Lecture 1 – Deterministic Finite Automaton, Regular Languages, Pumping Lemma

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2025

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

About the course

The path to success

- Study and self-test regularly (ideally each week). Can you write down the definition/theorem/proof, fully and correctly?
- Make your own notes. The slides are not fully self-contained.
- Review after each lecture. Complete your understanding.
- Try to attend all lectures. If you miss one, catch up before the next. Use office hours and the textbooks as needed.
- Learn to work with the formalism, comfortably and precisely.
- Pay attention to the tutorial in a similar way.

Aims of the course

- get familiar with abstract models of computation
- be able to formally describe such models
- understand how minor changes can lead to huge difference in expressive power
- experience the unavoidability of undecidable problems
- prepare for NTIN090 Intro to Complexity and Computability
- also used in NSWI098 Compiler Principles, and in linguistics

Two levels of understanding: the idea behind a concept and the ability to formalize said concept

Resources

The course is mostly based on the following two textbooks:

- Hopcroft et al: "Introduction to Automata Theory, Languages, and Computation" (3rd edition) – an online copy and several physical copies are available in the library
- Sipser: "Introduction to the theory of computation" (3rd edition) – a physical copy is in the library

Introduction

Formal languages

A language L over an alphabet Σ is a set of words (finite strings) consisting of symbols (letters) from the alphabet.

Languages can represent:

- natural languages (words, well-formed sentences),
- programming languages (expressions, statements), document formats (XML,...)
- formal proofs
- possible strings of input or sensor readings of a machine, or
- decision problems, for example, $\Sigma = \{0, 1\}$ and

$$L = \{w \in \Sigma^\star \mid w \text{ encodes a CNF formula which is satisfiable}\}$$

Classifying languages

- Testing (membership of) words: how complex is the computing device needed? (automata)
- Generating words: how complex rules? (grammars)

Turing machines $\longleftrightarrow$ Formal grammars

Linear bounded automata $\longleftrightarrow$ Context-sensitive grammars

Pushdown automata $\longleftrightarrow$ Context-free grammars

Finite automata $\longleftrightarrow$ Regular grammars

NB: Almost all languages have no such finite representation.

A brief history

1852 first formalization of an ‘algorithm’ (Ada Lovelace)

1930s more focus following the development of computers

- limits of computation (what can and cannot a machine do?)
- computability theory
- Church, Turing, Kleene, Post

1943 neural networks

1956 finite automata (Kleene), to represent neural nets

1960s formal grammars (Chomsky), pushdown automata, formal language theory

1965 time and space complexity of algorithms

1971 P vs. NP, NP-completeness (Cook, Levin)

1972 natural NP-complete problems, polynomial reductions (Karp)

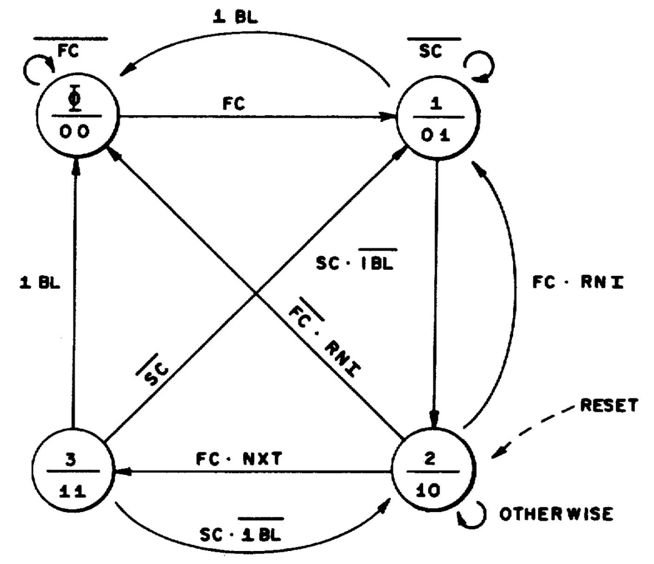
Automata and computability/complexity theory

- Automata are essential for the study of the limits of computation.
- Can a computer solve the task at all? decidability
(famous undecidable problems: Halting, Hilbert's 10th)
- Can a computer solve a task efficiently? tractability
(execution time as a function of input size)

