G52MAL
Machines and Their Languages
Lecture 1
Administrative Details and Introduction
Henrik Nilsson
University of Nottingham

Finding People and Information
- Henrik Nilsson
  Room A08
- Moodle
- Main module web page: www.cs.nott.ac.uk/~nhn/G52MAL

Aims of the Course
- To familiarize you with key Computer Science concepts in central areas like
  - Automata Theory
  - Formal Languages
  - Models of Computation
  - Complexity Theory
- To equip you with tools with wide applicability in the fields of CS and IT.

Organization (1)
- Lectures:
  - Two 1 h lectures per week.
  - Detailed but somewhat tentative schedule available on the module web page.
- Coursework:
  - 4 Bi-weekly problem sets.
  - Made available via the module web page.
  - Best 3 counts.
  - Deadlines: 6/2, 20/2, 6/3, 20/3.

Organization (2)
- Assessment:
  - Coursework, 25 %
  - 2 hour written examination, 75 %
- However, results are by 100 % written examination (standard School policy)

Assessment:
- Coursework, 25 %
- 2 hour written examination, 75 %
- However, results are by 100 % written examination (standard School policy)

Literature (1)
- Dr. Thorsten Altenkirch's and my G52MAL lecture notes. (Available via the G52MAL module page.)

Literature (2)
- Supplementary material; e.g., slides, sample program code. (Available via the G52MAL module page.)
- Your own notes from the lectures!

Content
1. Mathematical models of computation, such as:
   - Finite automata
   - Pushdown automata
   - Turing machines
2. How to specify formal languages?
   - Regular expressions
   - Context free grammars
   - Context sensitive grammars
3. The relation between 1 and 2.
4. Applications: Scanning and Parsing
Formal languages and automata have lots of applications in CS and IT. Some examples:

- Specification of programming languages
- Implementation of programming language processors (G52MAL feeds into G53CMP)
- XML and DTDs (Document Type Definition)
- Finding words and patterns in large bodies of text, e.g. in web pages.
- Verification of systems with finite number of states, e.g. communication protocols.

As a concrete example, just the other day, there was this job opening being advertised:

The Strats team at Standard Chartered is hiring a developer for a 1 year contracting role in London. The role is to develop and extend our parsing and validation library for FpML, using the FpML Haskell library to parse and build financial product data into our internal Haskell data types.

https://donsbot.wordpress.com/2015/01/28/

Automata are essential for the study of the limits of computation. Deep theoretical questions with big practical implications. Two key issues:

- What can a computer do at all? Decidability
- What can a computer do efficiently? Time and space Complexity

Imagine you’re the lead developer for a new web browser. It obviously needs the capability to run JavaScript.

To make your product stand out from the competition, your boss proposes you implement a termination check: any non-terminating JavaScript programs can then be rejected, without being run.

If you succeed, your salary will be doubled. But if you fail, you’d have to look for a new job.

Should you accept?

Example: The Halting Problem

Consider the following program. Does it terminate for all values of $n \geq 1$?

```java
while (n > 1) {
    if even(n) {
        n = n / 2;
    } else {
        n = n * 3 + 1;
    }
}
```

Not as easy to answer as it might first seem.

Say we start with $n = 7$, for example:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

In fact, for all numbers that have been tried (a lot!), it does terminate …

… but no one has ever been able to prove that it always terminates!

Example: The Halting Problem

It is impossible to write a program that decides if another, arbitrary, program terminates (halts) or not.

What might be surprising is that it is possible to prove such a result. This was first done by the British mathematician Alan Turing using Turing Machines.

Alan Turing (1912–1954):

- Introduced an abstract model of computation, Turing Machines, to give a precise definition of what problems that can be solved by a computer.
- Instrumental in the success of British code breaking efforts during WWII.
Noam Chomsky (1):

Noam Chomsky (1928–):
- American linguist who introduced Context Free Grammars in an attempt to describe natural languages formally.
- Also introduced the Chomsky Hierarchy which classifies grammars and languages and their descriptive power.

The Chomsky Hierarchy:

<table>
<thead>
<tr>
<th>Type 0 or recursively enumerable languages</th>
<th>Type 1 or context sensitive languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decidable languages</td>
<td>Type 2 or context free languages</td>
</tr>
<tr>
<td>Turing machines</td>
<td>pushdown automata</td>
</tr>
<tr>
<td>Type 3 or regular languages</td>
<td>finite automata</td>
</tr>
</tbody>
</table>

Languages:

The terms language and word are used in a strict technical sense in this course:
- A language is a (possibly infinite) set of words.
- A word is a finite sequence (or string) of symbols.
- $\epsilon$ denotes the empty word, the sequence of zero symbols.

