Lecture 1 – Deterministic Finite Automaton, Regular Languages, Myhill-Nerode Theorem

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2026

** 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.

How to study?

I highly recommend this minicourse on effective learning:

https://www.samford.edu/departments/academic-success-center/how-to-study

Invest 35 minutes now, save many hours later!

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
- a brief introduction to complexity theory (P, NP, and friends)
- 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.

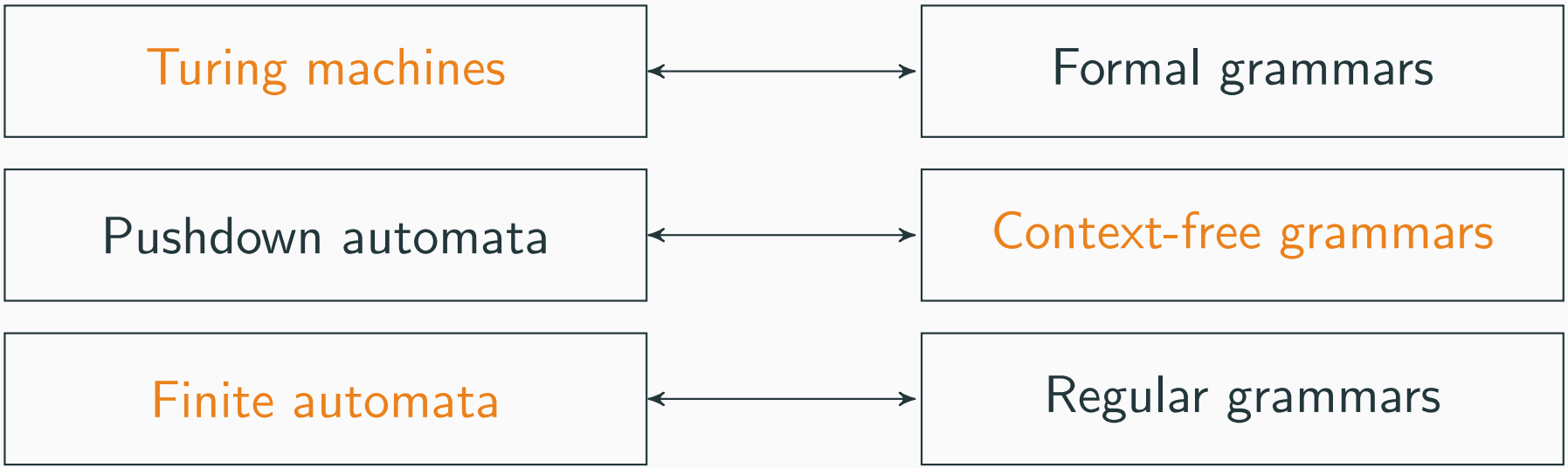
Languages can represent:

- 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)



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)