Applications

- Natural language processing
- Compilers (lexical analyzer, syntax analyzer)
- Hardware design and verification (circuits, machines)
- Software and protocol design and verification
- Text search: regex
- Cellular automata (biology, AI)

Intel 8086 memory buffer (x86 architecture, 1978)

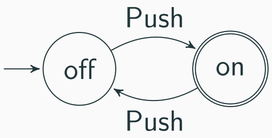


Chapter 1: Finite automata

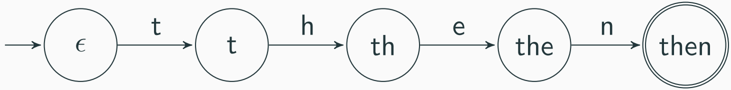
1.1 Deterministic Finite Automaton

Two simple examples

A finite automaton modeling an on/off switch.



A finite automaton modeling recognition of the word "then".



The definition

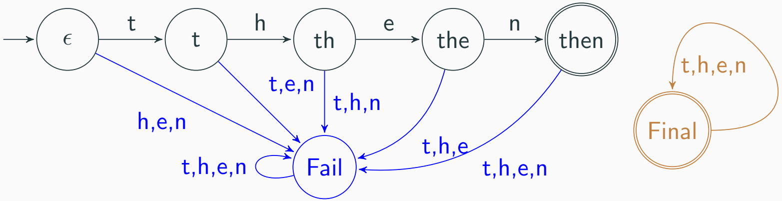
Definition (Deterministic Finite Automaton)

A deterministic finite automaton (DFA) is $A = (Q, \Sigma, \delta, q_0, F)$ consisting of:

- a finite nonempty set of states Q
- a finite nonempty set of input symbols Σ (the input alphabet)
- a transition function $\delta \colon Q \times \Sigma \to Q$
- an initial (start) state $q_0 \in Q$
- a set of accepting (final) states $F \subseteq Q$

Remarks

- Sometimes we allow δ to be a partial function. If needed, we can make δ total by adding a new "Fail" state and making it the target state for all missing transitions.
- Sometimes we require at least one accepting state. If $F = \emptyset$, we can add to F and Q a new "Final" state with no transitions from other states and $\delta(\text{"Final"}, s) = \text{"Final"}$ for all $s \in \Sigma$.

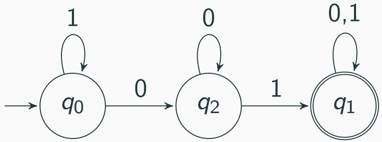


Representing a DFA

Example

A deterministic finite automaton A that accepts the language $L = \{u01v \mid u, v \in \{0,1\}^*\}$.

- the automaton: $A = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\})$
- a state diagram:



- a transition table:

δ	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

1.2 Regular Languages

Words and languages

- an alphabet Σ is a finite nonempty set of symbols (letters)
- a word w over Σ is a finite sequence of symbols from Σ
- that includes the empty word, denoted by ϵ (or sometimes λ)
- Σ^* denotes the set of all words over Σ, and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$
- a language $L \subseteq \Sigma^*$ is any set of words over the alphabet Σ

Note the difference: $L = \emptyset$ vs. $L' = \{\epsilon\}$.

Operations on words from Σ^*:

- concatenation: $u.v$ or uv
- powers: u^n ($u^0 = \epsilon$, $u^1 = u$, $u^{n+1} = u^n.u$)
- length: $|u|$ ($|\epsilon| = 0$, $|\text{banana}| = 6$)
- number of occurences of $s \in \Sigma$ in u: $|u|_s$ ($|banana|_a = 3$)

Extended transition function

Start in a state $q \in Q$ and read a ~~letter $a \in \Sigma$~~ word $w \in \Sigma^*$.

Definition (Extended transition function)

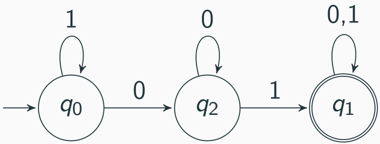
The extended transition function $\delta^* \colon Q \times \Sigma^* \to Q$:

- $\delta^*(q, \epsilon) = q$ for all $q \in Q$ (base case)
- $\delta^*(q, ua) = \delta(\delta^*(q, u), a)$ for $q \in Q, u \in \Sigma^*, a \in \Sigma$ (induction)

(We will sometimes write δ in place of δ^*.)

