Foundations of Informatics: a Bridging Course

Week 3: Formal Languages and Semantics

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www.b-it-center.de/Wob/en/view/class211_id569.html

B–IT, Bonn, Winter term 2006/07

Organization

- Schedule:
  - lecture 9:00–12:30 (Mon–Fri)
  - exercises 14:00–16:00 (Mon–Thu)
  - 30 min break in each block
- Examination after week 4
- Please ask questions!

Overview of Week 3

1. Regular Languages
2. Context–Free Languages
3. Processes and Concurrency

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 concepts)

Part I

Regular Languages

Outline

Words and Languages

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

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

Words and Languages

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

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

Example I.1

C++ = {all valid C++ programs},

$\mathit{Compiler}$: C++ $\rightarrow$ Machine code

Alphabets

Definition I.2

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

$\Sigma, \Gamma, \ldots$ denote alphabets

$a, b, \ldots$ denote letters

Alphabets

Definition I.2

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

$\Sigma, \Gamma, \ldots$ denote alphabets

$a, b, \ldots$ denote letters

Example I.3

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

Alphabets

Definition I.2

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

$\Sigma, \Gamma, \ldots$ denote alphabets

$a, b, \ldots$ denote letters

Example I.3

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

Alphabets

Definition I.2

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

$\Sigma, \Gamma, \ldots$ denote alphabets

$a, b, \ldots$ denote letters

Example I.3

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

Alphabets

Definition I.2

An alphabet is a finite, non–empty set of symbols ("letters").

$\Sigma, \Gamma, \ldots$ denote alphabets

$a, b, \ldots$ denote letters

Example I.3

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

Words

Definition I.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 I.5

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

Definition I.5

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

Example I.6

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

Formal Languages I

Definition I.5

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

Example I.6

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

Formal Languages I

Definition I.5

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

Example I.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 C++ 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

Open:

- Description of computations on words?

Outline

Outline

Example: Pattern Matching

Example I.7

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

Example: Pattern Matching

Example I.7

1. Read Boolean string bit by bit
2. Test whether it contains 1101
3. Idea: remember which (initial) part of 1101 has been recognized
4. Five prefixes: 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 succesful state

Deterministic Finite Automata I

Definition I.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 I.9

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

Acceptance by DFA I

Definition I.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

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

Formally:

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

Acceptance by DFA I

Definition I.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

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

Formally:

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

Example I.11

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

Acceptance by DFA II

Definition I.12

- $\mathfrak{A}$ accepts $w \in \Sigma^*$ if $\delta^*(q_0, w) \in F$.
- The language recognized 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–recognizable if there exists a 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 I.13

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

Example I.13

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

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

on the board

Acceptance by DFA III

Example I.13

1. The set of all bit strings containing $\mathsf{1101}$ is recognized by the automaton from Example I.9.
2. Two (equivalent) automata recognizing the language

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

   on the board
3. An automaton which recognizes

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

   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
- Recognizability of formal languages by automata

Seen:

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

Open:

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

Outline

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 recognize

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

Language Complement

Theorem I.14

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

Language Complement

Theorem I.14

If $L \subseteq \Sigma^*$ is DFA–recognizable, 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:

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

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

Language Complement

Theorem I.14

If $L \subseteq \Sigma^*$ is DFA–recognizable, 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:

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

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

Example I.15

on the board

Language Intersection I

Theorem I.16

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

Language Intersection I

Theorem I.16

If $L_1, L_2 \subseteq \Sigma^*$ are DFA–recognizable, 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 *both* $\mathfrak{A}_1$ *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 separately
- for acceptance both components need to be in a final state

$\square$

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

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

for every $w \in \Sigma^*$.
Thus we have:

$$\begin{aligned} & \mathfrak{A} \text{ accepts } w \\ \iff\quad & \delta^*((q_0^1, q_0^2), w) \in F_1 \times F_2 \\ \iff\quad & (\delta_1^*(q_0^1, w), \delta_2^*(q_0^2, w)) \in F_1 \times F_2 \\ \overset{(*)}{\iff}\quad & \delta_1^*(q_0^1, w) \in F_1 \text{ and } \delta_2^*(q_0^2, w) \in F_2 \\ \iff\quad & \mathfrak{A}_1 \text{ accepts } w \text{ and } \mathfrak{A}_2 \text{ accepts } w \end{aligned}$$

□

Language Intersection III

Example I.17

on the board

Language Union

Theorem I.18

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

Language Union

Theorem I.18

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

Proof.

