Course organization

Textbook J.E. Hopcroft, R. Motwani, J.D. Ullman *Introduction to Automata Theory, Languages, and Computation* Second Edition, Addison-Wesley, New York, 2001

We shall cover Chapters 1-7

I will add (at least): Moore/Mealy machines, Myhill-Nerode Theorem, abstract state (another algorithm for computing minimal automata)

Exercise Sessions: *very* important

Where does it come from?

Neurophysiologist Warren McCulloch and logician Walter Pitts

"A logical calculus of the ideas immanent in nervous activity"
Bull. Math. Biophysics 5 (1943), pp. 115-133

Developed models of neural networks based on their understanding of neurology

Rich paper: neural newtorks with feedback loops, and attempt to model the nervous structure

Where does it come from?

Work of the logician Church on checking and designing circuit

The model of Pitts-McCulloch was simplified by Kleene (1956)

Rabin and Scott present finite state machine as a simplification of Turing machine

Ken Thompson used the notion of *regular expressions* introduced by Kleene in the UNIX system

"Regular Expression Search Algorithm"
Comm. Assoc. Comp. Mach., Vol. 11, 6, pp. 419–422, 1968.

Applications

- software for finding patterns in large bodies of text (such as collection of web pages)
- software for designing circuits
- lexical analyser of compilers
- model-checking, modelling and verification of embedded systems, software model-checking
- model of real machines: watches, telephone, lock for cars, ...
- application in genetics, regular pattern in the language of protein
- application in linguistic, building of large dictionnary, spell programs, search

Motivation

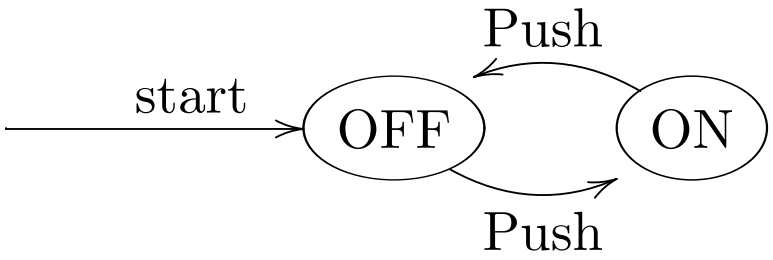
- **Automata** The word "Automata" is the plural of "automaton" which simply means any machine
- Mathematical *model* of what a (finite-state) machine is
- Good illustration of basic mathematical concepts, set theory and proofs (important in computer science)

"Historically, regular expressions are one of computer science's shining examples of how using good theory leads to good programs.

Today, regular expressions have also become a shining example of how ignoring good theory leads to bad programs. The regular expression implementations used by today's popular tools are significantly slower than the ones used in many of those thirty-year-old Unix tools."

Example: on/off switch

Simplest non trivial finite automaton



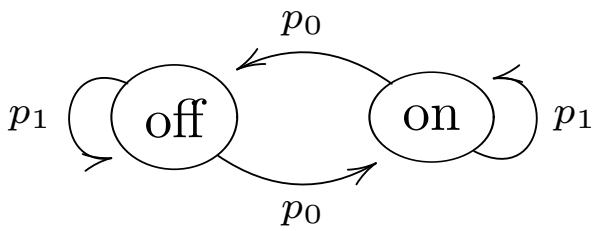
States represented by circles

One state is the *start* state

Arcs labelled by observable *events*

Example: parity counter

The states can be thought of as the "memory" of the machine



Two events: p_0 and p_1

The machine don't do anything for the event p_1

The machine counts the parity of the number of p_0s

finite-state automata $\rightarrow$ finite memory

Functional description

In functional programming, we have two functions f_1, f_0 and the input is a finite list, which can be represented by $N = 0 \mid S\ N$

$f_1\ (S\ n) = f_0\ n, \qquad f_0\ (S\ n) = f_1\ n$

$f_1\ 0 = 1, \qquad f_0\ 0 = 0$

Then $f_1\ n = 1$ iff n is *even*

Functional description

For the parity machine we have the data type $T = p_0\ T \mid p_1\ T \mid 0$ and two functions g_1, g_0

