Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2013

Lecture 1
Ana Bove

March 18th 2013

Overview of today's lecture:

- Course Organisation;
- Overview of the Course.

Course Organisation

Level: This course is a *bachelor* course.

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.
  How to submit? Via the Fire system, check course web page.
- *Exam:* 6pts.
  Dates: May 28th and August 21st.
  No books or written help during the exam.

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.

Course Organisation (Cont.)

Lectures: Monday 18/3 13:15–15:00 in EC, *ONLY* week 1.
Tuesdays mornings in EC, check the web for starting time (9:00 or 10:00).
Thursdays 13:15–15:00 in EC, *NOT* in weeks 6 and 8.
Ana Bove, `bove@chalmers.se`

Exercise Sessions: Mondays 13:15–15:00 in EC, *NOT* in week 1.
Note: If there is need and the room is not booked for something else, exercise sessions can be extended until 16:00.
VERY important!!
Pablo Buiras, `buiras@chalmers.se`
Simon Huber, `simonhu@chalmers.se`

Consultation Time: Thursdays 10:00–11:45, *NOT* in week 6.
Week 1 in EL41, other weeks in group room 5217.
Ana Bove

Course Organisation (Cont.)

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

Book: *Introduction to Automata Theory, Languages, and Computation*, by Hopcroft, Motwani and Ullman. Addison-Wesley.
Both second and third edition are fine.
We will cover chapters 1 to 7 and a bit of chapter 8 (if time allows).

Web Page: `http://www.cse.chalmers.se/edu/course/TMV027`
Accessible from Chalmers “studieportalen”.
Check it regularly for news!

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

Course Organisation (Cont.)

Course Evaluation: I need 3-4 GU + 3-4 IT student representatives this week.

Changes from last year:

- Assignments are compulsory;
- A recap lecture on sets, functions and relations;
- Some guest lecture(s);
- Consultation and exercises on different days.

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.
  (Many 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).

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 ...

- Software for designing circuits;
- Lexical analyser in a compiler;
- 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 simple non-trivial finite automaton:

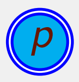
$$\begin{tikzcd}
{} \arrow[r] & \text{OFF} \arrow[r, bend left, "\text{Push}"] & \text{ON} \arrow[l, bend left, "\text{Push}"]
\end{tikzcd}$$

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 $N = 0 \mid \text{P } N$.

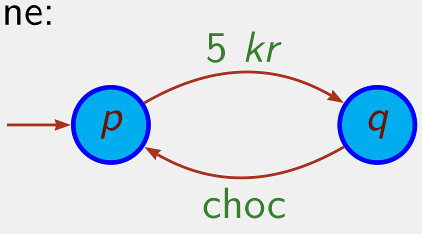
The functional description of the automaton is:

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

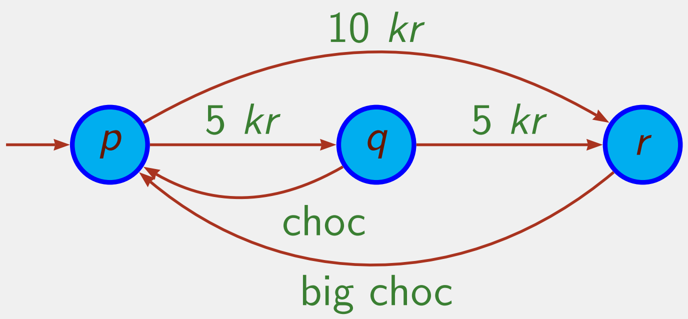
$$\begin{array}{ll} f_{\text{OFF}}\ 0 = \text{Off} & \quad f_{\text{ON}}\ 0 = \text{On} \\ f_{\text{OFF}}\ (\text{P }n) = f_{\text{ON}}\ n & \quad f_{\text{ON}}\ (\text{P }n) = f_{\text{OFF}}\ n \end{array}$$

Example: Vending Machines

A simple vending machine:



A more complex vending machine:



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

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.

