Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2016

Lecture 1
Ana Bove

March 21st 2016

Overview of today's lecture:

- Overview of the course;
- Course organisation.

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
```

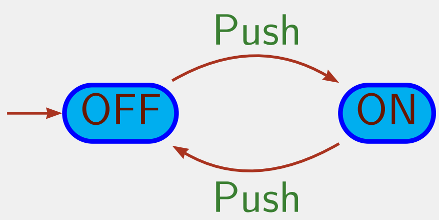
Automata: Applications

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 *starting* state, indicated with an arrow into it.

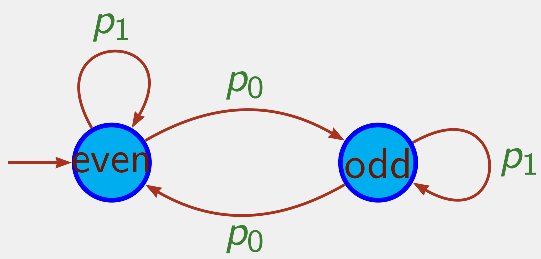
Labelled arcs between states represent observable *events*.

Sometimes one or more *final* states, indicated with a double circle.

Example: Parity Counter

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

A finite-state automaton has *finite memory*!



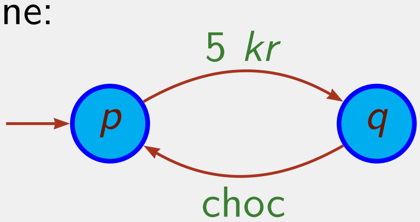
Two events: p_0 and p_1.

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

Correctness: We could like to prove that the automata is on the state even iff an even number of p_0 were pressed.

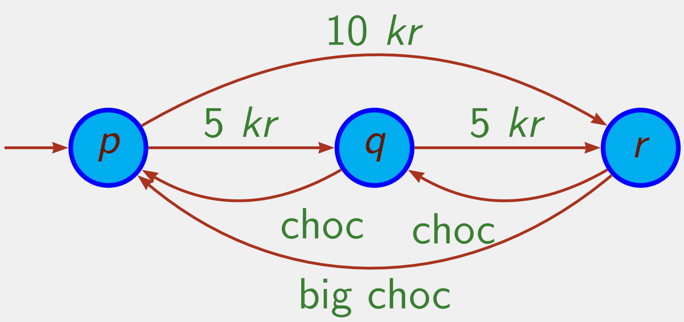
Example: Vending Machines

A simple vending machine:



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

A more complex vending machine:



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 wish to cross everything to the right bank.

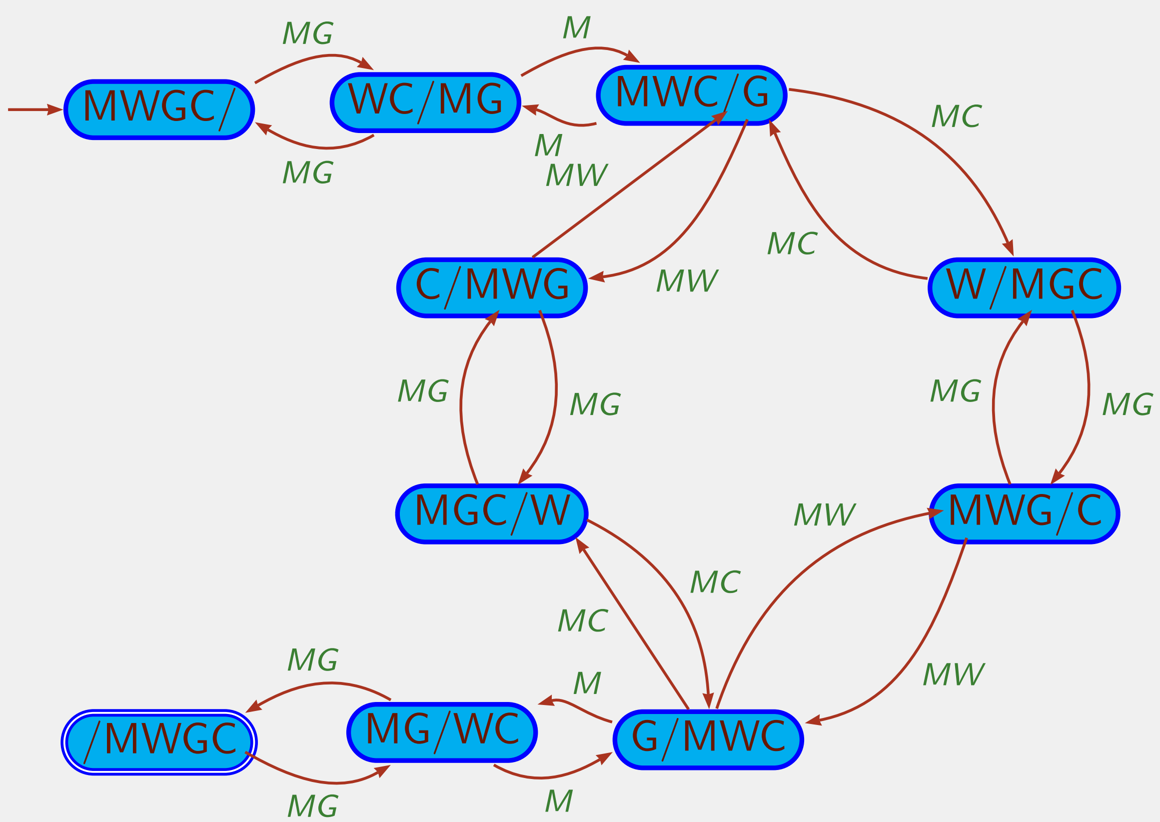
However if the man leaves the wolf and the goat unattended on either shore, the wolf surely will eat the goat.

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

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

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

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 in a Programming Language

A regular grammar for numbers and identifiers

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

A regular expression for numbers:

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

A regular expression for identifiers:

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

Example: Very Simple Expressions

A context-free grammar for simple expression:

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

Correctness: We could 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:

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

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

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

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 and obtain 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 (after ideas by Kurt Gödel);
- 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 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).

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.

People and Contact Information

Course Responsible and Examiner:
Ana Bove, `bove@chalmers.se`

Assistants:
(Pablo Buiras, `buiras@chalmers.se`)
Victor López, `lopezv@chalmers.se`
Daniel Schoepe, `schoepe@chalmers.se`
Marco Vassena, `vassena@chalmers.se`
Andrea Vezzosi, `vezzosi@chalmers.se`

Course Mailing List: `fafl@lists.chalmers.se`

You will get registered to it next week with your CTH/GU mail address.

The list is moderated.

Course Level, Load and Web Page

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

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

Note: It is important to follow the course's pace in order to pass it!

Web Page: `http://www.cse.chalmers.se/edu/course/TMV027`

Accessible from CTH "studieportalen" and GU "GUL".
Check it regularly for news!

Literature and Other Material

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.

Note: Notation in the book is sometimes different to that used in class.

Wikipedia: `http://en.wikipedia.org/wiki`

Youtube: `http://www.youtube.com`

Internet ...

Lectures (Check TimeEdit!)

Lectures: Ana Bove

Mondays 13:15–15:00: in HB3
Tuesday 22/3 9:00–11:45: in HA4, *ONLY* in week 1
Thursdays 13:15–15:00: in HB3, *EXCEPT* weeks 4 & 5
Wednesday 27/4 13:15–15:00: in HB3, *ONLY* in week 4

Note: You *MUST prepare* before each lecture in order to follow the course pace!

