
CMP 309: Automata Theory, Computability and Formal Languages

Adapted from the work of Andrej Bogdanov

Course outline

- Introduction to Automata Theory
- Finite Automata
 - Deterministic Finite state automata
 - Nondeterministic finite state automata
- Regular Expressions
- Properties of regular languages
- Limitations of finite automata
- DFA minimization and DFA Equivalence
- Context-free languages
- Normal forms and parsing
- Pushdown automata
- Limitations of context-free languages
- Parsers for programming languages
- Turing machines

Course resources

- Course website:
 - www.folajimiclass.weebly.com/automata.html
- Course online registration:
 - www.folajimiclass.weebly.com/register.html
- Lecture materials:
 - Lecture slides (downloadable on course site)
 - Course text:
 - Introduction To Automata Theory Languages , and Computation by J. Hopcroft, R. Motwani and J. D. Ullman (Chapters 1-8)
 - Wikipedia resources:
 - http://en.wikipedia.org/wiki/Automata_theory

Course resources

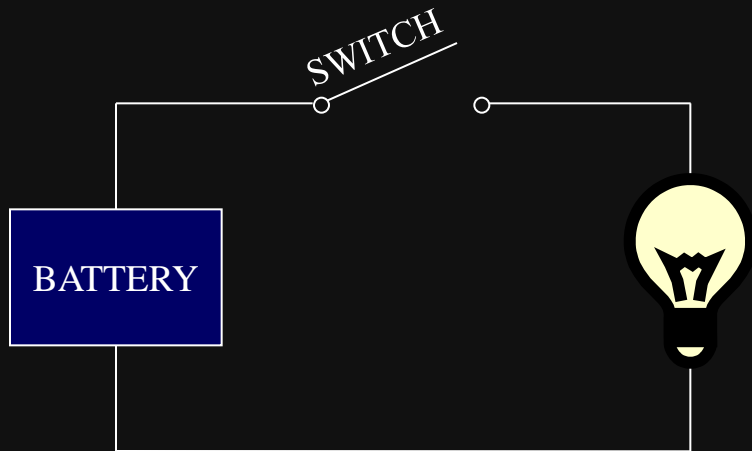
- Software:
 - JFLAP: software for experimenting with formal languages topics
 - Downloadable at www.jflap.org/jflaptmp/
 - **Visual Automata Simulator:** A tool for simulating, visualizing and transforming finite state automata and Turing Machines
 - Downloadable at <http://www.cs.usfca.edu/~jbovet/vas.html>

Introduction

What is automata theory

- Automata theory is the study of **abstract computational devices**
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, ...
- Why do we need abstract models?

A simple computer



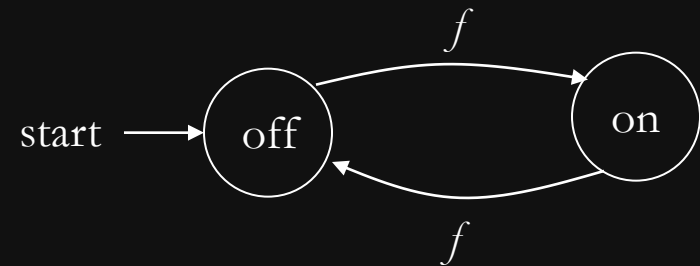
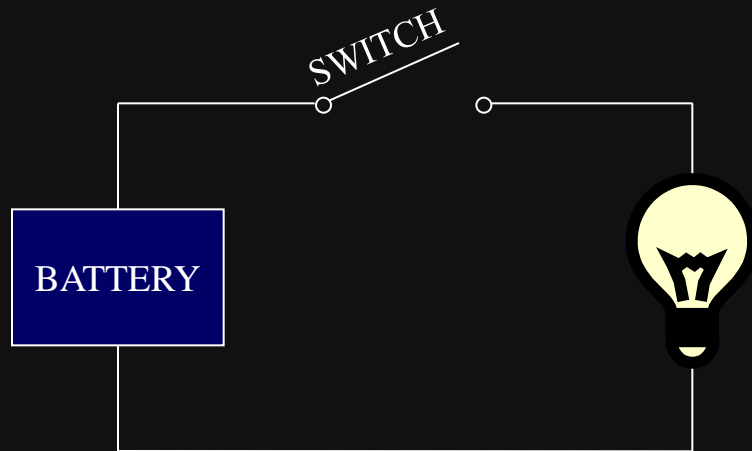
input: switch