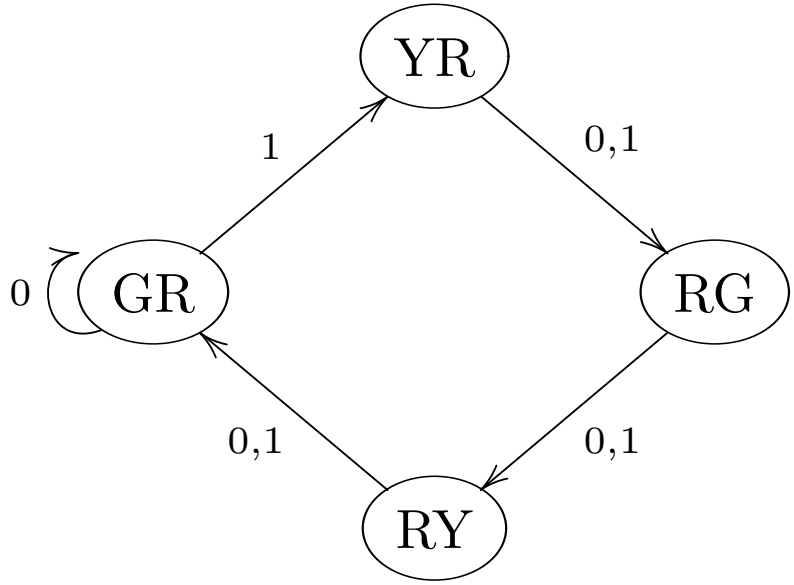
$g_1\ (p_0\ n) = g_0\ n, \qquad g_0\ (p_0\ n) = g_1\ n$

$g_1\ (p_1\ n) = g_1\ n, \qquad g_0\ (p_1\ n) = g_0\ n$

$g_1\ 0 = 1, \qquad g_0\ 0 = 0$

Then $g_1\ n = 1$ iff n contains an even number of constructor p_0

Example: A Simple Traffic Signal

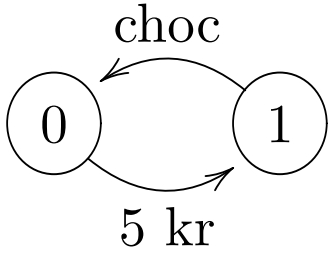


Transitions: 0 (no traffic detected), 1 (traffic detected)

The states represent a situation: GR one traffic light is Green and the other is Red

Example: vending machine

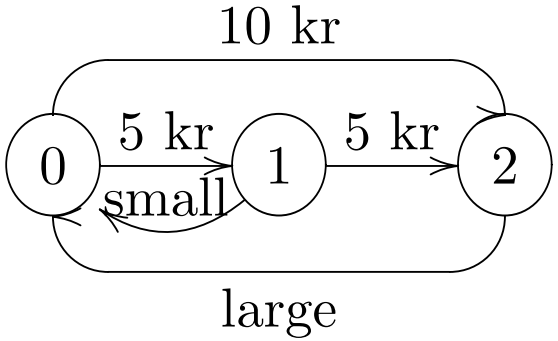
A (very) simple vending machine



What happens if we are in state 0 and ask for a chocolate??

Example: vending machine

A more complex vending machine



In state 1: the machine "remembers" that we have given 5 kr

Alphabet of a machine

Definitions: The alphabet of a given machine is the set of (observable) events of this machine

A possible *behavior* of a machine is a finite sequence of events

Some alphabets

- on/off switch $\Sigma = \{\text{Push}\}$
- simple vending machine $\Sigma = \{5\text{kr}, \text{choc}\}$
- complex vending machine

  $\Sigma = \{5\text{kr}, 10\text{kr}, \text{large}, \text{small}\}$
- parity counter $\Sigma = \{p_0, p_1\}$

Not necessarily "linguistic" applications. In biology, the DNA alphabet is $\{A, C, G, T\}$

Behaviour of a machine

- on/off switch:

  Push, Push Push, Push Push Push, ...

- simple vending machine:

  5kr choc 5kr choc ...

- parity counter $p_0 p_1$ or $p_0 p_0 p_1 p_0 \cdots$

The choice of the alphabet usually involves *deliberate simplifications*

For instance: no exact *timing* (however there exists also more complex notions such as timed automata or hybrid automata; we shall not deal with these automata in this course)

Finite automata provide a simple *mathematical model* of complex machines (or software)

Example: Man, wolf, goat and cabbage problem

A man with a wolf, goat and cabbage is on the left bank of a river.

There is a boat large enough to carry the man and *only one* of the other three. The man wish to cross to the right bank.

However if the man leaves the wolf and goat unattended on either shore, the wolf surely eat the goat.

Similarly, if the goat and the cabbage are left unattended, the goat will eat the cabbage.

Puzzle: Is it possible to cross the river without the goat or cabbage being eaten?

We write all the possible transitions, and look for possible paths between two nodes

Example: Man, wolf, goat and cabbage problem