

G51MAL Machines and Their Languages Lecture 1

Administrative Details and Introduction

Henrik Nilsson

University of Nottingham, UK

G51MALMachines and Their LanguagesLecture 1 – p.1/38

Contacting Me

- I will be available immediately after each lecture for course-related matters.
- Make an appointment if necessary.
- Please keep e-mail traffic to a minimum.

G51MALMachines and Their LanguagesLecture 1 – p.4/38

Literature

- Main reference: Hopcroft, Motwani, & Ullman. *Introduction to Automata Theory, Languages, and Computation, 2nd edition*, Addison Wesley, 2001.
- Dr. Thorsten Altenkirch's G51MAL updated lecture notes.
- Your own notes from the lectures!
- Possibly a new version of the lecture notes later.
- Supplementary material, e.g. slides, sample program code.

G51MALMachines and Their LanguagesLecture 1 – p.7/38

Finding People and Information (1)

- Henrik Nilsson
Room B47, Computer Science Building
e-mail: nhn@cs.nott.ac.uk
tel: 0115 846 6506
- Teaching Assistants:
 - Ondrej Rypacek
e-mail: oxr@cs.nott.ac.uk
 - Wouter Swierstra (head TA)
e-mail: wss@cs.nott.ac.uk

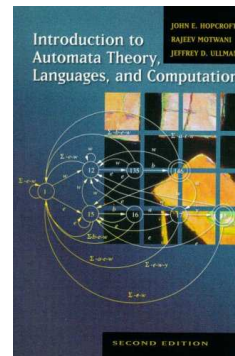
G51MALMachines and Their LanguagesLecture 1 – p.2/38

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, e.g. for
 - Compiler Construction
 - Text Processing
 - XML

G51MALMachines and Their LanguagesLecture 1 – p.5/38

Literature (2)



G51MALMachines and Their LanguagesLecture 1 – p.8/38

Finding People and Information (2)

- Main module web page:
www.cs.nott.ac.uk/~nhn/G51MAL
- Coursework/Tutorial web page:
www.cs.nott.ac.uk/~wss/teaching/mal

G51MALMachines and Their LanguagesLecture 1 – p.3/38

Organization

- **Lectures:** Two per week.
- **Tutorials:** Weekly in small (≈ 15 students) groups.
You are expected to participate regularly!
- **Coursework:** Weekly **compulsory** exercises. Marked and then discussed during tutorials.
- **Assessment:** 2 hour exam in May/June, 100% of the mark.

G51MALMachines and Their LanguagesLecture 1 – p.6/38

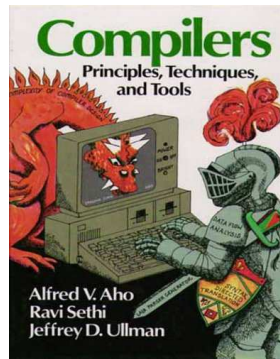
Literature (3)

If you are curious about an important application area you might want to check out:

Alfred V Aho, Ravi Sethi, Jeffrey D. Ullman. *Compilers — Principles, Techniques, and Tools*, Addison-Wesley, 1986. (The “Dragon Book”.)

G51MALMachines and Their LanguagesLecture 1 – p.9/38

Literature (4)



G51MALMachines and Their LanguagesLecture 1 – p.10/38

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.

G51MALMachines and Their LanguagesLecture 1 – p.11/38

Why Study Automata Theory?

Finite automata are a useful model for important kinds of hardware and software:

- Software for designing and checking digital circuits.
- Lexical analyzer of compilers.
- 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.

G51MALMachines and Their LanguagesLecture 1 – p.12/38

Why Study Automata Theory? (2)

The study of Finite Automata and Formal Languages are intimately connected. Methods for specifying formal languages are very important in many areas of CS, e.g.:

- **Context Free Grammars** are very useful when designing software that processes data with recursive structure, like the parser in a compiler.
- **Regular Expressions** are very useful for specifying lexical aspects of programming languages and search patterns.

G51MALMachines and Their LanguagesLecture 1 – p.13/38

Why Study Automata Theory? (3)

Automata are essential for the study of the limits of computation. Two issues:

- What can a computer do at all? (Decidability)
- What can a computer do efficiently? (Intractability)

G51MALMachines and Their LanguagesLecture 1 – p.14/38

Example: Regular Expressions (1)

Suppose you need to locate a piece of text in a directory containing a large number of files of various kinds. You recall only that the text mentions the year 1900-something.

The following UNIX-command will do the trick:

```
grep "19[0-9][0-9]" *.txt
```

G51MALMachines and Their LanguagesLecture 1 – p.15/38

Example: Regular Expressions (2)

The result is a list of names of files containing text matching the pattern, together with the matching text lines:

```
history.txt: In 1933 it became
notes.txt:  later on, around 1995,
```

G51MALMachines and Their LanguagesLecture 1 – p.16/38

Example: The Halting Problem (1)

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

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

G51MALMachines and Their LanguagesLecture 1 – p.17/38

Example: The Halting Problem (2)

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!

G51MALMachines and Their LanguagesLecture 1 – p.18/38

Example: The Halting Problem (3)

Then the following important decidability result should perhaps not come as a total surprise:

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**.

G51MALMachines and Their LanguagesLecture 1 – p.19/38

Alan Turing (1)

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.
- Thorsten recommends Andrew Hodges biography *Alan Turing: the Enigma*.