output: light bulb

actions: flip switch

states: on, off

A simple “computer”



input: switch

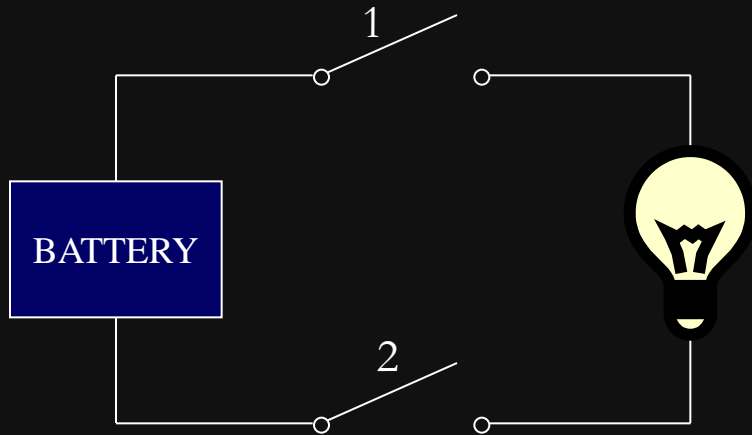
output: light bulb

actions: f for “flip switch”

states: on, off

bulb is on if and only if there was an **odd** number of flips

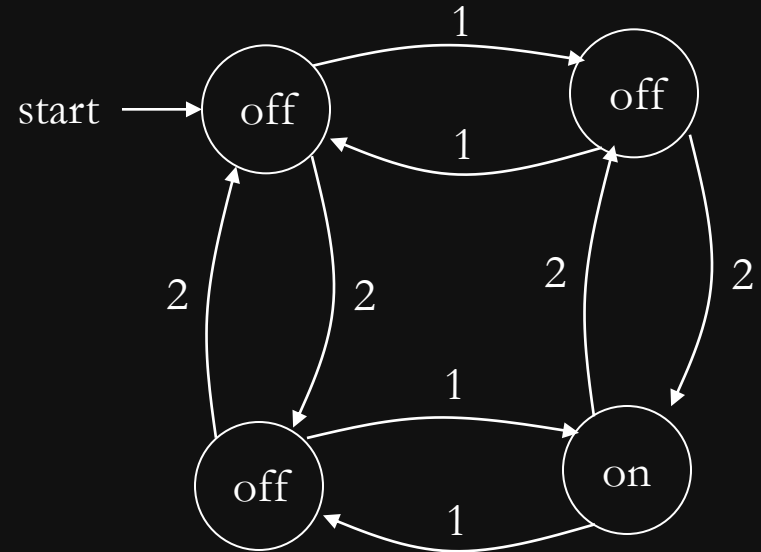
Another “computer”



