Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2014

Lecture 1
Ana Bove

March 17th 2014

Overview of today's lecture:

- Course organisation;
- Overview of the course.

Course Level, Load and Moments

Level: This course is a *bachelor* course in year 1–2.

Load: 7.5 pts means ca. 20–25 hours per week!!

Course Moments: (from VT2013)

- *Individual weekly assignments:* 1.5pts.
  To pass the assignment part you need to get at least 50% of the sum of the points of all the weekly assignments together.

- *Exam:* 6pts.
  Dates: May 28th and August 20th.
  No book or help allowed.

Note: To pass the course you actually need to *follow ALL* the course!

More on Assignments

- They must be done *completely* on your own!

  Note: Be aware that assignments are part of the examination of the course and they **should** be done *individually*!
  Standard procedure will be followed if copied solutions are detected.

- How to submit? Via the Fire system, check course web page.
  `http://xdat09.ce.chalmers.se/fafl`.

- Who shall submit? *Registered* students who haven't passed them yet!

Lectures and Consultation Time

Lectures: Ana Bove, `bove@chalmers.se`

Mondays 13:15–15:00: in HB3 (weeks 1–7) and HC4 (week 8)
Tuesday 18/3 9:00–11:45: in HA1 *ONLY* in week 1
Thursdays 13:15–15:00: all study week but week 5, in HB3

Consultation Time: Ana Bove

Wednesdays 15:15–17:00: all weeks but study week 1 and week 5, in ES51

Exercises and Assistants

Exercise Sessions: *VERY important!!*
Pablo Buiras, `buiras@chalmers.se`
Daniel Hausknecht, `daniel.hausknecht@chalmers.se`
Simon Huber, `simonhu@chalmers.se`
Inari Listenmaa, `inari@chalmers.se`
Andrea Vezzosi, `vezzosi@chalmers.se`

Thursday 20/3 10:00–11:45: for ALL students who need to recap on discrete math concepts, in HC3 (week 1).

Tuesdays 10:00–11:45: in EF (weeks 2–8).
Mainly for students NOT in their first year.

Thursdays 10:00–11:45: in EE (weeks 2–3, 6–8) and EB (week 4).
Wednesday 30/4 10:00–11:45: in EF (ONLY in week 5).
Mainly for FIRST years students.

Information and Literature

Web Page: `http://www.cse.chalmers.se/edu/course/TMV027`
Accessible from CTH "studieportalen" and GU "GUL".
Check it regularly for news!

Mailing List: `fafl@lists.chalmers.se`
With your CTH/GU mail address.

Book: *Introduction to Automata Theory, Languages, and Computation*, by Hopcroft, Motwani and Ullman. Addison-Wesley.
We will cover chapters 1 to 5, 7 and a bit of chapter 8.

Wikipedia: `http://en.wikipedia.org/wiki`
Youtube: `http://www.youtube.com`

Course Evaluation

I need 2-3 GU + 2-3 IT student representatives this week.

Comments/Changes from last year:

- Assignments were very much appreciated and useful!
- We got many more students so some re-organisation of schema;
- Try not to re-use the examples from the book;
- Removed most of guest lecture(s).

Next year the final grade will account work on both assignments and exam.

Programming Bits in the Course

The course doesn't require much programming tasks.

Still

- I will present some Haskell programs simulating certain automaton or implementing an algorithm.
  (Some of you should know Haskell, if you do not I really recommend you learn it: it is very elegant and nice!);
- You might be required to implement some algorithm as part of an assignment (here you can use either Haskell or Java);
- You should implement the algorithm to improve your knowledge and understanding!

Chinese Proverb

I hear and I forget.

I see and I remember.

I do and I understand.

Confucius

Chinese philosopher and reformer (551 BC - 479 BC)

Automata

Dictionary definition:

```
Main Entry: au·tom·a·ton
Function: noun
Inflected Form(s): plural au·tom·atons or au·tom·a·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 sequence of operations or
    respond to encoded instructions
3 : an individual who acts in a mechanical fashion
```

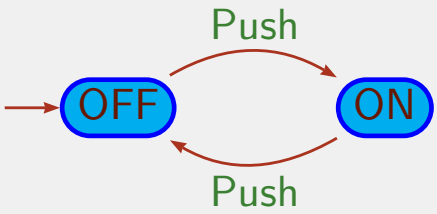
Automata: Applications

Models for ...

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

Example: on/off-switch

A very simple finite automaton:



States represented by "circles".

One state is the *starting* state, indicated with an arrow into it.

Arcs between states are labelled by observable *events*.

Often we need one or more *final* states, indicated with a double circle.

Functional Description of on/off-switch

Let us define 2 functions f_{OFF} and f_{ON} representing the 2 states of the automaton.

The input can be represented as a "finite list" give by data $N = 0 \mid \mathrm{P}\ N$.