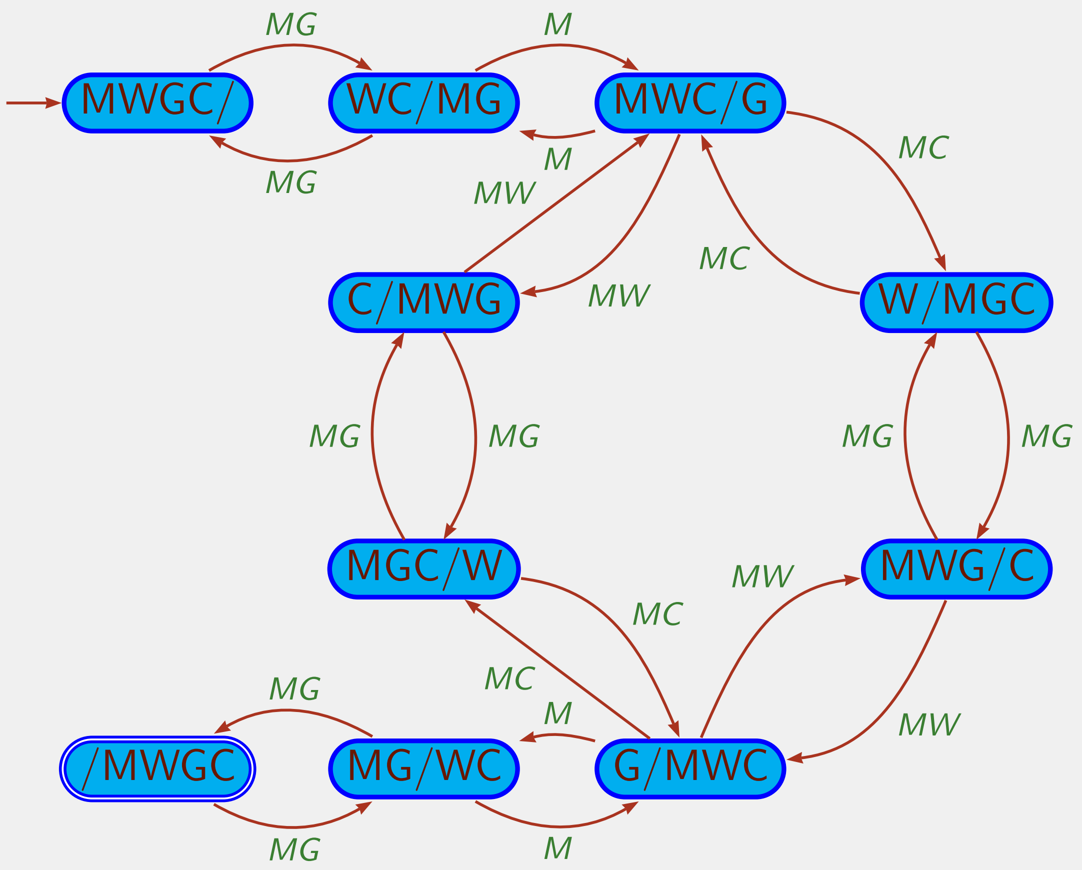
However if the man leaves the wolf and the goat unattended on either shore, the wolf surely will eat the goat.

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?

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: C++ Compound Statements

A context free grammar for statements:

$$
\begin{array}{lcl}
S & \to & \{LC\} \\
LC & \to & C\ LC \mid \epsilon \\
C & \to & S \mid \mathit{if}\ (E)\ C \mid \mathit{if}\ (E)\ C\ \mathit{else}\ C \mid \\
 & & \mathit{while}\ (E)\ C \mid \mathit{do}\ C\ \mathit{while}\ (E) \mid \mathit{for}\ (C\ E; E)\ C \mid \\
 & & \mathit{case}\ E : C \mid \mathit{switch}\ (E)\ C \mid \mathit{return}\ E; \mid \mathit{goto}\ Id; \\
 & & \mathit{break}; \mid \mathit{continue}; \\
E & \to & \ldots \\
Id & \to & L\ LLoD \\
LLoD & \to & L\ LLoD \mid D\ LLoD \mid \epsilon \\
L & \to & A \mid B \mid ... \mid Z \mid a \mid b \mid ... \mid z \\
D & \to & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

A regular expression for identifiers:

$$(A \mid ... \mid Z \mid a \mid ... \mid z)(A \mid ... \mid Z \mid a \mid ... \mid z \mid 0 \mid ... \mid 9)^*$$

Overview of the Course

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

Formal Proofs

Many times you will need to prove that your program is "correct" (satisfies a certain specification).

In particular, you won't get a complex program 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.

Example: on/off-switch

Recall the description of the on/off-switch.

We would like to prove that:

the automaton is in state OFF after n pushes iff n is even

and

the automaton is in state ON after n pushes iff n is odd.

Alternatively, we could prove that:

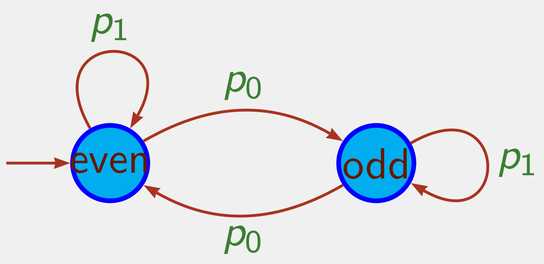
$f_{\mathrm{OFF}}\ n = \mathrm{Off}$ iff n is even

and

$f_{\mathrm{ON}}\ n = \mathrm{On}$ iff n is odd.

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.

Functional Description: Parity Counter

Let us define 2 functions f_{even} and f_{odd} representing the 2 states of the automaton.

The input can be represented by the data type $T = 0 \mid p_0\ T \mid p_1\ T$.

The description of the automaton is:

$$f_{\text{even}}, f_{\text{odd}} : T \to \{\text{Even}, \text{Odd}\}$$

$$\begin{array}{ll}
f_{\text{even}}\ 0 = \text{Even} & f_{\text{odd}}\ 0 = \text{Odd} \\
f_{\text{even}}\ (p_0\ n) = f_{\text{odd}}\ n & f_{\text{odd}}\ (p_0\ n) = f_{\text{even}}\ n \\
f_{\text{even}}\ (p_1\ n) = f_{\text{even}}\ n & f_{\text{odd}}\ (p_1\ n) = f_{\text{odd}}\ n
\end{array}$$

We now would like to prove that $f_{\text{even}}\ n = \text{Even}$ iff n contains an even number of constructors p_0.

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 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 computational processes were shown to be equivalent by Church, Kleene, (John Barkley) Rosser (1934—6) and Alan 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.