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

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

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

$\square$

Language Concatenation

Definition I.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: $w \cdot L := \{w\} \cdot L$, $L \cdot w := L \cdot \{w\}$

Language Concatenation

Definition I.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 I.20

1. If $L_1 = \{101, 1\}$ and $L_2 = \{011, 1\}$, then

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

Language Concatenation

Definition I.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 I.20

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

DFA–Recognizability of Language Concatenation

Conjecture

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

DFA–Recognizability of Language Concatenation

Conjecture

If $L_1, L_2 \subseteq \Sigma^*$ are DFA–recognizable, 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 recognized by $\mathfrak{A}_1$, and if $\mathfrak{A}_2$ accepts the remaining suffix.
Idea: choose $Q := Q_1 \cup Q_2$ where F_1 and q_0^2 are identified
But: on the board □

DFA–Recognizability of Language Concatenation

Conjecture

If $L_1, L_2 \subseteq \Sigma^*$ are DFA–recognizable, 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 recognized by $\mathfrak{A}_1$, and if $\mathfrak{A}_2$ accepts the remaining suffix.

Idea: choose $Q := Q_1 \cup Q_2$ where F_1 and q_0^2 are identified

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 I.21

- The *nth power* of a language $L \subseteq \Sigma^*$ is the n–fold composition of L with itself ($n \in \mathbb{N}$): $L^n := \underbrace{L \cdot \ldots \cdot L}_{n \text{ times}}$.

  Inductively: $L^0 := \{\varepsilon\}$, $L^{n+1} := L^n \cdot L$
- The *iteration* (or: *Kleene star*) of L is

$$L^* := \bigcup_{n \in \mathbb{N}} L^n.$$

Language Iteration

Definition I.21

- The *nth power* of a language $L \subseteq \Sigma^*$ is the n–fold composition of L with itself ($n \in \mathbb{N}$): $L^n := \underbrace{L \cdot \ldots \cdot L}_{n \text{ times}}$.
  Inductively: $L^0 := \{\varepsilon\}$, $L^{n+1} := L^n \cdot L$
- The *iteration* (or: *Kleene star*) of L is

$$L^* := \bigcup_{n \in \mathbb{N}} L^n.$$

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–recognizable language is DFA–recognizable, 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

Open:

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

Outline

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

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

Nondeterministic Finite Automata II

Definition I.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

Remark: every DFA can be considered as an NFA
$((q, a, q') \in \Delta \iff \delta(q, a) = q')$

Acceptance by NFA

Definition I.23

- A w–labeled $\mathfrak{A}$–run from q_1 to q_2 is a sequence

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

  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: $q_1 \xrightarrow{w} q_2$).
- $\mathfrak{A}$ accepts w if there is a w–labeled $\mathfrak{A}$–run from q_0 to some $q \in F$
- The language recognized by $\mathfrak{A}$ is

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

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

Acceptance Test for NFA

Algorithm I.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: *successive computation of the reachability set*

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

Inductive definition:

$$\begin{aligned} R_{\mathfrak{A}}(\varepsilon) &:= \{q_0\} \\ R_{\mathfrak{A}}(va) &:= \{q \in Q \mid p \xrightarrow{a} q \text{ for some } p \in R_{\mathfrak{A}}(v)\} \end{aligned}$$

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 I.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: *successive computation of the reachability set*

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

Inductive definition:

$$\begin{aligned} R_{\mathfrak{A}}(\varepsilon) &:= \{q_0\} \\ R_{\mathfrak{A}}(va) &:= \{q \in Q \mid p \xrightarrow{a} q \text{ for some } p \in R_{\mathfrak{A}}(v)\} \end{aligned}$$

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

Remark: this algorithm solves the word problem for NFA

Example I.25

on the board

NFA–Recognizability of Language Concatenation

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

NFA–Recognizability of Language Concatenation

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

Solution: admit empty word ε as transition label

Definition I.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 with $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$
- $q_0 \in Q$ 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

ε–NFA

Definition I.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 with $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$
- $q_0 \in Q$ 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 I.27

on the board

ε–NFA–Recognizability of Language Concatenation

Theorem I.28

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

ε–NFA–Recognizability of Language Concatenation

Theorem I.28

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

Proof.