The functional description of the automaton is:

$$f_{\mathrm{OFF}}, f_{\mathrm{ON}} : N \to \{\mathrm{Off}, \mathrm{On}\}$$

$$\begin{array}{ll} f_{\mathrm{OFF}}\ 0 = \mathrm{Off} & f_{\mathrm{ON}}\ 0 = \mathrm{On} \\ f_{\mathrm{OFF}}\ (\mathrm{P}\ n) = f_{\mathrm{ON}}\ n & f_{\mathrm{ON}}\ (\mathrm{P}\ n) = f_{\mathrm{OFF}}\ n \end{array}$$

Proving Properties about the on/off-switch

We would like to prove that, if we start on the state OFF and we push p times then:

the automaton is in state OFF iff p is even

and

the automaton is in state ON iff p is odd.

Alternatively, we could prove that for $n \in N$:

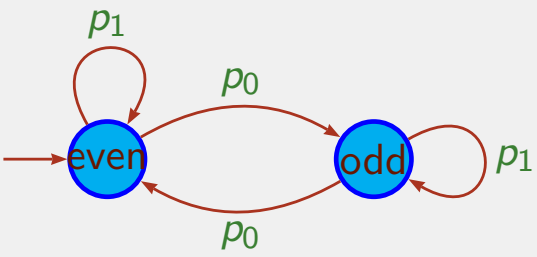
$f_{\mathrm{OFF}}\ n = \mathrm{Off}$ iff number of P's in n is even

and

$f_{\mathrm{ON}}\ n = \mathrm{On}$ iff number of P's in n is even.

Example: Parity Counter

The states of an automaton can be thought of as the *memory* of the machine.



Two events: p_0 and p_1.

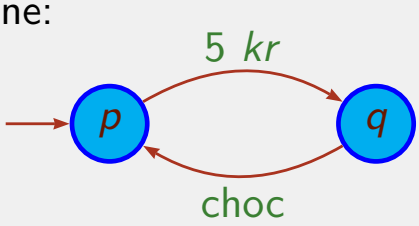
The machine does nothing on the event p_1.
The machine counts the parity of the number of p_0's.

A finite-state automaton has *finite memory*!

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

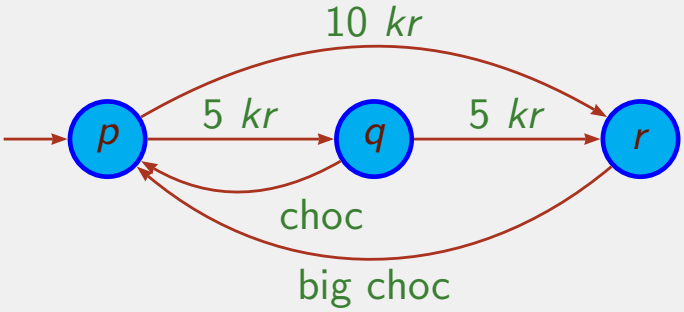
Example: Vending Machines

A simple vending machine:



What does it happen if we ask for a chocolate on p?

A more complex vending machine:



State q remembers it has already got 5 *kr*.

Problem: 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 wish to cross everything to the right bank.

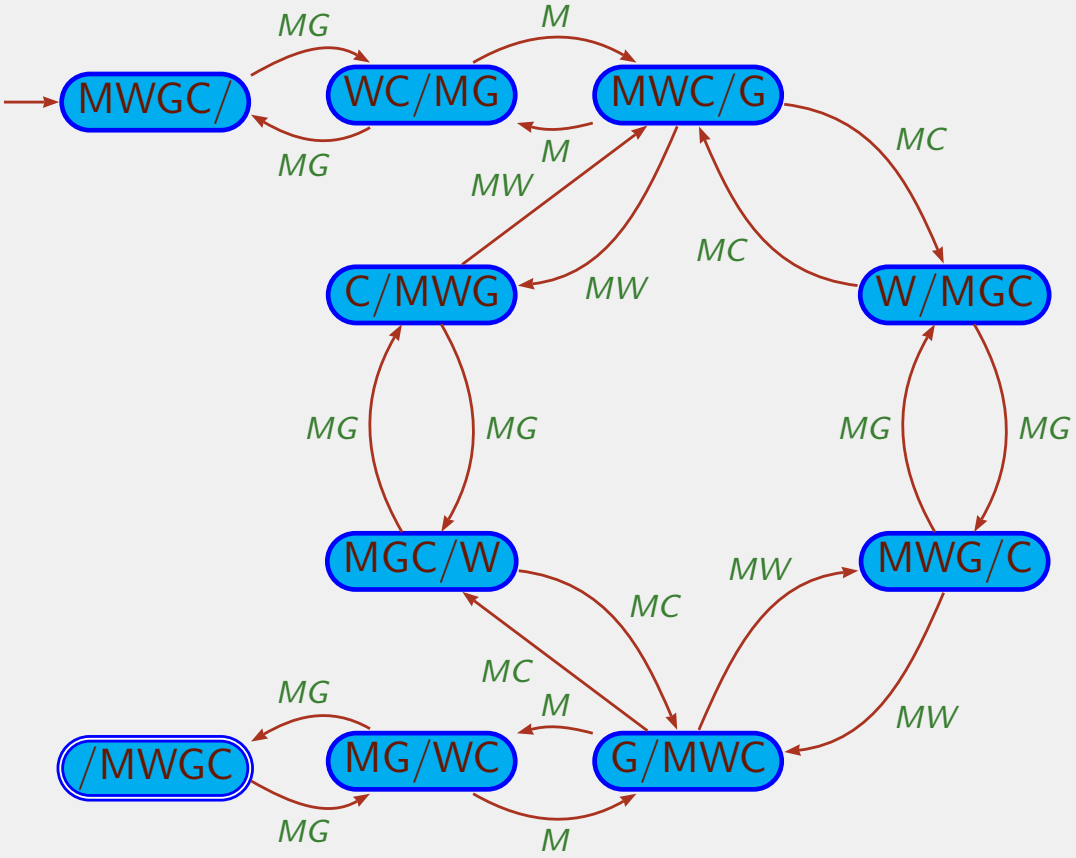
However if the man leaves the wolf and the goat unattended on either shore, the wolf surely will eat the goat.