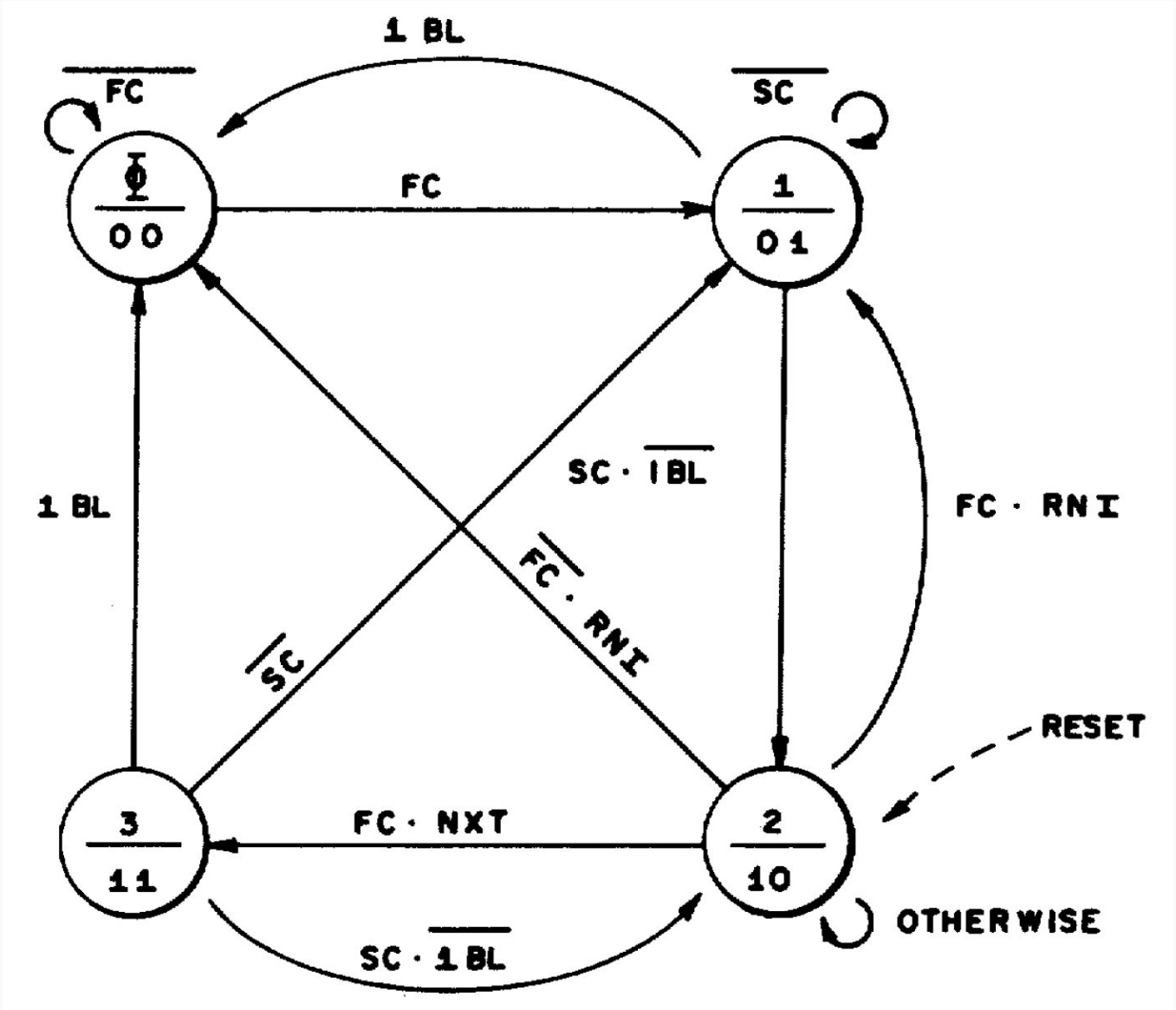
Automata and computability/complexity theory

- 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)

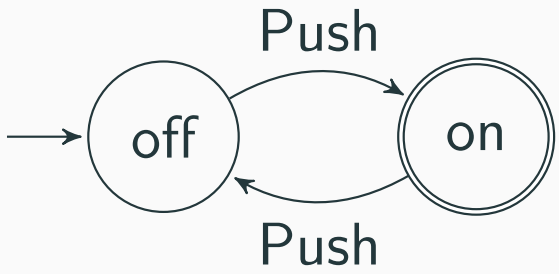


Chapter 1: Finite automata

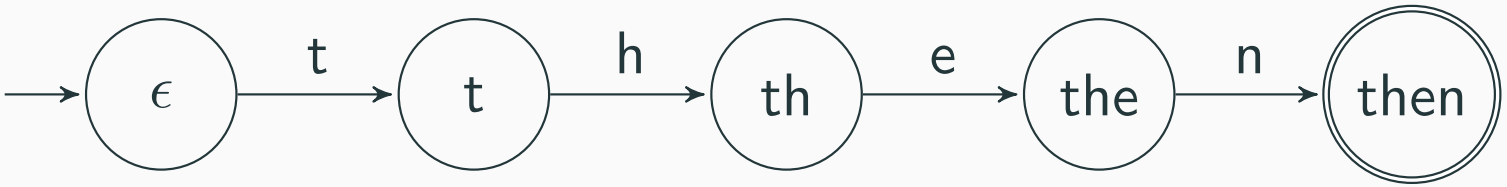
1.1 Deterministic Finite Automaton

Two simple examples

A finite automaton modeling an on/off switch.