(idea) on the board $\square$

ε–NFA–Recognizability of Language Iteration

Theorem I.29

If $L \subseteq \Sigma^*$ is ε–NFA–recognizable, then so is L^*.

Theorem I.29

If $L \subseteq \Sigma^*$ is ε–NFA–recognizable, then so is L^*.

Proof.

(idea) on the board □

Syntax Diagrams as ε–NFA

Syntax diagrams (without recursive calls) can be interpreted as ε–NFA

Example I.30

decimal numbers (on the board)

Types of Finite Automata

1. DFA
2. NFA
3. ε–NFA

Types of Finite Automata

1. DFA
2. NFA
3. ε–NFA

Corollary I.31

1. *Every DFA–recognizable language is NFA–recognizable.*
2. *Every NFA–recognizable language is ε–NFA–recognizable.*

Types of Finite Automata

1. DFA
2. NFA
3. ε–NFA

Corollary I.31

1. *Every DFA–recognizable language is NFA–recognizable.*
2. *Every NFA–recognizable language is ε–NFA–recognizable.*

Goal: establish reverse inclusions

Corollary I.32

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

From NFA to DFA I

Theorem I.33

Every NFA can be transformed into an equivalent DFA.

Proof.

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

- $Q' := \mathsf{Pow}(Q) := \{P \mid P \subseteq Q\}$
- $\delta' : Q' \times \Sigma \to Q'$ with

$$q \in \delta'(P, a) \iff \text{there exists } p \in P \text{ such that } (p, a, q) \in \Delta$$

- $q_0' := \{q_0\}$
- $F' := \{P \subseteq Q \mid P \cap F \neq \emptyset\}$

This yields

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

and thus

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

Example I.34

on the board

From ε–NFA to NFA

Theorem I.35

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

From ε–NFA to NFA

Theorem I.35

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

Proof.

(idea) Let $\mathfrak{A}$ 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 $\mathfrak{A}'$. $\square$

From ε–NFA to NFA

Theorem I.35

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

Proof.

(idea) Let $\mathfrak{A}$ 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 $\mathfrak{A}'$. □

Example I.36

on the board

Nondeterministic Finite Automata

Seen:

- Definition of ε–NFA
- Determinization of (ε–)NFA

Seen:

- Definition of ε–NFA
- Determinization of (ε–)NFA

Open:

- More decidablity results

Outline

The Word Problem

Definition I.37

The word problem for DFA is specified as follows:

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

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

The Word Problem

Definition I.37

The word problem for DFA is specified as follows:

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

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

As we have seen:

Theorem I.38

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

The Emptiness Problem

Definition I.39

The emptiness problem for DFA is specified as follows:

Given a DFA $\mathfrak{A}$, decide whether

$$L(\mathfrak{A}) = \emptyset.$$

The Emptiness Problem

Definition I.39

The emptiness problem for DFA is specified as follows:

Given a DFA $\mathfrak{A}$, decide whether

$$L(\mathfrak{A}) = \emptyset.$$

Theorem I.40

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 Equivalence Problem

Definition I.41

The equivalence problem for DFA is specified as follows:

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

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

The Equivalence Problem

Definition I.41

The equivalence problem for DFA is specified as follows:

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

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

Theorem I.42

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

Proof.