inputs: switches 1 and 2

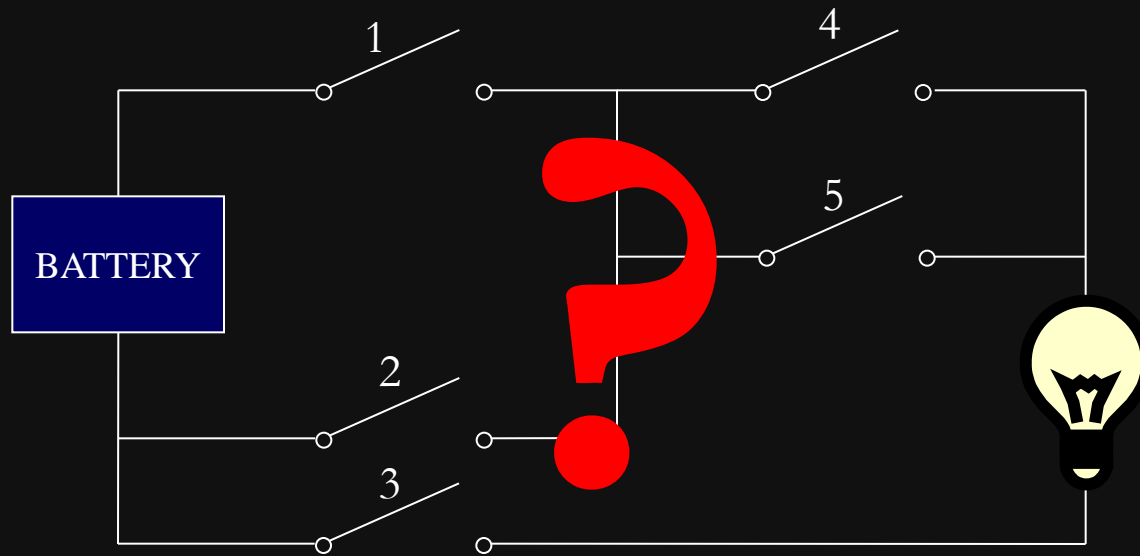
actions: 1 for “flip switch 1”
2 for “flip switch 2”

states: on, off



bulb is on if and only if
both switches were flipped
an **odd** number of times

A design problem



Can you design a circuit where the light is on if and only if all the switches were flipped **exactly the same number of times**?

A design problem

- Such devices are difficult to reason about, because they can be designed in an infinite number of ways
- By representing them as abstract computational devices, or **automata**, we will learn how to answer such questions

These devices can model many things

- They can describe the operation of any “small computer”, like the control component of an alarm clock or a microwave
- They are also used in **lexical analyzers** to recognize well formed expressions in programming languages:

ab1 is a legal name of a variable in C
5u= is not

Different kinds of automata

- This was only one example of a computational device, and there are others
- We will look at different devices, and look at the following questions:
 - What can a given type of device compute, and what are its limitations?
 - Is one type of device more powerful than another?

Some devices we will see

finite automata

Devices with a finite amount of memory.
Used to model “small” computers.

push-down
automata

Devices with infinite memory that can be
accessed in a restricted way.

Used to model parsers, etc.

Turing Machines

Devices with infinite memory.

Used to model any computer.

time-bounded
Turing Machines

Infinite memory, but bounded running time.

Used to model any computer program that
runs in a “reasonable” amount of time.

Preliminaries of automata theory

- How do we formalize the question

Can device A solve problem B?

- First, we need a formal way of describing the problems that we are interested in solving

Problems

- Examples of problems we will consider
 - Given a **word** s , does it contain the subword “fool”?
 - Given a **number** n , is it divisible by 7?
 - Given a **pair of words** s and t , are they the same?
 - Given an expression with brackets, e.g. $(() ())$, does every left bracket match with a subsequent right bracket?
- All of these have “yes/no” answers.
- There are other types of problems, that ask “**Find this**” or “**How many of that**” but we won’t look at those.

Alphabets and strings

- A common way to talk about words, number, pairs of words, etc. is by representing them as **strings**
- To define strings, we start with an **alphabet**

An **alphabet** is a finite set of symbols.

- **Examples**

$\Sigma_1 = \{a, b, c, d, \dots, z\}$: the set of letters in English

$\Sigma_2 = \{0, 1, \dots, 9\}$: the set of (base 10) digits

$\Sigma_3 = \{a, b, \dots, z, \#\}$: the set of letters plus the special symbol #

$\Sigma_4 = \{ (,) \}$: the set of open and closed brackets

Strings

A string over alphabet Σ is a finite sequence of symbols in Σ .

- The empty string will be denoted by ε
- Examples

abfbz is a string over $\Sigma_1 = \{a, b, c, d, \dots, z\}$

9021 is a string over $\Sigma_2 = \{0, 1, \dots, 9\}$

ab#bc is a string over $\Sigma_3 = \{a, b, \dots, z, \#\}$

))()(is a string over $\Sigma_4 = \{ (,) \}$

Languages

A language is a set of strings over an alphabet.