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



The definition

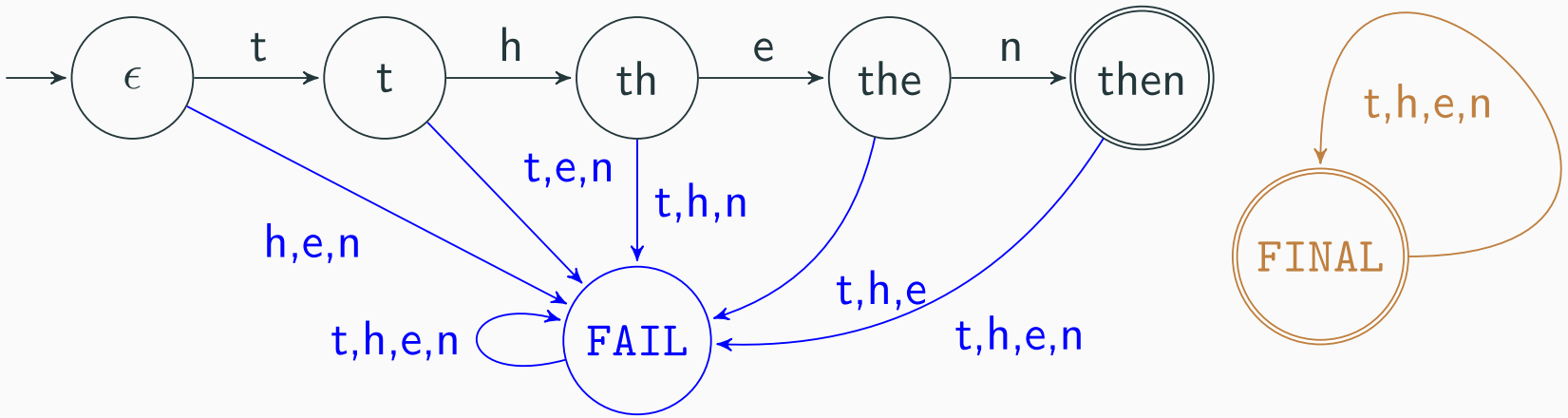
Definition (Deterministic Finite Automaton)

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

- Some people allow δ to be a partial function. We don't. They can always make δ total by adding a new `FAIL` state and making it the target state for all missing transitions.
- Some people require at least one accepting state. We don't. If $F = \emptyset$, they can add to F and Q a new `FINAL` state with no transitions from other states and $\delta(\texttt{FINAL}, s) = \texttt{FINAL}$ for all $s \in \Sigma$.

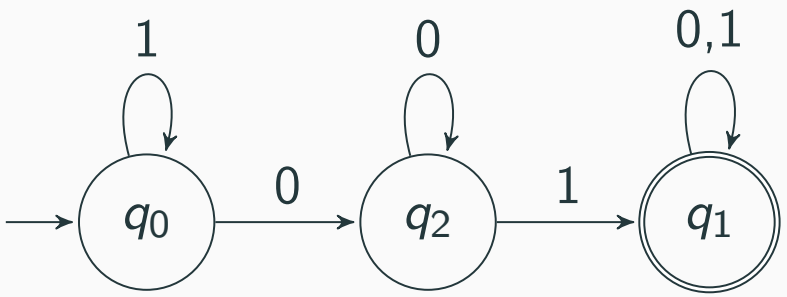


Representing a DFA

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 ϵ (some people use 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)