Example

$\delta^*(q_0, 1100) = q_2$, $\delta^*(q_0, 110011111111001) = q_1$



Language recognized by an automaton

Definition (Language of a DFA)

The language of the DFA A is:

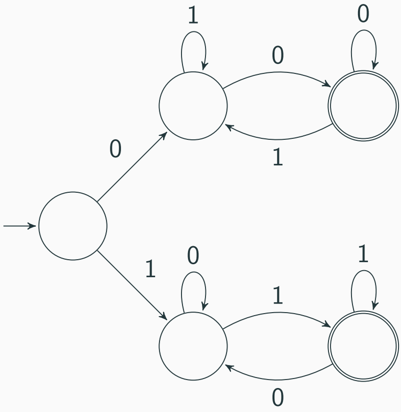
$$L(A) = \{w \mid \delta^*(q_0, w) \in F\}$$

- a word w is accepted [recognized] by A, if $w \in L(A)$
- a language L can be recognized by a DFA, if there exists a DFA A such that $L = L(A)$
- languages recognized by DFAs are called regular languages

Examples of regular languages 1/3

Example

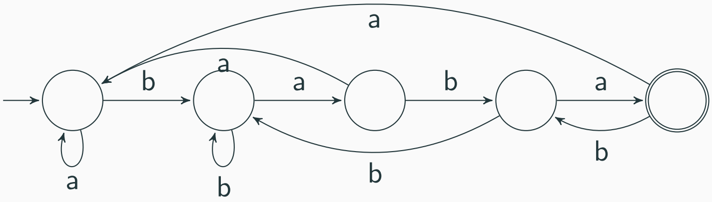
$L = \{w \in \{0,1\}^* \mid w = xux \text{ for some } x \in \{0,1\}, u \in \{0,1\}^*\}$



Examples of regular languages 2/3

Example

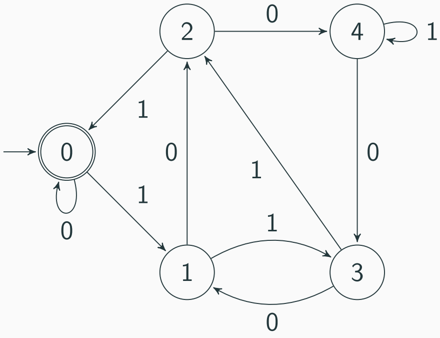
$L = \{w \mid w = ubaba, u \in \{a, b\}^*\}$



Examples of regular languages 3/3

Example

$L = \{w \in \{0,1\}^* \mid w$ is the binary encoding of a positive integer divisible by $5\}$



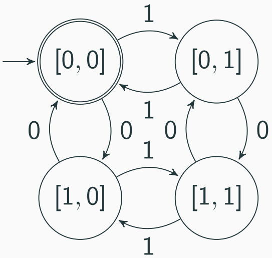
Exercise

Improve to disallow zeros at the beginning of nonzero numbers.

Product automaton

Example

$L = \{w \in \{0,1\}^* \mid |w|_0 = 2k \text{ and } |w|_1 = 2\ell \text{ for some } k, \ell \geq 0\}$.



δ	0	1
$* \rightarrow [0,0]$	$[1,0]$	$[0,1]$
$[0,1]$	$[1,1]$	$[0,0]$
$[1,0]$	$[0,0]$	$[1,1]$
$[1,1]$	$[0,1]$	$[1,0]$

Exercise

Formalize the construction of the product automaton.

Corollary

If L and L' are regular, then $L \cap L'$ is also regular.

1.3 Pumping Lemma

A language that is not regular

Example

$L = \{0^n1^n \mid n \geq 0\}$ is not regular.