- Languages can be used to describe problems with “yes/no” answers, for example:

$L_1 =$ The set of all strings over Σ_1 that contain the substring “fool”

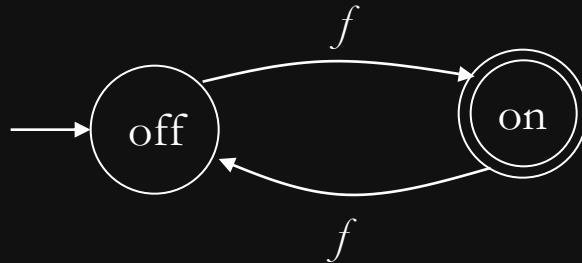
$L_2 =$ The set of all strings over Σ_2 that are divisible by 7
 $= \{7, 14, 21, \dots\}$

$L_3 =$ The set of all strings of the form $s\#s$ where s is any string over $\{a, b, \dots, z\}$

$L_4 =$ The set of all strings over Σ_4 where every (can be matched with a subsequent)

Finite Automata

Example of a finite automaton

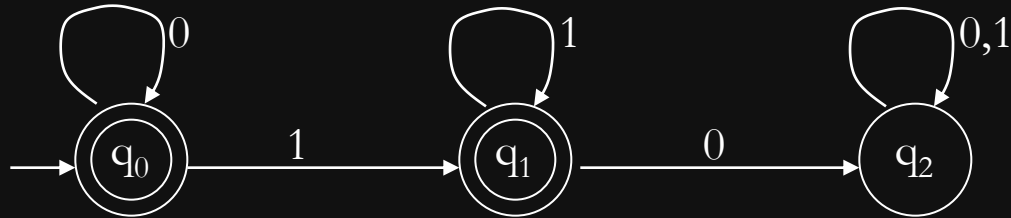


- There are **states** `off` and `on`, the automaton **starts** in `off` and tries to reach the “**good state**” `on`
- What sequences of `f`s lead to the good state?
- Answer: $\{f, fff, fffff, \dots\} = \{f^n: n \text{ is odd}\}$
- This is an **example** of a deterministic finite automaton over alphabet $\{f\}$

Deterministic finite automata

- A **deterministic finite automaton (DFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
 - Q is a finite set of **states**
 - Σ is an **alphabet**
 - $\delta: Q \times \Sigma \rightarrow Q$ is a **transition function**
 - $q_0 \in Q$ is the **initial state**
 - $F \subseteq Q$ is a set of **accepting states (or final states)**.
- In diagrams, the accepting states will be denoted by double loops

Example



alphabet $\Sigma = \{0, 1\}$

start state $Q = \{q_0, q_1, q_2\}$

initial state q_0

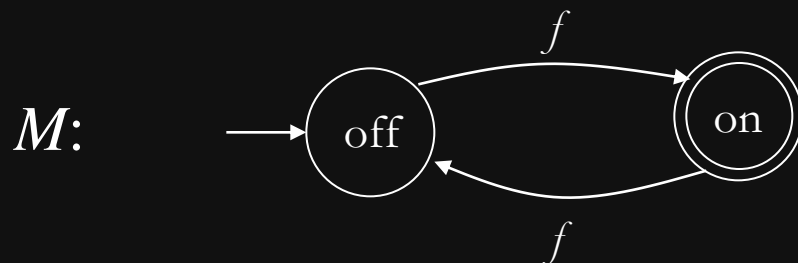
accepting states $F = \{q_0, q_1\}$

transition function δ :