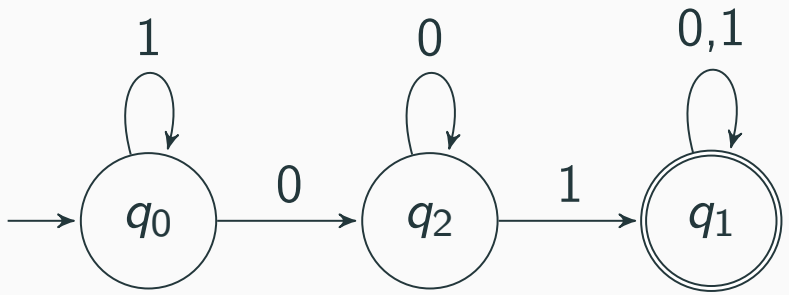
The extended transition function $\delta^* : Q \times \Sigma^* \to Q$:

- $\delta^*(q, \epsilon) = q$ 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

$\delta^*(q_0, 1100) = q_2$, $\delta^*(q_0, 1100111111111001) = q_1$



Language recognized by an automaton

Definition (Language of a DFA)

The language of the DFA A is:

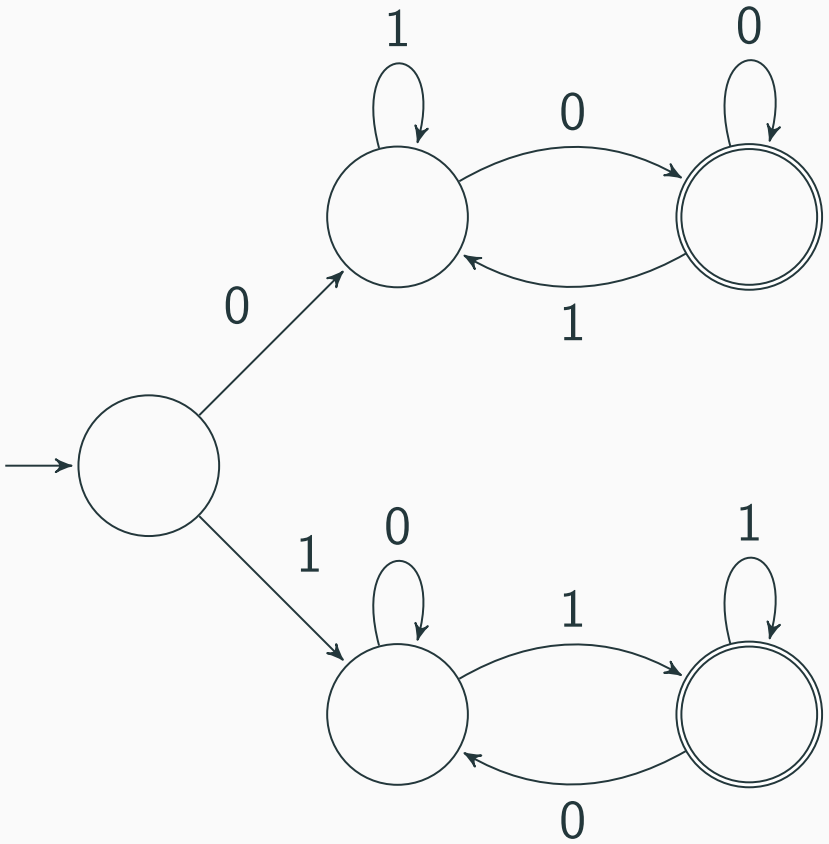
$$L(A) = \{w \mid \delta^*(q_0, w) \in F\}$$