- Intuition: the automaton cannot 'remember' arbitrarily large n using only finitely many states
- Formalization: Pumping lemma

Pumping lemma

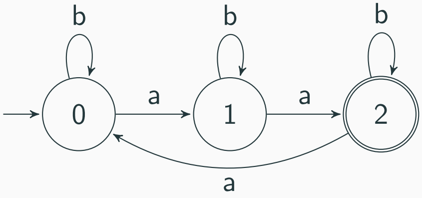
Theorem (Pumping Lemma For Regular Languages)

Let L be a regular language. Then there exists a constant $n \in \mathbb{N}$ (which depends on L) such that for every string $w \in L$ such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

- $y \neq \epsilon$.
- $|xy| \leq n$.
- *For all $k \geq 0$, the string xy^kz is also in L.*

Proof idea: The constant n is the number of states. Reading a word corresponds to a walk on the state diagram. Using the Pigeonhole principle, for long enough words we visit some state twice. The part of the walk between the first and second visit can be repeated (or skipped for $k = 0$).

Illustration: a regular language



- $abbbba = a(b)bbba$; for all $k \geq 0$ we have $a(b)^k bbba \in L(A)$
- $aaaaba = (aaa)aba$; for all $k \geq 0$ we have $(aaa)^i aba \in L(A)$
- aa cannot be pumped but it is too short: $|aa| < n = 3$

Proof of the Pumping lemma

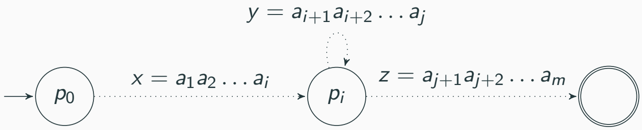
Suppose L is regular, then $L = L(A)$ for some DFA A with n states.

Take any word $w \in L$, $w = a_1 a_2 \ldots a_m$ of length $m \geq n$, $a_i \in \Sigma$.

Define $\forall i \ p_i = \delta^*(q_0, a_1 a_2 \ldots a_i)$. Note $p_0 = q_0$.

We have $n + 1$ p_i's and n states, therefore there are i, j such that $0 \leq i < j \leq n : p_i = p_j$.

Define: $x = a_1 a_2 \ldots a_i$, $y = a_{i+1} a_{i+2} \ldots a_j$, $z = a_{j+1} a_{j+2} \ldots a_m$.
Note $w = xyz$.



The loop above p_i can be repeated any number of times and the input is also accepted. $\square$

Application: proving nonregularity [an adversarial game]

Example

The language $L_{eq} = \{w; |w|_0 = |w|_1\}$ of all strings with an equal number of 0's and 1's is not a regular language.

Proof.

Suppose for contradiction that L_{eq} is regular. Take n from the pumping lemma.

- Pick $w = 0^n 1^n \in L_{eq}$.
- Break $w = xyz$ as in the pumping lemma, $y \neq \epsilon$, $|xy| \leq n$.
- Since $|xy| \leq n$ and it's at the beginning of w, it has only 0's.
- The pumping lemma says: $xz \in L_{eq}$ (for $k = 0$). However, it has less 0's and the same # of 1's as w so it's not in L_{eq}. $\square$

Example

The language $L = \{0^i 1^i; i \geq 0\}$ is not regular. (Same proof as the previous example.)

Example

The language L_{pr} of all prime-length strings of 1's is not regular.

Proof.

Suppose it were. Take the constant n from the pumping lemma.

- Consider some prime $p \geq n + 2$, let $w = 1^p$.
- Break $w = xyz$ by the PL, let $|y| = m$. Then $|xz| = p - m$.
- By the PL, $xy^{p-m}z \in L_{pr}$. But $|xy^{p-m}z| = |xz| + (p-m)|y| = p - m + (p-m)m = (m+1)(p-m)$ which is not a prime (none of two factors are 1). $\square$

Not a characterization of regular languages!

The Pumping Lemma is not a characterization of regular languages. (It is only an implication, not an equivalence.)

Example (Nonregular language that can be 'pumped')

The language $L = \{u \in \{a, b, c\}^* \mid u = a^+ b^i c^i \text{ or } u = b^i c^j\}$ is not regular but the first symbol can be always pumped.

(a^+ means at least one a, notation from regular expressions)

Why? We can use the Myhill–Nerode theorem (later) which is a characterization or alternatively a 'Pumping Lemma with pumping near the end'.

Exercise

State and prove a pumping lemma with pumping near the end.

Pumping lemma and finiteness

Theorem

A regular language L is infinite if and only if there exists $u \in L; n \leq |u| < 2n$, where n is the constant from the PL.

