Introduction, Myhill-Nerode theorem

Jakub Bulín

Adapted from the Czech lecture slides by Marta Vomlelová.
Translation, minor modifications, and any errors are mine.

KTIML MFF UK

Organizational matters

- Course web:

  https://jbulin.github.io/teaching/spring/ntin071/
- Office hours (in S303): as scheduled or make an appointment by email
- Information about exams:

  https://jbulin.github.io/teaching/spring/ntin071/about-exams.pdf
- My tutorial class:

  https://jbulin.github.io/teaching/spring/ntin071/tutorial/
- If you are enrolled in Dr. Kuboň's tutorial class, refer to him for any questions
- Credit from the tutorial is required to take the exam

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

see the course webpage for additional resources

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 lecture notes 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

Formal languages

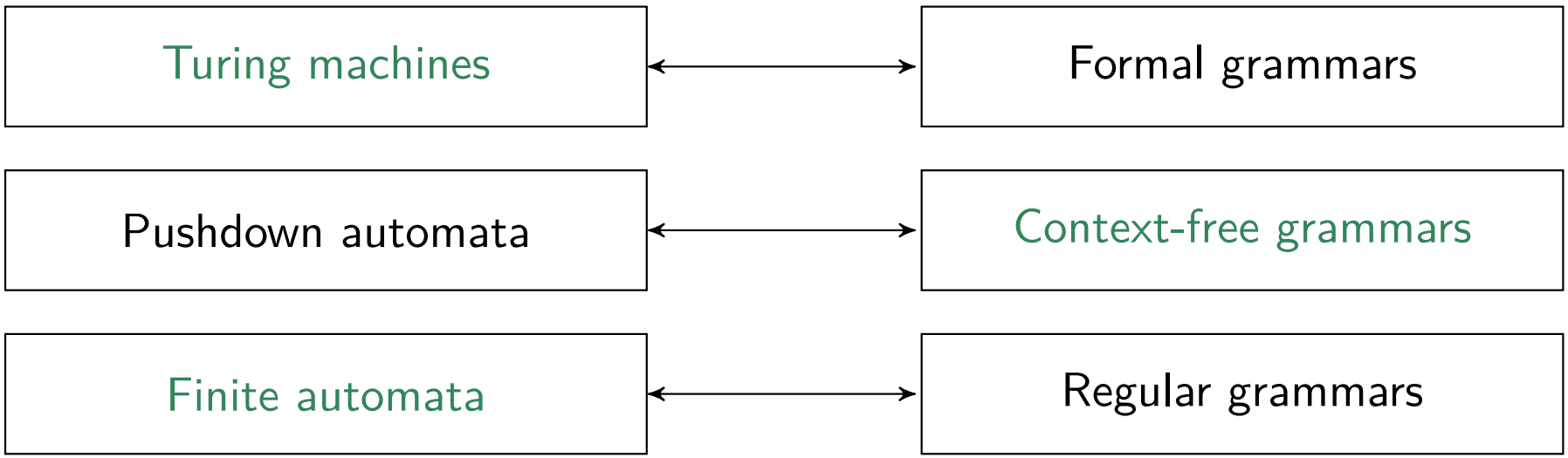
A language L over an alphabet Σ is a set of words (finite strings) consisting of symbols (letters) from the alphabet.
Languages can represent:

- 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.

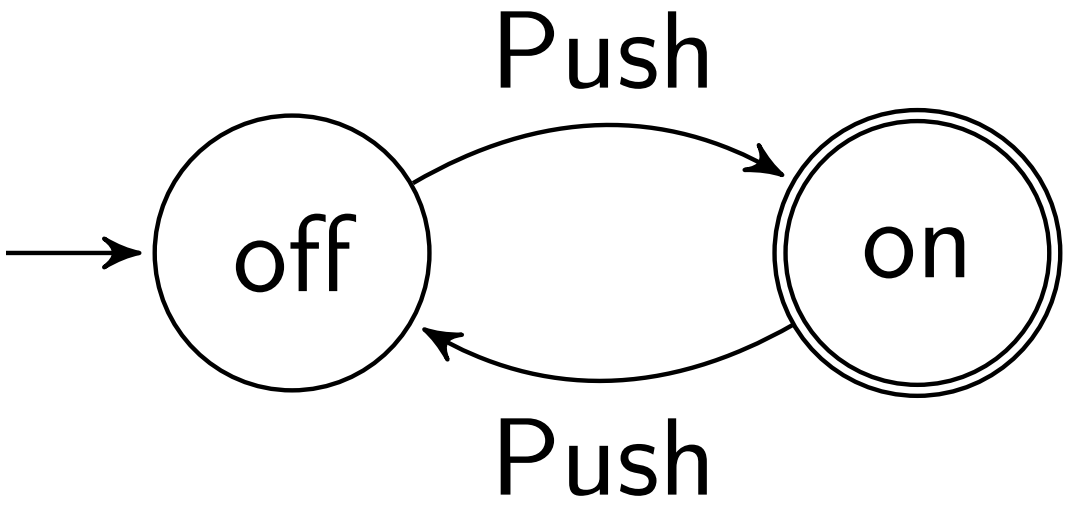
Automata and computability/complexity theory