G51MALMachines and Their LanguagesLecture 1 – p.20/38

Alan Turing (2)



G51MALMachines and Their LanguagesLecture 1 – p.21/38

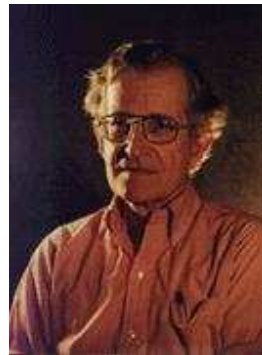
Noam Chomsky (1)

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.
- Chomsky is also widely known for his controversial political views and his criticism of the foreign policy of U.S. governments.

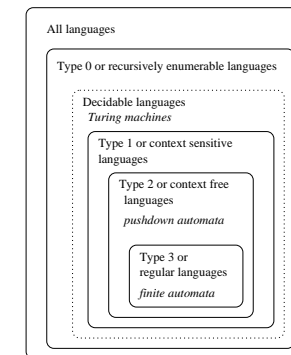
G51MALMachines and Their LanguagesLecture 1 – p.22/38

Noam Chomsky (2)



G51MALMachines and Their LanguagesLecture 1 – p.23/38

The Chomsky Hierarchy



G51MALMachines and Their LanguagesLecture 1 – p.24/38

Languages

The terms *language* and *word* are used in a strict technical sense in this course:

- A **language** is a set of words.
- A **word** is a sequence (or string) of symbols.

ϵ denotes the **empty word**, the sequence of zero symbols.

G51MALMachines and Their LanguagesLecture 1 – p.25/38

Symbols and Alphabets

What is a symbol, then?

Anything, but it has to come from an **alphabet** Σ which is a **finite** set.

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

ϵ , the empty word, is **never** an symbol of an alphabet.

G51MALMachines and Their LanguagesLecture 1 – p.26/38

Alphabet, Word, and Language

alphabet	$\Sigma = \{a, b\}$
words over Σ	$\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, \dots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\}, \{\epsilon, a, aa, aaa\}, \{a^n n \geq 0\}, \{a^n b^n n \geq 0, n \text{ even}\}$

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

G51MALMachines and Their LanguagesLecture 1 – p.27/38

All Words over an Alphabet (1)

Given an alphabet Σ we define the set Σ^* as set of words (or sequences) over Σ :

- The empty word $\epsilon \in \Sigma^*$.
- given a symbol $x \in \Sigma$ and a word $w \in \Sigma^*$, $xw \in \Sigma^*$.
- These are all elements in Σ^* .

This is called an **inductive definition**.

G51MALLMachines and Their LanguagesLecture 1 – p.28/38

All Words over an Alphabet (2)

Example: Given $\Sigma = \{0, 1\}$, some elements of Σ^* are

- ϵ (the empty word)
- 0, 1
- 00, 10, 01, 11
- 000, 100, 010, 110, 010, 110, 011, 111
- ...

We are just applying the inductive definition.

Note: although there are infinitely many words in Σ^* , each word has a **finite** length!

G51MALLMachines and Their LanguagesLecture 1 – p.29/38

Concatenation of Words (2)

Concatenation is associative and has unit ϵ :

$$u(vw) = (uv)w$$
$$\epsilon u = u = u\epsilon$$

where u, v, w are words.

G51MALLMachines and Their LanguagesLecture 1 – p.31/38

Languages Revisited

The notion of a language L of a set of words over an alphabet Σ can now be made precise:

- $L \subseteq \Sigma^*$, or equivalently
- $L \in \mathcal{P}(\Sigma^*)$.

G51MALLMachines and Their LanguagesLecture 1 – p.32/38

Examples of Languages (2)

- The set of words which contain the same number of 0s and 1s modulo 2 (i.e. both are even or odd) is a language over $\Sigma = \{0, 1\}$.
- The set of palindromes using the English alphabet, e.g. words which read the same forwards and backwards like `abba`. This is a language over $\{a, b, \dots, z\}$.
- The set of correct Java programs. This is a language over the set of UNICODE characters.

G51MALLMachines and Their LanguagesLecture 1 – p.34/38

Examples of Languages (3)

- 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 $\Sigma = \{0, 1\}$.

G51MALLMachines and Their LanguagesLecture 1 – p.35/38

Concatenation of Words (1)

An important operation on Σ^* 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:

$$\epsilon v = v$$
$$(xw)v = x(wv)$$

G51MALLMachines and Their LanguagesLecture 1 – p.30/38

Examples of Languages (1)

Some examples of languages:

- 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 $\Sigma = \{1\}$.
- The set of words that contain the same number of 0s and 1s is a language over $\Sigma = \{0, 1\}$.

G51MALLMachines and Their LanguagesLecture 1 – p.33/38

Concatenation of Languages (1)

Concatenation of words is extended to languages by:

$$MN = \{uv \mid u \in M \wedge v \in N\}$$

Example:

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

G51MALLMachines and Their LanguagesLecture 1 – p.36/38

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 $\{\epsilon\}$:

$$L\{\epsilon\} = L = \{\epsilon\}L$$

Concatenation of Languages (3)

- Concatenation distributes through set union:

$$L(M \cup N) = LM \cup LN$$

$$(L \cup M)N = LN \cup MN$$

But note e.g. $L(M \cap N) \neq LM \cap LN!$

For example, with $L = \{\epsilon, a\}$, $M = \{\epsilon\}$, $N = \{a\}$, we have

$$L(M \cap N) = L\emptyset = \emptyset$$

$$LM \cap LN = \{\epsilon, a\} \cap \{a, aa\} = \{a\}$$