- 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

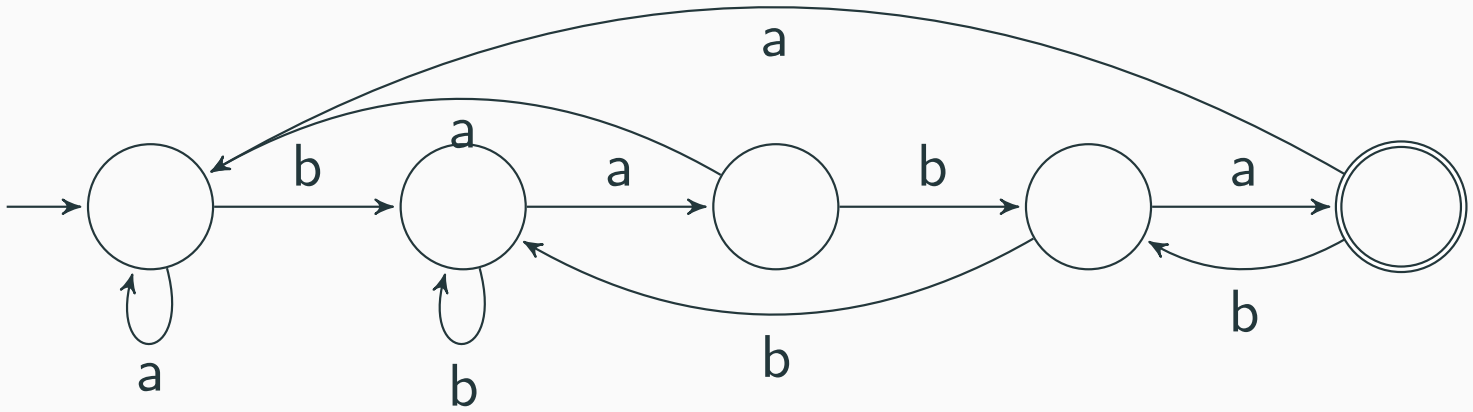
$L = \{w \in \{0,1\}^* \mid w = xux \text{ for some } x \in \{0,1\}, u \in \{0,1\}^*\}$



Examples of regular languages 2/3

Example

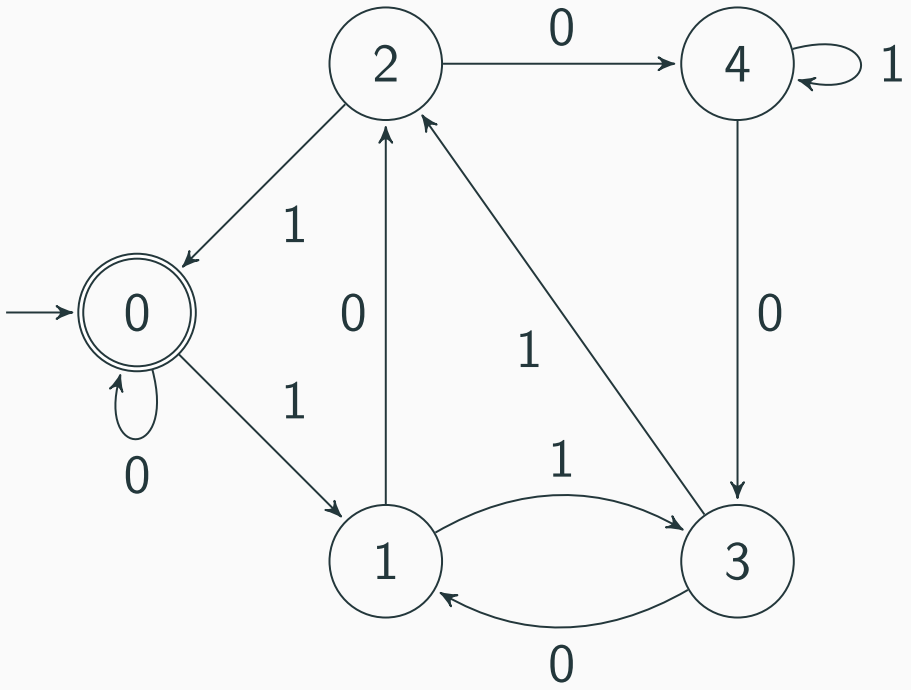
$L = \{w \mid w = ubaba, u \in \{a, b\}^*\}$