- 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)

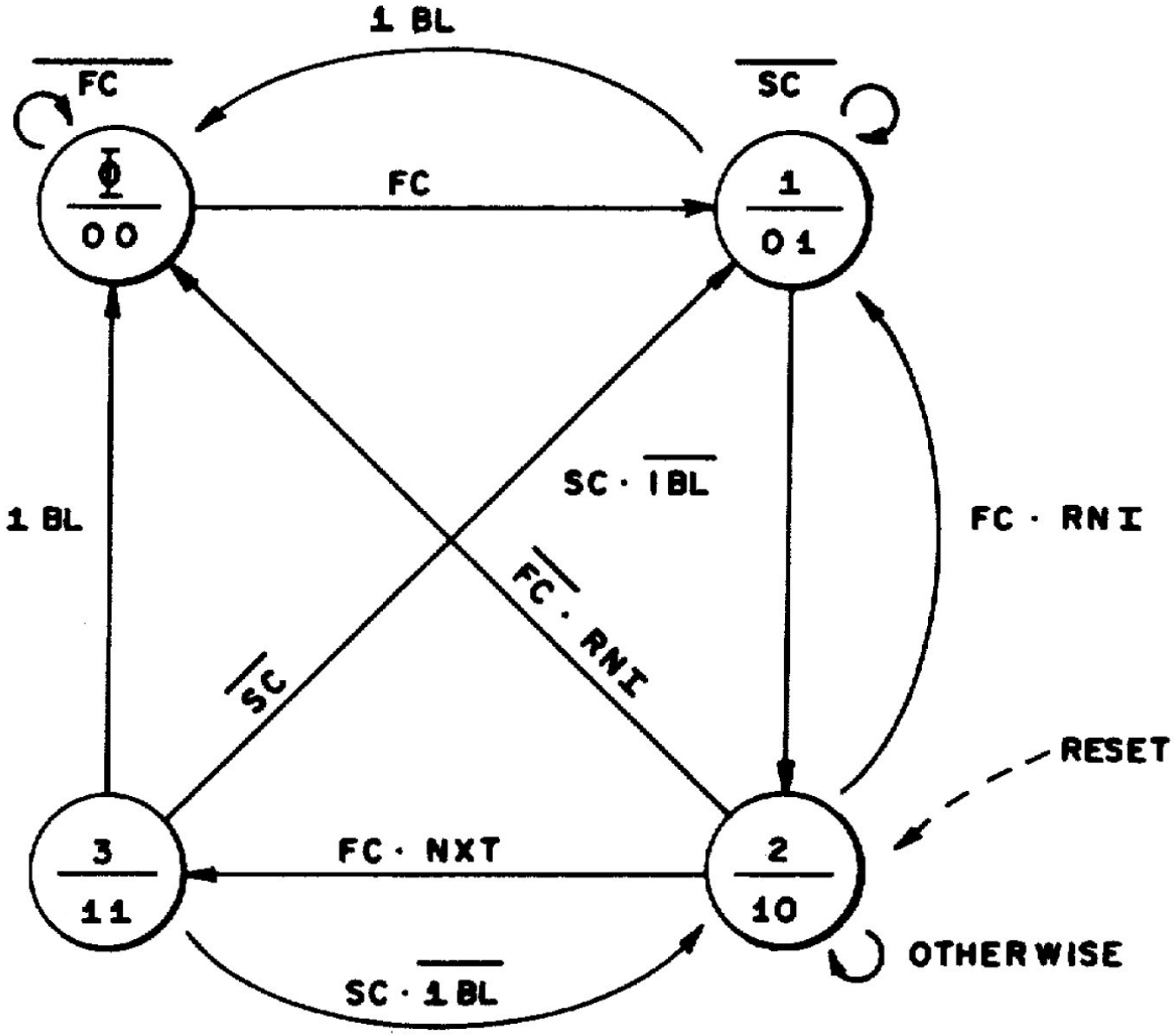
Applications

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

Hardware design: on/off switch



Intel 8086 memory buffer (x86 architecture, 1978)



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)

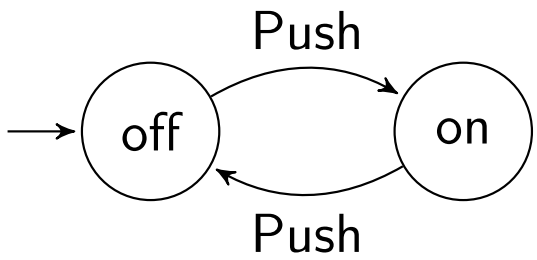
1972 natural NP-complete problems, polynomial reductions (Karp)

Practical applications

- Reflection on the correctness of a program, algorithm, compiler,
- natural language processing,
- compilers:
  - lexical analysis,
  - syntactic analysis,
- design, description, verification of hardware
  - integrated circuits
  - machines
  - automata
- implementation using software
  - searching for occurrences of a word in a text (grep)
  - verification of finite-state systems.

- Design and verification of integrated circuits.

Finite automaton modeling an on/off switch.



- Lexical analysis

Finite automaton recognizing the word `then`.