Exercises and Consultation Time (Check TimeEdit!)

Exercise Sessions: Run by the assistants.
VERY important to start working on the weekly problems
Come and ask questions!!!

Thursday 24/3 10:00–11:45: in HC3, *ONLY* week 1
for ALL students who need to recap on discrete math concepts

Mondays 15:15–17:00: in EA, weeks 2–8
Tuesday 10:00–11:45: in EA, weeks 2–8

Consultation Time: Ana Bove

Wednesdays 15:15–17:00: in EL41, weeks 2–8
Very important if you have trouble understanding something

Solutions to the Exercises

Note: There are NO solutions to exercises!!!

(This is usually the main complain in this course.)

Why?

- Expensive to create good solutions;
- Some problems have many possible solutions;
- Pedagogically not always good: you learn mostly by doing;
- Asking the teachers in case of doubt is a better way to learn;
- You need to learn how to solve problems without a solution....
  Who will pay you to solve something one already has the solution to?

If you *really* need exercises with solutions, google and you will find plenty!

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.

  (If you do not know Haskell I recommend you learn it: it is very elegant, nice and of increasing importance in industry!);

- You should implement the algorithm to improve your knowledge and understanding!

Examination

From VT2013 the course has 2 *obligatory* parts:

- *Individual weekly assignments:* 1.5pts.
- *Individual written exam:* 6pts, no book or help allowed.
  Dates for 2016: June 1st pm and August 17th am.

From VT2015 the final grade is based on the performance on *both* parts!

More on Assignments

- To pass the assignment part you need to get at least 50% of the sum of the points of all the weekly assignments together.
- 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.
- Who shall submit? Students who *registered VT13 or later* and who have *NOT* passed them yet!

Final Grade

- Students registered before VT2015: final grade = grade in the exam.
  See course web page for information that might apply in your case.

- Students registered VT2015 or later:
  Max. points in exam: 60, min 27 pts to pass it.
  Max. points in assignments: 64, min 32 pts to pass them.

  Final points (max. 76) = nr of points got in the exam + 25% of the points gathered in the assignment

Chalmers:

Final Grade	Final Pts
3	35
4	46
5	57

GU:

Final Grade	Final Pts
G	35
VG	53

Note: You need to pass *both parts* in order to pass the course!

Comments from Course Evaluation

If I would redo the course, I would tried to have done the excercises before the lecture = trying to be one week ahead of the lectures.
I figured it out in the end, that I was doing the excercises after the lectures = sometimes it was hard to follow the advanced parts of each topics.

Tell people even more how much it really is to take in, and how extremely important it is to be in phase.

The consultation time, keep it as it is, it's perfect and was a very important part of the course for me.

Comments from Course Evaluation (Cont.)

I also liked that there weren't any answers to the exercises; though scary at the beginning, it gave a deeper understanding of the subject of the exercise when one had to think twice and trust one's own solution. It was also a great practice for the exam and future employment.

The assignments were extremely helpful and most probably the reason on why I passed this course exam.

The assignments were essential for me in completing the course, the gave the motivation I needed to keep up with the course and properly learn last weeks material.

I was also very happy to see that the exam was basically the same type of question as the assignments, so I felt the most prepared for the exam that I have in any of courses so far.

Changes from Last Year

Not many ...

- Might try to use kahoots;
- Will try to identify algorithms where the example can be shown before the theory.

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.