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

Puzzle: Is it possible to cross the river without the goat or cabbage being eaten?
How many possible solutions the problem has?

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

Solution: The Man, the Wolf, the Goat and the Cabbage



Formal Languages

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.

Example: Formal Representation of Numbers and Identifiers

A context-free grammar for numbers and identifiers

$$
\begin{array}{lcl}
L & \rightarrow & A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \\
D & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

$$
\begin{array}{lcl}
Nr & \rightarrow & D \mid D\ Nr \\
Id & \rightarrow & L\ LLoD \\
LLoD & \rightarrow & L\ LLoD \mid D\ LLoD \mid \epsilon
\end{array}
$$

A regular expression for numbers:

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

A regular expression for identifiers:

$$(A \mid \ldots \mid Z \mid a \mid \ldots \mid z)(A \mid \ldots \mid Z \mid a \mid \ldots \mid z \mid 0 \mid \ldots \mid 9)^{*}$$

Example: Very Simple Expressions

A context-free grammar for simple expression:

$$\begin{array}{lcl} E & \rightarrow & E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid Nr \mid Id \\ Nr & \rightarrow & \cdots \\ Id & \rightarrow & \cdots \end{array}$$

We might now want to prove that

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

More Complex Examples

A better context-free grammar for simple expression:

$$\begin{array}{lcl} E & \rightarrow & E + T \mid E - T \mid T \\ T & \rightarrow & T * F \mid T/F \mid F \\ F & \rightarrow & (E) \mid Nr \mid Id \end{array}$$

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

$$\begin{array}{lcl} S & \rightarrow & \{LC\} \\ LC & \rightarrow & \epsilon \mid C\ LC \\ C & \rightarrow & S \mid \mathit{if}\ (E)\ C \mid \mathit{if}\ (E)\ C\ \mathit{else}\ C \mid \\ & & \mathit{while}\ (E)\ C \mid \mathit{do}\ C\ \mathit{while}\ (E) \mid \mathit{for}\ (C\ E; E)\ C \mid \\ & & \mathit{case}\ E : C \mid \mathit{switch}\ (E)\ C \mid \mathit{return}\ E; \mid \mathit{goto}\ Id; \\ & & \mathit{break}; \mid \mathit{continue}; \\ & \vdots & \end{array}$$

Overview of the Course

- Formal proofs;
- Regular languages;
- Context-free languages;
- Turing machines (as much as time allows).

Formal Proofs

Many times you will need to prove that your program/model/grammar/... is "correct" (satisfies 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 counterexamples;
- Proofs by (structural) induction.

Regular Languages

Finite automata were originally proposed in the 1940's as models of neural networks.

Turned out to have many other applications!

In the 1950s, 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*.

Context-Free Languages

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

In this way we extend finite automata into a *push down automata*.

In the mid-1950s 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 formalisms define the *context-free languages*.

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;
- λ-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 thesis* states that if an algorithm (a procedure that terminates) exists then, there is an equivalent Turing machine, a recursively-definable function, or a definable λ-function for that algorithm.

Turing Machine (ca 1936–7)

Simple theoretical device that manipulates symbols contained on a strip of tape.

It is as "powerful" as the computers we know today (in term 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).

Learning Outcome of the Course

After completion of this course, the student should be able to:

- Explain and manipulate the different concepts in automata theory and formal languages;
- Have a clear understanding about the equivalence between (non-)deterministic finite automata and regular expressions;
- Acquire a good understanding of the power and the limitations of regular languages and context-free languages;
- Prove properties of languages, grammars and automata with rigorously formal mathematical methods;
- Design automata, regular expressions and context-free grammars accepting or generating a certain language;
- Describe the language accepted by an automata or generated by a regular expression or a context-free grammar;
- Simplify automata and context-free grammars;
- Determine if a certain word belongs to a language;
- Define Turing machines performing simple tasks;
- Differentiate and manipulate formal descriptions of languages, automata and grammars.

Overview of Next Lecture

Section 1.5 in the book and more:

- Recap on logic;
- Recap on sets, relations and functions;
- Central concepts of automata theory.