Proof.

$\Leftarrow$ If $(\exists u \in L)\ n \leq |u| < 2n$, then we can pump: split $u = xyz$, $xy^iz \in L$ for all $i \in \mathbb{N}$. That gives us infinitely many words in L.
$\Rightarrow$ If L is infinite, then it must contain a word w with $n \leq |w|$. If $|w| < 2n$, then we are done. Otherwise, using the PL: $w = xyz$ and $xy^0z = xz \in L$, and we get a shorter word. If $2n \leq |xz|$, we shorten xz further (PL again). Each time we cut $\leq n$ symbols; thus we hit the interval $[n, 2n)$. $\square$

Corollary

To check if a regular language is infinite it is sufficient to check a finite number of strings: $\{u \mid n \leq |u| < 2n\}$.

Summary of Lecture 1

- Deterministic Finite Automaton (DFA): $A = (Q, \Sigma, \delta, q_0, F)$
- Extended transition function δ^*
- The language recognized by the DFA A is the language

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

- Languages recognized by some DFA are called regular
- Finite automata encode only finite information, but can recognize infinite languages
- Product automaton, intersection of reg. languages is regular
- Pumping lemma for regular languages (prove nonregularity)
- PL not a characterization, some nonregular can be pumped
- A regular language is infinite iff it contains a word of length $n \leq |w| \leq 2n$ where n is #states of a recognizing automaton

Appendix: A toy application

Example of a (bad) electronic-money protocol

- Three parties: the customer, the store, the bank.
- Only one 'money' file (for simplicity).
- Customer may decide to transfer money to the store, which will the redeem the file from the bank and ship goods to the customer. The customer has the option to cancel the file.
- Five events:
  - Customer may **pay**.
  - Customer may **cancel**.
  - Store may **ship** the goods to the customer.
  - Store may **redeem** the money.
  - The bank may **transfer** the money by creating a new, suitably encrypted money file and sending it to the store.

[Hopcroft et al: Introduction to automata theory, languages, and computation]

(Incomplete) finite automata for the bank example

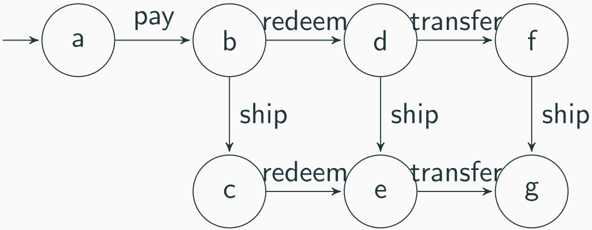


Figure 1: Store

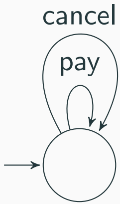


Figure 2: Customer

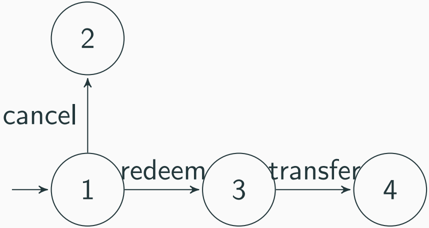


Figure 3: Bank

An edge for each Input

- Formally, the automaton should perform an action for any input. The store automaton needs an additional arc from each state to itself, labeled *cancel*.
- Customer mustn't kill the automaton by executing *pay* again, so loops labelled *pay* are necessary. Similarly other commands.

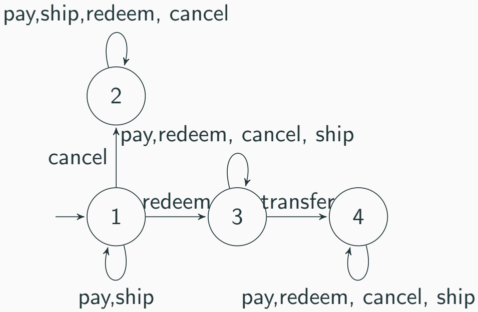


Figure 4: Extended Automaton for the Bank

Product automaton

The states of the product automaton of Bank and Store are pairs $B \times S$. To construct the arc of the product automaton, we need to run the bank and store automata 'in parallel'. If any automaton dies, the product dies too.