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

The Equivalence Problem

Definition I.41

The equivalence problem for DFA is specified as follows:

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

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

Theorem I.42

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

Proof.

$$\begin{array}{ll} & L(\mathfrak{A}_1) = L(\mathfrak{A}_2) \\ \iff & (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset \end{array}$$

The Equivalence Problem

Definition I.41

The equivalence problem for DFA is specified as follows:

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

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

Theorem I.42

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

Proof.

$$\begin{aligned}
& L(\mathfrak{A}_1) = L(\mathfrak{A}_2) \\
\iff\ & (L(\mathfrak{A}_1) \setminus L(\mathfrak{A}_2)) \cup (L(\mathfrak{A}_2) \setminus L(\mathfrak{A}_1)) = \emptyset \\
\iff\ & \underbrace{\underbrace{\underbrace{(L(\mathfrak{A}_1) \cap \underbrace{\overline{L(\mathfrak{A}_2)}}_{\text{DFA–recognizable (Thm. I.14)}})}_{\text{DFA–recognizable (Thm. I.16)}} \cup \underbrace{(L(\mathfrak{A}_2) \cap \underbrace{\overline{L(\mathfrak{A}_1)}}_{\text{DFA–recognizable (Thm. I.14)}})}_{\text{DFA–recognizable (Thm. I.16)}}}_{\text{DFA–recognizable (Thm. I.18)}} = \emptyset}_{\text{decidable (Thm. I.40)}}
\end{aligned}$$

□

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

Open:

- Non–algorithmic description of languages

Outline

Example I.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*:

$$(a + aaa)(bb^*ab^*ab^* + b^*abb^*ab^* + b^*ab^*abb^*)^*b^*$$

Definition I.44

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 $\alpha \cdot \alpha^*$

Definition I.45

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

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

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 I.46

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

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

Regular Languages II

Example I.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 Languages II

Example I.46

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

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

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

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

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

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

Theorem I.47 (Kleene's Theorem)

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

Theorem I.47 (Kleene's Theorem)

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

Proof.

$\Longrightarrow$ using induction over the given regular expression α, we construct an ε–NFA $\mathfrak{A}_\alpha$

- with exactly one final state
- without transitions into the initial state
- without transitions leaving the final state

(on the board)

$\Longleftarrow$ by solving a regular equation system (details omitted)

$\square$

Corollary I.48

The following properties are equivalent:

- L *is regular*
- L *is DFA–recognizable*
- L *is NFA–recognizable*
- L *is* ε*–NFA–recognizable*

Implementation of Pattern Matching

Algorithm I.49 (Pattern Matching)

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

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

Procedure:

1. *let* $\beta := \Sigma^* \cdot \alpha$
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

Seen:

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

Regular Expressions

Seen:

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

Open:

- Limitations of regular languages?

Outline

Motivation

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

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

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

Theorem I.50 (Pumping Lemma for Regular Languages)

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

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

The Pumping Lemma II

Proof.

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

Then: $|w| \geq n$

$\Longrightarrow$ the accepting run visits $\geq n + 1$ states

$\Longrightarrow$ some $q \in Q$ occurs (at least) twice

Choose y to be the substring which is read between the two visits of q. Clearly, $y \neq \varepsilon$. Moreover the cycle can be omitted or repeated such that $xz \in L$, $xyz \in L$, $xy^2z \in L$, ... $\square$

The Pumping Lemma II

Proof.

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

Then: $\qquad |w| \geq n$

$\implies$ the accepting run visits $\geq n + 1$ states

$\implies$ some $q \in Q$ occurs (at least) twice

Choose y to be the substring which is read between the two visits of q. Clearly, $y \neq \varepsilon$. Moreover the cycle can be omitted or repeated such that $xz \in L$, $xyz \in L$, $xy^2z \in L$, ... $\square$

Remark: Pumping Lemma states a *necessary condition* for regularity

$\implies$ can only be used to show the non–regularity of a language

Example I.51

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

The Pumping Lemma III

Example I.51

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

The Pumping Lemma III

Example I.51

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

Conclusion

Finite automata are to weak for defining the syntax of programming languages!

The Pumping Lemma IV

Seen:

- Necessary condition for regularity of languages
- Counterexamples

Seen:

- Necessary condition for regularity of languages
- Counterexamples

Open:

- More expressive formalisms for describing languages?

Outline

Outlook

- Minimization of DFA
- More language operations (reverse, homomorphisms, ...)
- Construction of scanners for compilers