Examples of regular languages 3/3

Example

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



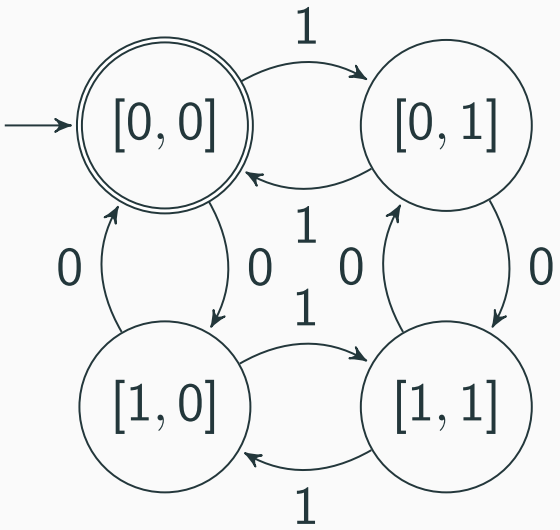
Exercise

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
$* \to [0,0]$	$[1,0]$	$[0,1]$
$[0,1]$	$[1,1]$	$[0,0]$
$[1,0]$	$[0,0]$	$[1,1]$
$[1,1]$	$[0,1]$	$[1,0]$

Exercise

Formalize the construction of the product automaton.

Corollary

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

1.3 Myhill–Nerode Theorem

A language that is not regular

Example

$L = \{0^n 1^n \mid n \geq 0\}$ is not regular.

- Intuition: the automaton cannot 'remember' arbitrarily large n using only finitely many states
- Formalization: Myhill-Nerode theorem, or Pumping lemma (later)

Characterize regular languages

How to recognize whether a given language is regular?

If it is, we can build a DFA. But how? And what if it is not regular? How can we prove that?

We would like to have a characterization.

Luckily, as we'll see, every regular language comes with an implicit automaton 'hiding' in the set of all words over its alphabet.

Congruences on words

Let Σ be a finite alphabet and $\sim$ an equivalence relation on Σ^* (reflexive, symmetric, transitive). Then:

- $\sim$ is a right congruence iff $(\forall u, v, w \in \Sigma^*) u \sim v \Rightarrow uw \sim vw$.
- $\sim$ has finite index iff the partition $\Sigma^*/\sim$ has a finite number of classes.
- the class containing a word u is denoted $[u]_\sim$ or simply $[u]$

Example

- $\sim_{end}$ 'end with the same letter': right congruence of finite index,
- $\sim_{\#a}$ 'same number of a's': right congruence but not of finite index

The theorem

Theorem (Myhill–Nerode theorem)

Let Σ be a finite alphabet and $L \subset \Sigma^*$ a language over Σ. The following statements are equivalent:

(i) *L is regular,*

(ii) *there exists a right congruence $\sim$ on Σ^* with finite index such that L is a union of some classes of the partition $\Sigma^*/\sim$.*

Proof idea: Group together words that end in the same state when we start reading from q_0.

The proof

(i)⇒(ii) from an automaton to a right congruence of finite index

- we define $u \sim v \equiv \delta^*(q_0, u) = \delta^*(q_0, v)$
- it is indeed a right congruence (from the definition of δ^*)
- it has a finite index (Q is finite)
- $L = \{w \mid \delta^*(q_0, w) \in F\} = \bigcup_{q \in F} \{w \mid \delta^*(q_0, w) = q\}$
  $= \bigcup_{q \in F} [w \mid \delta^*(q_0, w) = q]_\sim$.

(ii)⇒(i) from a right congruence of finite index to an automaton

- the alphabet is Σ, states Q are the congruence classes $\Sigma^*/\sim$
- the initial state $q_0 = [\epsilon]$, final states $F = \{c_1, \ldots, c_n\}$ where $L = \bigcup_{i=1,\ldots,n} c_i$
- trans. function $\delta([u], x) = [ux]$ (well-defined right congruence)
- to show that $L(A) = L$, using $\delta^*([\epsilon], w) = [w]$:
  $w \in L \Leftrightarrow w \in \bigcup_{i=1,\ldots,n} c_i \Leftrightarrow w \in c_1 \vee \ldots w \in c_n \Leftrightarrow$
  $[w] = c_1 \vee \ldots [w] = c_n \Leftrightarrow [w] \in F \Leftrightarrow w \in L(A)$ □

