# Finite Automata Theory and Formal Languages
# Föreläsning 1

Erik Sjöström

March 21, 2016

## 1 Automata

### 1.1 Dictionary definition

*Main Entry:* $au \cdot tom \cdot a \cdot ton$
*Function:* noun
*Inflected Form(s):* plural $au \cdot tom \cdot atons$ or $au \cdot tom \cdot a \cdot ta$
*Etymology:* Latin, from Greek, neuter of automatos
*Date:* 1645

1. a mechanism that is relatively self-operating;
   especially: robot
2. a machine or control mechanism designed to follow automatically a predetermined sequense of operations or respond to encoded instructions
3. an individual who acts in a mechanical fashion

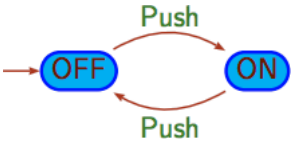
### 1.2 Applications

Models for ...

- Lexical analyser in a compiler
- Software for designing circuits
- Software for finding patterns in large bodies of text such as collections of web pages
- Software for verifying systems with a finit number of different states such as protocols
- Real machines like vending machines, telephones, street lights, ...
- Application in linguistics, building of large dictionary, spell programs, search
- Application in genetics, regular pattern in the language of protein

### 1.3 Example: on/off-switch

A very simple finite automaton:



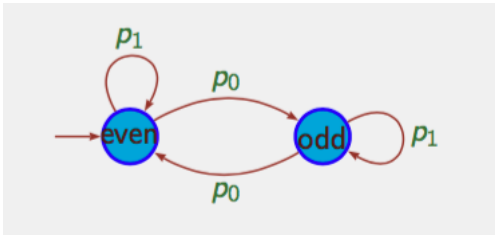
*States* represented by "circles".
One *starting* state, indicated with an arrow into it.
Labelled arc between states represent observable *events*.

Sometimes one of more *final* states, indicated with a double circle: 



## 1.4 Example: Parity Counter

The states of an automaton can be thought of as its *memory*.
A finite-state automaton has *finite memory*!

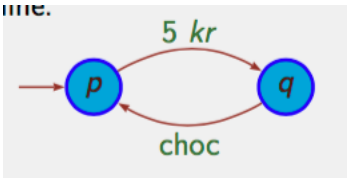



Two events: $p_0$ and $p_1$.
The machine does nothing on the event $p_1$.
The machine remembers the parity of the number of $p_0$'s.

Correctnes: We would like to prove that the automata is on the state even *iff* an even number of $p_0$ were pressed.

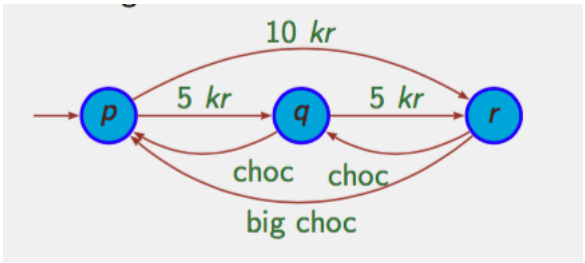
## 1.5 Example: Vending Machines

A simple vending machine:




What happens if we ask for chocolate on $p$?

**A more complex vending machine**:



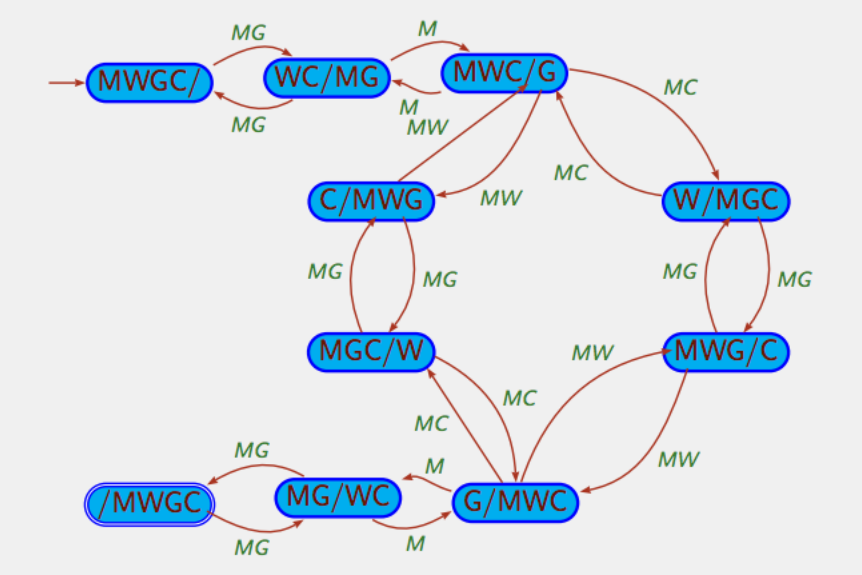

## 1.6 Example: The Man, the Wolf, the Goat, and the Cabbage

A man with a wolf, a goat and a 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 things.
The man wants to move everything to the right bank.