The term string is often used interchangeably with the term word.

Symbols and Alphabets:

What is a symbol, then?

Anything, but it has to come from an alphabet $\Sigma$ which is a finite set.

A common (and important) instance is $\Sigma = \{0, 1\}$.

$\epsilon$, the empty word, is never a symbol of an alphabet.

Languages: Examples

alphabet $\Sigma = \{a, b\}$
- words: $\epsilon, a, b, aa, ab, ba, bb, aaaa, aab, aba, abb, baa, bab, \ldots$
- languages: $\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\}, \{\epsilon, a, aa, aaaa\}, \{a^n | n \geq 0\}, \{a^nb^n | n \geq 0, n \text{ even}\}$

Note the distinction between $\epsilon$, $\emptyset$, and $\{\epsilon\}$!

Exercises:

- Is the set of natural numbers, $\mathbb{N}$, a possible alphabet? Why/why not?
- What about the set of all natural numbers smaller than some given number $n$?
- Suggest an alphabet of a handful of drink ingredients. What are the symbols of your alphabet, and how many are they?
- List some words over your alphabet?
- What might an interesting language over your alphabet be? Does your language include all possible words over your alphabet?
- Is $\Sigma^*$ always infinite? Always non-empty?
An important operation on \( \Sigma^* \) is **concatenation**: given \( w, v \in \Sigma^* \), their concatenation \( wv \in \Sigma^* \).

For example, concatenation of \( ab \) and \( ba \) yields \( abba \).

This operation can be defined by primitive recursion:

\[
\epsilon v = v \\
(xw)v = x(wv)
\]

### Examples of Languages (1)

- The set \( \{0010, 00000000, \epsilon\} \) is a language over \( \Sigma = \{0, 1\} \).
  - This is an example of a **finite** language.
- The set of words with odd length over \( \Sigma = \{1\} \).
  - (Finite or infinite?)
- The set of words that contain the same number of 0s and 1s is a language over \( \Sigma = \{0, 1\} \).
  - (Finite or infinite?)

### Concatenation of Languages (1)

Concatenation of words is extended to languages by:

\[
MN = \{uv \mid u \in M \land v \in N\}
\]

Example:

\[
M = \{\epsilon, a, aa\} \\
N = \{b, c\} \\
MN = \{uv \mid u \in \{\epsilon, a, aa\} \land v \in \{b, c\}\} \\
= \{\epsilon b, c, ab, ac, aab, aac\} \\
= \{b, c, a, ab, ac, aab, aac\}
\]

### Concatenation of Languages (2)

- Concatenation of languages is associative:
  \[
  L(MN) = (LM)N
  \]
- Concatenation of languages has zero \( \emptyset \):
  \[
  L\emptyset = \emptyset = \emptyset L
  \]
- Concatenation of languages has unit \( \{\epsilon\} \):
  \[
  L\{\epsilon\} = L = \{\epsilon\} L
  \]

### Examples of Languages (2)

- The set of palindromes (words that read the same forwards and backwards, like \( abba \)) is a language for any alphabet.
- The set of correct Java programs. This is a language over the set of UNICODE characters.
- The set of programs that, if executed successfully on a Windows machine, prints the text “Hello World!” in a window. This is a language over \( \Sigma = \{0, 1\} \).

### Concatenation of Languages (3)

- Concatenation distributes through set union:
  \[
  L(M \cup N) = LM \cup LN \\
  (L \cup M)N = LN \cup MN
  \]
  But not through intersection! \( L(M \cap N) \neq LM \cap LN \)

  Counterexample: \( L = \{\epsilon, a\}, M = \{\epsilon\}, N = \{a\} \):
  \[
  L(M \cap N) = L\emptyset = \emptyset \\
  LM \cap LN = \{\epsilon, a\} \cap \{a, aa\} = \{a\}
  \]

### Concatenation of Languages (4)

- Exponent notation is used to denote iterated concatenation:
  \[
  L^1 = L \\
  L^2 = LL \\
  L^3 = LLL \\
  \ldots
  \]
- By definition: \( L^0 = \{\epsilon\} \) (for any language, incl. \( \emptyset \))

\[
L^* = \bigcup_{n=0}^{\infty} L^n
\]
Language Membership

Fundamental question for a language $L$: $w \in L$?

- $L$ finite: Easy! (Enumerate $L$ and check)
- $L$ infinite: ?

We need:
- A finite (and preferably concise) formal description of $L$.
- An algorithmic method to decide if $w \in L$ given a suitable description.

Various approaches to achieve this will be key a theme throughout the module.