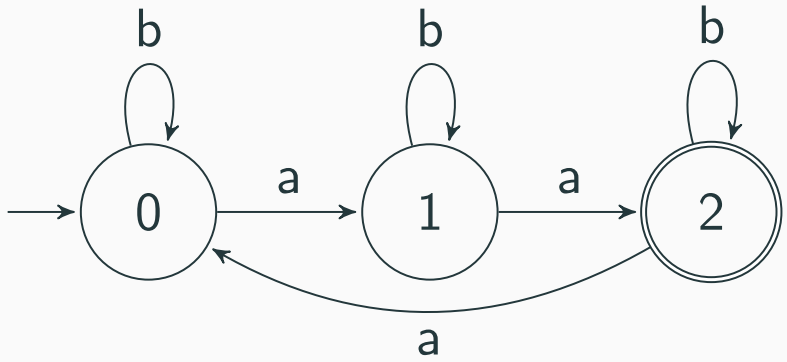
Application: proof of regularity

Example

Construct an automaton that recognizes the language $L = \{w \in \{a, b\}^* \mid |w|_a = 3k + 2 \text{ for some } k \geq 0\}$.

$$u \sim v \text{ iff } |u|_a \equiv |v|_a \pmod 3$$

- equivalence classes are 0,1,2
- L corresponds to the class 2
- a – transitions to the next class
- b – stay in the same class.



Application: proof of nonregularity

Example

Show that $L = \{u \in \{a, b, c\}^* \mid u = a^+ b^i c^i \text{ or } u = b^i c^j\}$ is not regular. (Note that the first letter can be pumped.)

Suppose for contradiction that L is regular. Let $\sim_L$ be a right congruence of finite index where L is a union of some $\sim_L$-classes.

Consider the set of words $S = \{a, ab, abb, \ldots\} = \{ab^n \mid n \in \mathbb{N}\}$.

For any two $i \neq j$ there is a string (c^i) distinguishing the words (in/out of the language): $ab^i c^i \in L$ but $ab^j c^i \notin L$

No two elements of S can be in the same class of $\sim_L$ (L would split the class). Since S is infinite, this contradicts finite index of $\sim_L$. $\square$

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
- Myhill–Nerode theorem (DFAs $\leftrightsquigarrow$ right congruences of Σ^* of finite index where L is a union of classes)