However if the man leaves the wolf and the goat unattended on either shore, the wolf surely will eat the goat. Similarly, if the goat and the cabbage are left unattended, the goat will eat the cabbage.

**Problem**: Is it possible to cross the river without the goat or cabbage being eaten?
How many possible solutions does the problem have?

**Solution**: We design an automaton that models the problem with all its possible transitions, and look for paths between the initial and final state.




# 2 Formal Language

## 2.1 From Wikipedia:

In mathematics, computer science, and linguistics, a formal language is a set of strings of symbols that may be constrained by rules that are specific to it.
The alphabet of a formal language is the set of symbols, letters, or tokens from which the strings of the language may be formed; frequently it is required to be finite.
The strings formed from this alphabet are called words, and the words that belong to a particular formal language are sometimes called well-formed words or well-formed formulas.
A formal language is often defined by means of a formal grammar such as a regular grammar or context-free grammar, also called its formation rule.

## 2.2 Example: Formal Representation of Numbers and Identifiers in a Programming Language

A regular grammar for numbers and identifiers:

$$L \rightarrow A \mid B \mid ... \mid Z \mid a \mid b \mid ... \mid z$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

$$Nr \rightarrow D \mid D\ Nr$$
$$Id \rightarrow L\ LLoD$$
$$LLoD \rightarrow L\ LLoD \mid D\ LLoD \mid \epsilon$$

A regular expression for numbers:

$$(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)^{+}$$

A regular expression for identifiers:

$$(A + ... + Z + a + ... + z)(A + ... + Z + a + ... + z + 0 + ... + 9)^{*}$$

### 2.3 Example: Very Simple Expression

A context-free grammar for simple expression:

E → E+E | E-E | E*E | E/E | (E) | Nr | Id
Nr → ...
Id → ...

Correctness: We could want to prove that:

- Any expression has as many "(" as ")"
- Any expression has 1 more "term" than the number of "operations"
- ...

### 2.4 More Complex Example

A better context-free grammar for simple expression:

E → E+T | E-T | T
T → T*F | T/F | F
F → (E) | Nr | Id

A context-free grammar for C++ compund statements:

S → LC
LC → $\epsilon$ | C LC
C → S | **if** (E) C | **if** (E) C **else** C |
**while** (E) C | **do** C **while** (E) | **for** (C E;E) C |
**case** E:C | **switch** (E) C | **return** E; | **goto** Id;
**brake** ; | **continue**;
⋮

## 3 Formal proofs

Many times you will need to prove that your program/model/grammar/... is "correct" (satisifies a certain specification/property).
In particular, you won't get a complex program/model/grammar/... right if you don't understand what is going on.
**Different kind of formal proofs:**

- Deductive proofs
- Proofs by contradiction
- Proofs by counterexample
- Proofs by (structural) induction

## 4 Regular Language

Finite automata were originally prposed in 1940's as models of neural networks.
Turned out to have many other applications!

In the 1950's, the mathematician Stephen Kleene described these models using mathematical notation (regular expressions, 1956).
Ken Thompson used the notion of regular expressions introduced by Kleene in the UNIX system.
(Observe that Kleene's regular expressions are not really the same as UNIX's regular expressions.)

Both formalisms define the regular languages.

# 5 Context-Free Languages

We can give a bit more power to finite automata by adding a stack that contains data and obtain a push down automata.

In the mid-1950's Noam Chomsky developed the context-free grammars.
Context-free grammars play a central role in the description and design of programming languages and compilers.

Both formalism define the context-free language.

# 6 Church-Turing Thesis

In the 1930's there has been quite a lot of work about the nature of effectively computable (calculable) functions:

- Recursive functions by Stephen Kleene (after ideas by Kurt Gödel)
- $\lambda$-calculus by Alonzo Church
- Turing machines by Alan Turing

The three models of computation were shown to be equivalent by Church, Kleene & (John Barkley) Rosser (1934-6) and Turing (1936-7)

The Church-Turing theses states that if an algorithm (a procedure that terminates) exists then, there is an equivalen Turing machine, a revursively-definable function , or a definable $\lambda$-function for that algorithm.

# 7 Turing Machine

Simple theoretical device that manipulates symbols contained on a tape.

It is as "powerful" as the computers we know today (in terms of what they can compute)

It allows the study of decidability: what can or cannot be done by a computer (halting problem)

Computability vs complexity theory: we should distinguish between what can or cannot be done by a computer, and the inherent difficulty of the problem (tractable (polynomial)/intractable (NP-hard) problems).