
<td>one of Chars; ranges possible (“0–9”)</td>
</tr>
<tr>
<td>[^Chars]</td>
<td>none of Chars</td>
</tr>
<tr>
<td>, , [, etc.</td>
<td>, , [, etc.</td>
</tr>
<tr>
<td>&quot;Text&quot;</td>
<td>Text without interpretation of ., [, , etc.</td>
</tr>
<tr>
<td>^α</td>
<td>α at beginning of line</td>
</tr>
<tr>
<td>α$</td>
<td>α at end of line</td>
</tr>
<tr>
<td>α?</td>
<td>zero or one α</td>
</tr>
<tr>
<td>α*</td>
<td>zero or more α</td>
</tr>
<tr>
<td>α+</td>
<td>one or more α</td>
</tr>
<tr>
<td>α{n, m}</td>
<td>between n and m times α (“, m” optional)</td>
</tr>
<tr>
<td>(α)</td>
<td>α</td>
</tr>
<tr>
<td>α₁α₂</td>
<td>concatenation</td>
</tr>
<tr>
<td>α₁</td>
<td>α₂</td>
</tr>
</tbody>
</table>
Regular Expressions

Regular Expressions

Seen:

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

Outline of Part A

Formal Languages

Finite Automata
   Deterministic Finite Automata
   Operations on Languages and Automata
   Nondeterministic Finite Automata
   More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
Minimisation of DFA

Motivation

**Goal:** space-efficient implementation of regular languages

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

**Wanted:** DFA $\mathcal{A}_{\text{min}} = \langle Q', \Sigma, \delta', q_0', F' \rangle$ such that $L(\mathcal{A}_{\text{min}}) = L(\mathcal{A})$ and $|Q'|$ minimal
Minimisation of DFA

State Equivalence

Example A.50

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

\[
\begin{array}{c}
\text{a, b} \\
\text{a} \\
\text{q_0} \\
\text{b} \\
\text{a} \\
\text{q_1} \\
\text{b} \\
\text{a} \\
\text{q_2} \\
\end{array}
\]

Observation:

\{q_0, q_2\} and \{q_0, q_1, q_2\} are equivalent

Definition A.51

Given DFA \(A = \langle Q, \Sigma, \delta, q_0, F \rangle\), states \(p, q \in Q\) are equivalent if

\[\forall w \in \Sigma^* : \delta^*(p, w) \in F \iff \delta^*(q, w) \in F.\]
Minimisation of DFA

State Equivalence

Example A.50

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

\[
\begin{array}{cccc}
& a, b & a, b \\
q_0 & q_1 & q_2 \\
\end{array}
\]

Powerset construction yields DFA \(\mathcal{A}\):

\[
\begin{array}{cccc}
& a & b & a \\
\{q_0\} & \{q_0, q_1\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} \\
\end{array}
\]

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

\[
\forall w \in \Sigma^*: \delta^*(p, w) \in F \iff \delta^*(q, w) \in F.
\]
Minimisation of DFA

State Equivalence

Example A.50

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

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_0 & \xrightarrow{b} q_0 \\
q_1 & \xrightarrow{a} q_1 \\
q_2 & \xrightarrow{a} q_2
\end{align*}
\]

Powerset construction yields DFA \(\mathcal{A}\):

\[
\begin{align*}
\{q_0\} & \xrightarrow{a} \{q_0, q_1\} \\
\{q_0, q_1\} & \xrightarrow{b} \{q_0, q_2\} \\
\{q_0, q_2\} & \xrightarrow{a} \{q_0, q_1, q_2\} \\
\{q_0, q_1, q_2\} & \xrightarrow{a} \{q_0, q_1, q_2\}
\end{align*}
\]

Observation: \(\{q_0, q_2\}\) and \(\{q_0, q_1, q_2\}\) are equivalent
Minimisation of DFA

State Equivalence

**Example A.50**

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

\[
\begin{array}{c}
 q_0 \\
 \downarrow a \\
\{ q_0 \}
\end{array} \quad \begin{array}{c}
 q_1 \\
 \downarrow b \\
\{ q_0, q_1 \}
\end{array} \quad \begin{array}{c}
 q_2 \\
 \uparrow a \\
\{ q_0, q_2 \}
\end{array}
\]

Powerset construction yields DFA \(\mathcal{A}\):

\[
\begin{array}{c}
 \{ q_0 \} \\
 \downarrow a \\
\{ q_0, q_1 \}
\end{array} \quad \begin{array}{c}
 \{ q_0, q_2 \} \\
 \downarrow b \\
\{ q_0, q_1, q_2 \}
\end{array}
\]

**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

\[
\forall w \in \Sigma^*: \delta^*(p, w) \in F \iff \delta^*(q, w) \in F.
\]
Minimisation of DFA

Minimisation

Minimisation: merging of equivalent states

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

DFA after state merging:

\[ \begin{array}{c}
\text{b} \\
\text{a} \\
\text{b} \\
\end{array} \]

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 \( \delta^*(p, w) \in F \) and \( \delta^*(q, w) \notin F \) (or vice versa, i.e., \( p \) and \( q \) can be distinguished by \( w \))
Minimisation of DFA

Minimisation

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 \( w \))
Lemma A.54

**Inductive characterisation of state inequivalence:**

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

Computing State (In-)Equivalence

Lemma A.54

*Inductive characterisation of state inequivalence:*

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

Algorithm A.55 (State Equivalence for DFA)

*Input:* \( \mathcal{A} = \langle Q, \Sigma, \Delta, q_0, F \rangle \)

*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
Minimisation of DFA

Minimisation Example

Example A.56

Given DFA:

Equivalence matrix: on the board
Minimisation Example

Example A.56

Given DFA:

Equivalence matrix: on the board

Resulting minimal DFA:
Minimisation of DFA

Correctness of Minimisation

Theorem A.57

For every DFA $A$, 

$$L(A) = L(A_{\text{min}})$$

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

$$\forall \text{DFA } A, B: L(A) = L(B) \Rightarrow A_{\text{min}} \approx B_{\text{min}}$$

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

Correctness of Minimisation

Theorem A.57

For every DFA $A$, 

$$L(A) = L(A_{\text{min}})$$

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

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

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

Formal Languages

Finite Automata
  Deterministic Finite Automata
  Operations on Languages and Automata
  Nondeterministic Finite Automata
  More Decidability Results

Regular Expressions

Minimisation of DFA

Outlook
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