		inputs	
		0	1
states	q_0	q_0	q_1
	q_1	q_2	q_1
	q_2	q_2	q_2

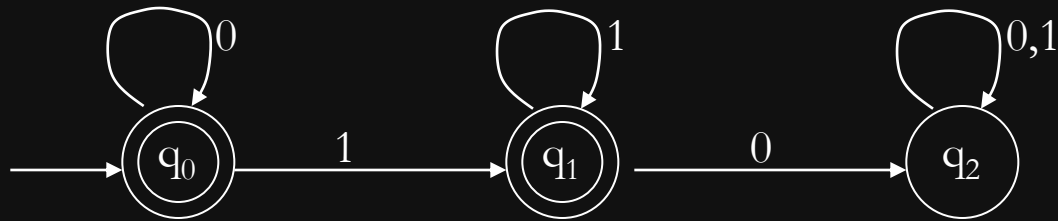
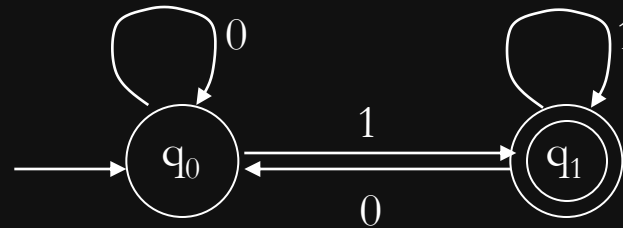
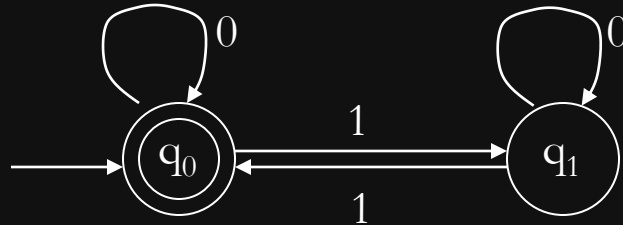
Language of a DFA

The language of a DFA $(Q, \Sigma, \delta, q_0, F)$ is the set of all strings over Σ that, starting from q_0 and following the transitions as the string is read left to right, will reach some accepting state.



- Language of M is $\{f, fff, fffff, \dots\} = \{f^n: n \text{ is odd}\}$

Examples



What are the languages of these DFAs?

Examples

- Construct a DFA that accepts the language

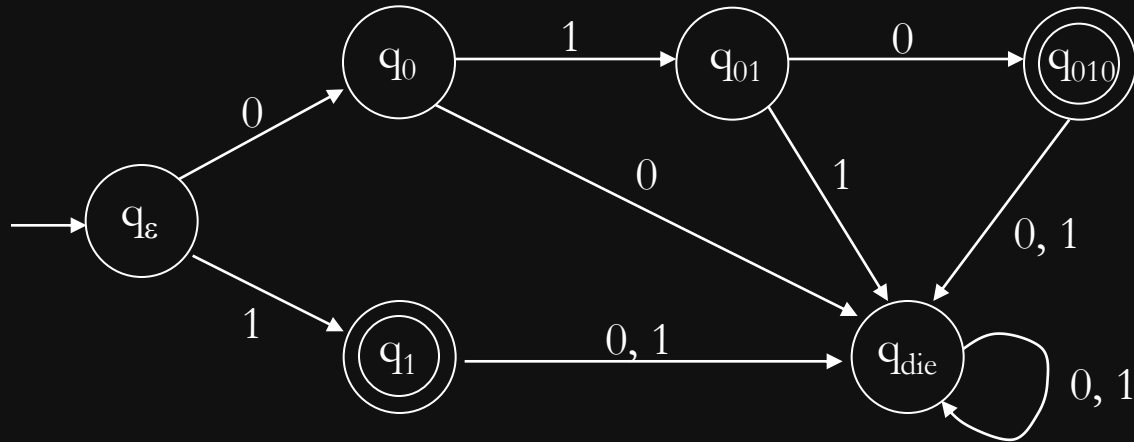
$$L = \{010, 1\} \quad (\Sigma = \{0, 1\})$$

Examples

- Construct a DFA that accepts the language

$$L = \{010, 1\} \quad (\Sigma = \{0, 1\})$$

- Answer



Examples

- Construct a DFA over alphabet $\{0, 1\}$ that accepts all strings that end in 101

Examples

- Construct a DFA over alphabet $\{0, 1\}$ that accepts all strings that end in 101
- **Hint:** The DFA must “remember” the last 3 bits of the string it is reading

Examples

- Construct a DFA over alphabet $\{0, 1\}$ that accepts all strings that end in 101
- Sketch of answer:

