# Logic and Discrete Structures -LDS


Course 13 – Automata. Regular expressions


S.l. dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro


**Finite automata**

Languages

Deterministic Finite Automata (DFA)

Non-deterministic Finite Automata (NFA)

Regular expressions

# In today's course

Systems with simple behaviour: automata

a model for finite-memory computations

Languages (string sets) of a simple form:

concatenation, alternative, repetition

# An automaton example: the coffee machine

actions (user): insert coin, press button

response (automatic): give coffee

After an action does something happen?

| | |
|---|---|
| coin | No |
| button | not immediately |
| coin, button | Yes |

Coin had an internal effect: the machine switched to another state (it behaves differently when the button is pressed)

Coin coin button button gives two coffees?

if yes, how many coins can it remember?

one or more, but basically a finite number ⇒ finite states

# Automata in practice

Many systems can be modeled as automata:

- counters, displays, simple on/off control
- communication protocols: send, receive, wait, ...

# Automata in practice

Many systems can be modeled as automata:

- counters, displays, simple on/off control
- communication protocols: send, receive, wait, ...

Automata are a model for what can be calculated with finite memory (automata with finite number of states)

# Automata in practice

Many systems can be modeled as automata:

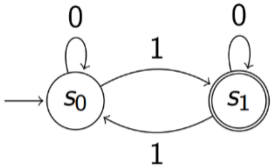
- counters, displays, simple on/off control
- communication protocols: send, receive, wait, ...

Automata are a model for what can be calculated with finite memory (automata with finite number of states)

Other problems with automata:

Testing with various input sequences:
Does it meet specification?

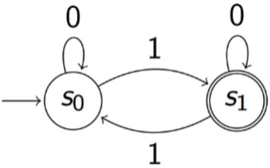
# A very simple automaton




starts in state s0
when it receives 1, it changes state
when it receives 0, it remains in place

After an even numbered string of 1, the machine will be in s0
After an odd numbered string of 1, the automaton will be in s1
⇒ the automaton can distinguish between the two kinds of strings

# A very simple automaton




starts in state s0
when it receives 1, it changes state
when it receives 0, it remains in place

After an even numbered string of 1, the machine will be in s0
After an odd numbered string of 1, the automaton will be in s1
⇒ the automaton can distinguish between the two kinds of strings

If we want an odd number of 1, we mark s1 as accepting state
string accepted: only if the machine is in accepting state at the end

⇒ the automaton defines a lot of strings, i.e. a language


Finite automata

**Languages**

Deterministic Finite Automata (DFA)

Non-deterministic Finite Automata (NFA)

Regular expressions

# What is a language

The alphabet is a lot of symbols (characters)

{a, b, c} or {0, 1} or {0, 1, ..., 9}, ...

With the symbols in the alphabet we can form strings (words, sequences):

aba, 010010, 437, ...

# What is a language

The alphabet is a lot of symbols (characters)

{a, b, c} or {0, 1} or {0, 1, ..., 9}, ...

With the symbols in the alphabet we can form strings (words, sequences):

aba, 010010, 437, ...

A language is a set of words (strings)

- like any explicitly defined set: {a, ab, ac, abc}
- or by a rule: strings of a, b, begin with a, more a than b

# What is a language (formal)

Let an alphabet Σ: a set of symbols (e.g. characters)

A finite word over the alphabet Σ is a string of symbols from Σ

$a1a2 \ldots an$ $ai \in \Sigma$ any number in any order

# What is a language (formal)

Let an alphabet Σ: a set of symbols (e.g. characters)

A finite word over the alphabet Σ is a string of symbols from Σ

a1a2 . . . an ai ∈ Σ                any number in any order

We denote by Σ∗ the set of all finite words over the alphabet Σ

$$\Sigma^* = \{a_1 a_2 \ldots a_n \mid a_i \in \Sigma\}$$

∗ Kleene star: repetition (zero or more occurrences)

contains empty string: zero repetition

Important: Σ∗ has words of unlimited length, but not infinite

# What is a language (formal)

Let an alphabet Σ: a set of symbols (e.g. characters)

A finite word over the alphabet Σ is a string of symbols from Σ

a1a2 . . . an ai ∈ Σ                    any number in any order

We denote by Σ∗ the set of all finite words over the alphabet Σ

$$\Sigma^* = \{a_1 a_2 \ldots a_n \mid a_i \in \Sigma\}$$

∗ Kleene star: repetition (zero or more occurrences)
contains empty string: zero repetition
Important: Σ∗ has words of unlimited length, but not infinite

A formal language L is a set of words L ⊆ Σ∗ , defined by certain rules: automata, regular expressions, grammars, etc.

Exemple: the language of strings of balanced parentheses; of palindromic strings; of strings of 0s and 1s that do not have three consecutive 0s; etc.


Finite automata
Languages
**Deterministic Finite Automata (DFA)**
Non-deterministic Finite Automata (NFA)
Regular expressions

# Deterministic Finite Automaton (DFA)

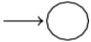
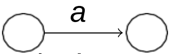
An automaton is given by:

- input symbols
- states
- transitions (from one state to another)
- initial state
- acceptor states (where we want to go)

# Deterministic Finite Automaton (DFA)

An automaton is given by: input symbols
states
transitions (from one state to another)
initial state
acceptor states (where we want to go)

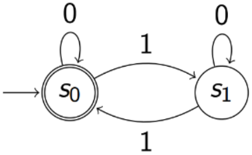
Formally, a finite automaton is a 5-element tuple $(\Sigma, S, s0, \delta, F)$

- $\Sigma$ is an unempty finite alphabet of input symbols $\{a, 0, 1, \ldots\}$
- $S$ is a finite non-empty set of states
- $s0 \in S$ is the initial state (one, in the usual definition) 
- $\delta : S \times \Sigma \rightarrow S$ is the transition function 

  deterministic: at any state and input, a single next state
- $F \subseteq S$ is the set of acceptor states 
  finally, we want to be here if the string is good (from the language)

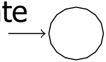
# Example of deterministic automaton (1)

parity automaton: accepts strings of 0 and 1 with even number of 1




or as a transitions table

| | 0 | 1 |
|---|---|---|
| $s_0$ | $s_0$ | $s_1$ |
| $s_1$ | $s_1$ | $s_0$ |

$s_0$ is the initial state  and accepting state at the same time 

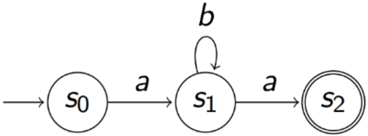
Acceptor states can have transitions:

here, from s0 exit when reading 1

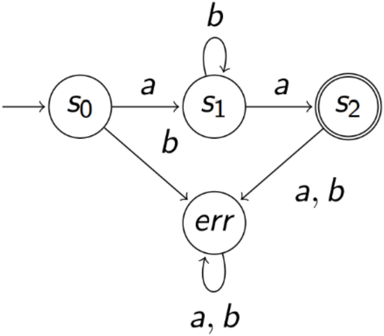
the state we reach when the string ends counts

# Example of deterministic automaton (2)

automaton that accepts words with any b (incl. 0) between two a




for δ to be defined everywhere another state err is needed in practice can be omitted




if from a state there is no transition the automaton is stuck, the string is no good

# Language supported by an automaton

We denote $\varepsilon \in \Sigma*$ the empty word (without any symbol).

We define a transition function $\delta* : S \times \Sigma* \rightarrow S$ with word entries:

In what state does the automaton reach for a given input word?

For any state $s \in S$, we define inductive:

$\delta^*(s, \varepsilon) = s$ empty word: do nothing

$\delta^*(s, a_1a_2 \ldots a_n) = \delta^*(\delta(s, a_1), a_2 \ldots a_n)$ for $n > 0$

In other words, $\delta^*(s_0, a_1a_2 \ldots a_n) = \delta^*(s_1, a_2 \ldots a_n)$, $s_1 = \delta(s_0, a_1)$

we obtain state s1 after input a1, and apply $\delta*$ on the remaining string

The automaton accepts the word $w \in \Sigma*$ if and only if $\delta*(s0, w) \in F$

(the word leads the automaton to an accepting state)

# How do we represent an automaton?

Matrix $S \times \Sigma$ with elements from S
(for each state and input, the next state)

explicitly represents each combination

| | a | b |
|---|---|---|
| $s_0$ | $s_0$ | $s_1$ |
| $s_1$ | $s_1$ | $s_0$ |

Or: a dictionary that gives for each state the transition function also represented as a dictionary (entry, state)

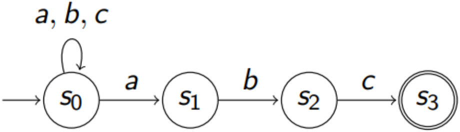
If from one state many symbols lead to the same next state, we associate each state:

- a dictionary (input, state)
- a default next state (for the other inputs)


Finite automata
Languages
Deterministic Finite Automata (DFA)
**Non-deterministic Finite Automata (NFA)**
Regular expressions

# Non-deterministic finite automata (NFA): Example (1)

Example: all strings of a, b, c ending in abc




From s0, receiving the symbol a, the automaton can:

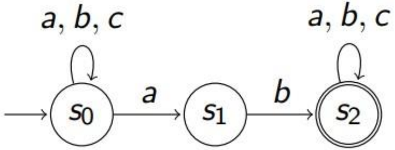
- remain in s0
- move to s1

⇒ the automaton can follow one of several paths

An NFA accepts if there is a choice leading to the accepting state.
If for a string ...abc we choose to pass into s1 at symbol a, the string will be accepted.

# Non-deterministic finite automata (NFA): Example (2)

All strings of a, b, c containing a substring ab




Once ab is found, the string is good, however the transitions continue from the accepting state to the accepting state.

**Advantages of NFA:**

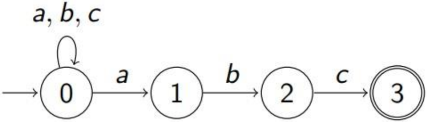
- sometimes easier to write than a deterministic automaton (we have to describe the acceptor path, not all the others)
- useful when specifying a system: we can leave several possibilities open, allows us a choice when implementing

# Deterministic and non-deterministic automata

Every non-deterministic automaton has an equivalent deterministic automaton (accepts the same strings).

We show how we do the conversion.

# NFA-DFA conversion (example)

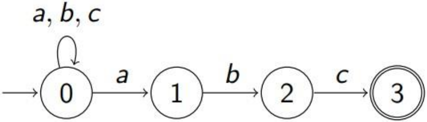



We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |

When we get a new set (red) we add a line to the table.

# NFA-DFA conversion (example)

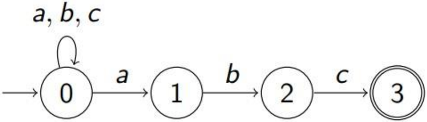



We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |
| {0, 1} | {0, 1} | {0, 2} | {0} |

When we get a new set (red) we add a line to the table.

# NFA-DFA conversion (example)

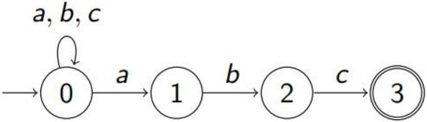



We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |
| {0, 1} | {0, 1} | {0, 2} | {0} |
| {0, 2} | {0, 1} | {0} | {0, 3} |

When we get a new set (red) we add a line to the table.

# NFA-DFA conversion (example)

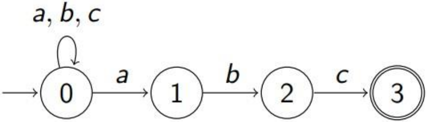



We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |
| {0, 1} | {0, 1} | {0, 2} | {0} |
| {0, 2} | {0, 1} | {0} | {0, 3} |
| {0, 3} | {0, 1} | {0} | {0} |

When we get a new set (red) we add a line to the table.

# NFA-DFA conversion (example)



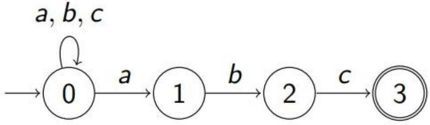

We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |
| {0, 1} | {0, 1} | {0, 2} | {0} |
| {0, 2} | {0, 1} | {0} | {0, 3} |
| {0, 3} | {0, 1} | {0} | {0} |

When we get a new set (red) we add a line to the table.

Each set obtained becomes a state in the resulting DFA

# NFA-DFA conversion (example)

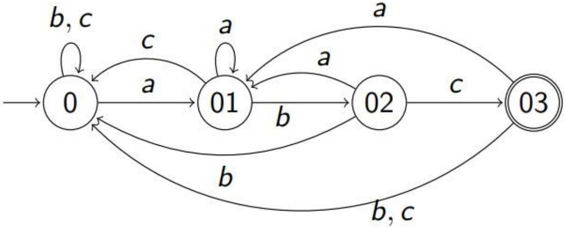



We write the transition table with the set of states in which each symbol is passed

| | a | b | c |
|---|---|---|---|
| {0} | {0, 1} | {0} | {0} |
| {0, 1} | {0, 1} | {0, 2} | {0} |
| {0, 2} | {0, 1} | {0} | {0, 3} |
| {0, 3} | {0, 1} | {0} | {0} |

When we get a new set (red) we add a line to the table.

Each set obtained becomes a state in the resulting DFA




The acceptor states 03 are those containing an acceptor state from the original automaton.

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

Σ = {a, d }

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| {1} | | |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

Σ = {a, d }

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| {1} | {2, 4} | {5} |
| {2, 4} | {1, 3, 5, 7} | {2, 4, 6, 8} |
| {5} | {2, 4, 6, 8} | {1, 3, 7, 9} |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |
| $\{5\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 7, 9\}$ |
| $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |
| $\{5\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 7, 9\}$ |
| $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |
| $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |
| $\{5\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 7, 9\}$ |
| $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |
| $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ |
| $\{1, 3, 7, 9\}$ | $\{2, 4, 6, 8\}$ | $\{5\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

accepted state: 9

$\Sigma$ = {a, d }

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |
| $\{5\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 7, 9\}$ |
| $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |
| $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ |
| $\{1, 3, 7, 9\}$ | $\{2, 4, 6, 8\}$ | $\{5\}$ |
| $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |

# Another example: moving by rule

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

initial state: 1

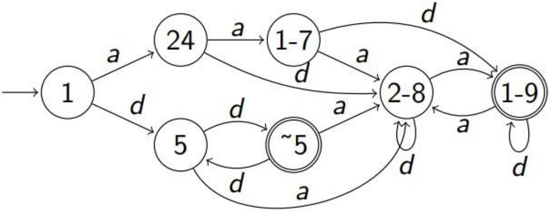
accepted state: 9

$\Sigma = \{a, d\}$

a: moves adjacent

d: moves diagonally

| | a | d |
|---|---|---|
| $\{1\}$ | $\{2, 4\}$ | $\{5\}$ |
| $\{2, 4\}$ | $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ |
| $\{5\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 7, 9\}$ |
| $\{1, 3, 5, 7\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |
| $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ |
| $\{1, 3, 7, 9\}$ | $\{2, 4, 6, 8\}$ | $\{5\}$ |
| $\{1, 3, 5, 7, 9\}$ | $\{2, 4, 6, 8\}$ | $\{1, 3, 5, 7, 9\}$ |