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

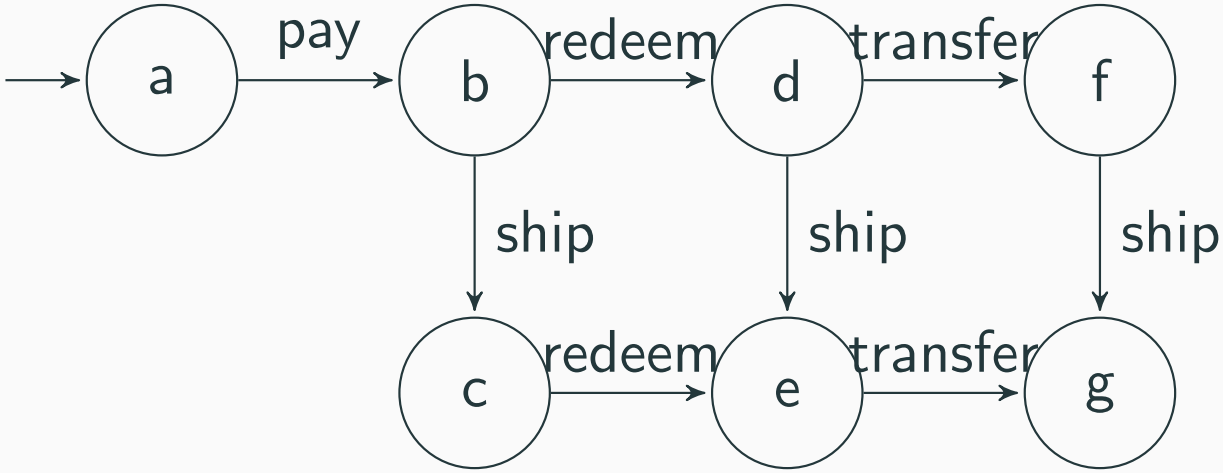


Figure 1: Store

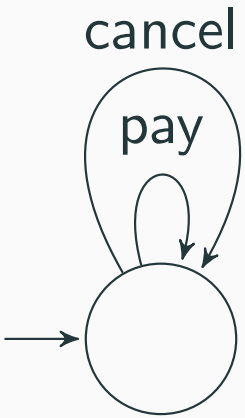


Figure 2: Customer

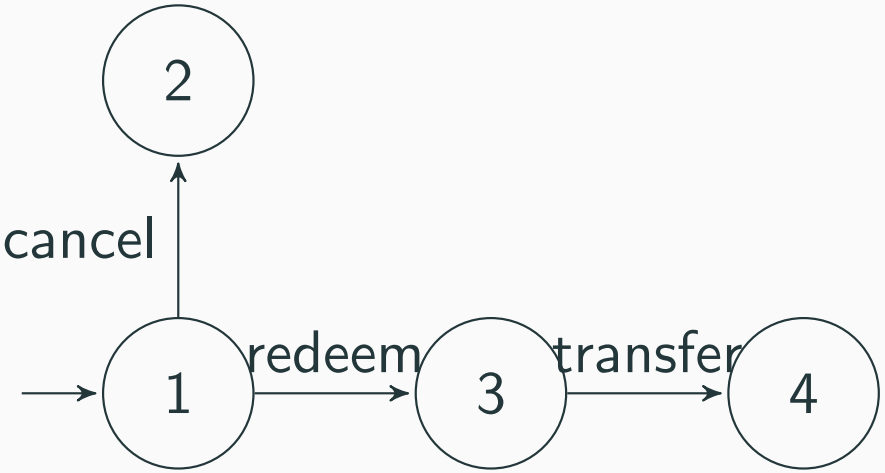


Figure 3: Bank

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.

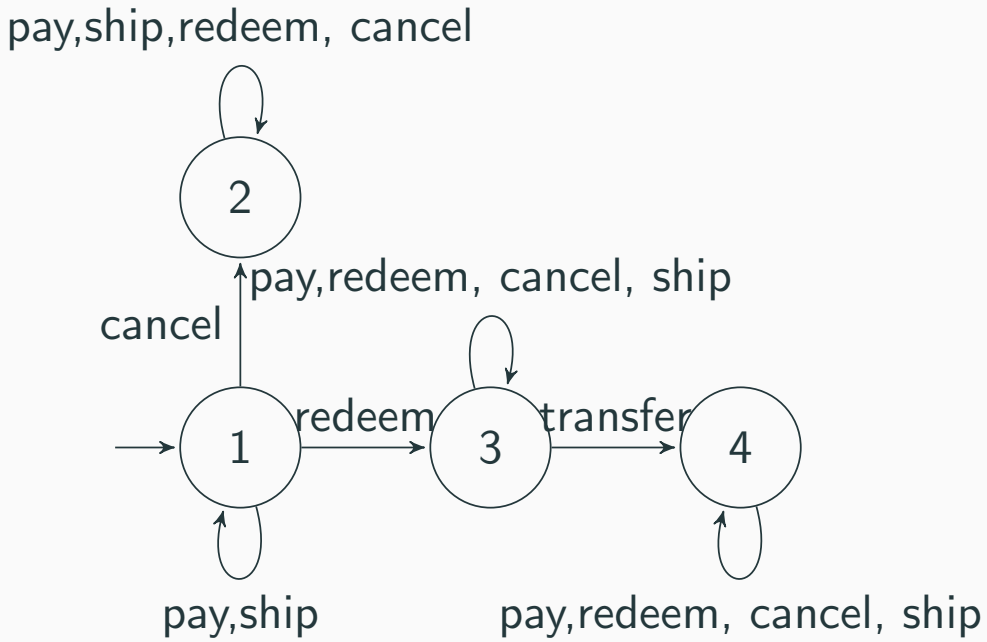


Figure 4: Extended Automaton for the Bank

Product automaton for the example

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.