1-7 = $\{1, 3, 5, 7\}$

2-8 = $\{2, 4, 6, 8\}$

~5 = $\{1, 3, 7, 9\}$

1-9 = $\{1, 3, 5, 7, 9\}$


Finite automata
Languages
Deterministic Finite Automata (DFA)
Non-deterministic Finite Automata (NFA)
**Regular expressions**

# Can we express the definition of a language more concisely?

One language = a set of words over an alphabet

We are often interested in words with a simple, "regular" structure:

- an integer: a sequence of digits, possibly with a sign
- a real: integer part + decimal part (one of them optional), optional exponent
- an identifier: letters, digits, _ beginning with letter or _
- file names: 01-title.mp3, 02-title.mp3, ...

Some languages can be efficiently recognized by finite automata but writing automata takes effort
⇒ can be written more simply as regular expressions

# Regular expressions: formal definition

A regular expression describes a (regular) language.

A regular expression over an alphabet Σ is either:

3 base cases:

| | |
|---|---|
| $\emptyset$ | empty language |
| $\varepsilon$ | language {ε} (with empty string) |
| $a$ | language {a} with a ∈ Σ (a one-letter word) |

3 recursive cases: given e1, e2 regular expressions, we can form:

| | |
|---|---|
| $e_1 + e_2$ | reunion of languages<br>in practice, often denoted e1\|e2 (alternative, "or") |
| $e_1 \cdot e_2$ | language concatenation |
| $e_1{}^*$ | Kleene's closure of language |

# Writing rules and examples

Omit parentheses when clear from the precedence relationships most prior: $*$, then concatenation and then reunion + the dot for concatenation is omitted

In practice abbreviations are also used:

| | |
|---|---|
| e? | for e + ε (e, optional) |
| $e^+$ | for e$*$ \ ε (occurs at least once) |

$(0 + 1)*$ the set of all strings from 0 or 1

$(0 + 1)*0$ as above, ending with 0 (even numbers in binary)

$1(0 + 1)* + 0$ binary numbers, without unnecessary leading zeros

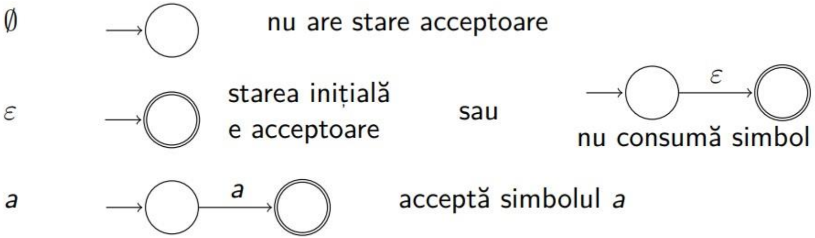
# Any regular expression is recognized by an automaton

Construction by Ken Thompson (creator of UNIX, 1983 Turing Award)

We define by structural induction:

- how we translate the 3 basic regular expression cases
- how we combine automata into the 3 recursive cases

⇒ by decomposing, we convert any regular expression into an automaton




in the three recursive cases, we combine the automata of the given languages

⇒ non-deterministic finite automaton with ε transitions (does not consume symbol)

# Important - Finite automata

A deterministic finite automaton defines an accepted language.
Such a language is called a regular language.
It can also be expressed by a regular expression.

The intersection, union, and complement of regular languages produce regular languages, as well as concatenation and Kleene closure. So they can be recognized by finite automata.

Non-deterministic finite automata can become deterministic

- so they still recognize regular languages
- but the number of states can increase exponentially.

Deterministic and non-deterministic automata and regular expressions have the same expressive power (they describe regular languages).


Thank you!

# Bibliography

The content of the course is based on the material from the LSD course taught by Prof. Dr. Eng. Marius Minea and S.l. Dr. Eng. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)