G52MAL
Machines and their Languages
Lecture 1: Administrative Details and Introduction

Thorsten Altenkirch
based on slides by Neil Sculthorpe

Room A10
School of Computer Science
University of Nottingham
United Kingdom
txa@cs.nott.ac.uk

1st February 2012

Module Prerequisites

- G51FUN: Functional Programming
- G51MCS: Mathematics for Computer Scientists
- G51PRG: Programming

Finding People and Information (1)

- Lecturer: Thorsten Altenkirch
  Location: Room A10, Computer Science Building
  e-mail: `txa@cs.nott.ac.uk`
- Tutors
  - Laurence Day, Room A04, CS, `led@cs.nott.ac.uk`
  - Bas van Gijzel, Room A04, CS, `bmv@cs.nott.ac.uk`

Finding People and Information (2)

- Module webpage:
  `www.cs.nott.ac.uk/~txa/g52mal`
- Coursework support page:
  `www.cs.nott.ac.uk/~txa/g52mal/cswk.html`
- Web Forum:
  `webservices.cs.nott.ac.uk/forum`

Contacting Us

- The best time to come and talk to me is immediately after a lecture.
- Failing that, you can come and see me in my office; but I can't guarantee that I'll be available.
- The best way to seek coursework help is on the web forum.
- If you don't want to post on the forum for some reason, then you can email: `g52mal-tutors@cs.nott.ac.uk`

Lectures

- Thursday, 9am, A25, Business School South
- Fridays, 2pm, LT3, Business School South
- Mixture of slides and whiteboard — you are expected to take notes!

Tutorials

- Attending tutorials is compulsory and essential for coursework.
- Tutorials provide an excellent opportunity to ask questions! Use it!
- Coursework handin and feedback takes place in the tutorial. In person!
- Tutorial groups on Wednesday: 9-10, 10-11 or 14-15.
- You will be assigned a group, but you can change with good reason.
- Tutorials start on Wednesday, 15 February

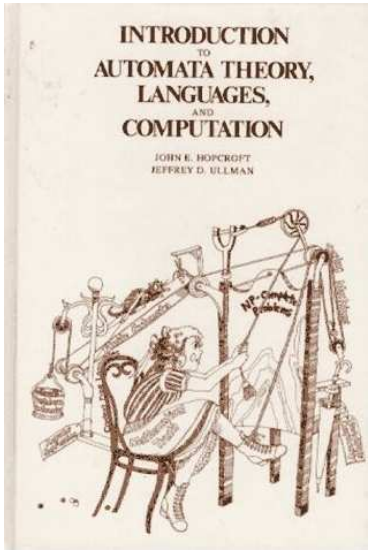
Coursework

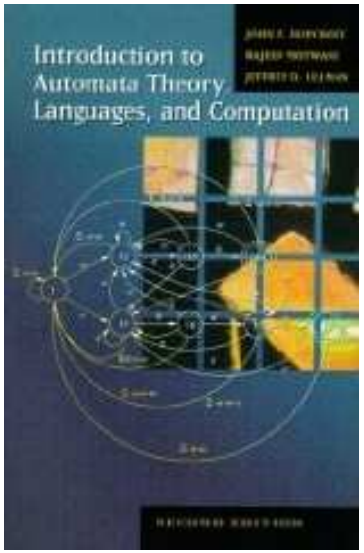
- 8 weekly problem sets.
- The best 5 solutions count.
- Made available via the module web page.
- First coursework available: Friday 10th February
- First deadline: Wednesday 22nd February, in your tutorial!
- Late submission: Thursday at the school office (- 10 %).
- Tutor may ask for clarification! You may loose points if you cannot answer! Or if you miss your tutorial.
- Each coursework will contain a bonus exercise.

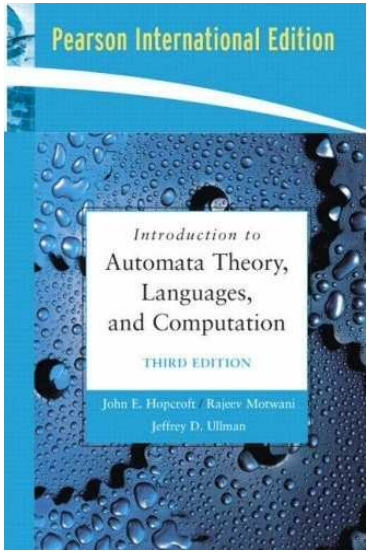
Assessment

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

Literature



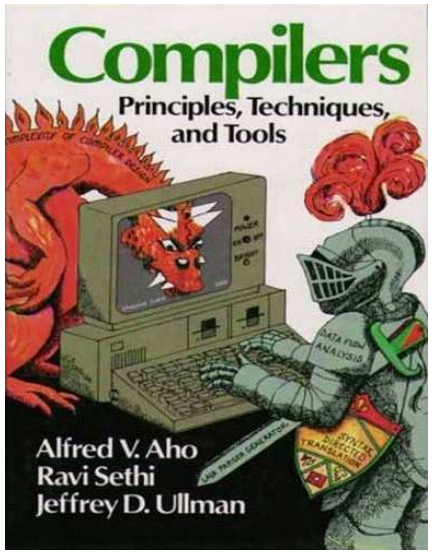




- Main reference:
  Introduction to Automata Theory, Languages, and Computation; 2nd or 3rd edition; Hopcroft, Motwani and Ullman; (1979, 2000 & 2006).
- *G52MAL Lecture Notes*, Altenkirch and Nilsson, 2007. Available via the G52MAL module web-page.
- Your own notes from the lectures!

Compilers

- Topics covered in G52MAL have important applications in the area of *compiler construction*.
- In fact, G52MAL feeds directly into next year's Compilers module.
- If you are curious you might want to check out "The Dragon Book": *Compilers — Principles, Techniques, and Tools*; Aho, Sethi & Ullman; 1986.



Aims of the Course

- To familiarise you with key Computer Science concepts in central areas such as:
  - Automata Theory
  - Formal Languages
  - Models of Computation
- This includes:
  - the equivalence between machine types and language types;
  - the specification of formal languages by grammars and other notations;
  - the relevance of machines that process strings from an alphabet as models of computation;
  - and the limits on what is computable by any machine.

Content

- Mathematical models of computation, such as:
  - Finite Automata
  - Pushdown Automata
  - Turing Machines
- How to specify formal languages?
  - Regular Expressions
  - Context Free Grammars
  - Context Sensitive Grammars
- The relationship between the two.

Why Study Automata Theory? (1)

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

- Software for designing and checking digital circuits.
- Lexical analyser 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).

Why Study Automata Theory? (2)

The study of Automata and Formal Languages are intimately connected. Methods for specifying formal languages are very important in many areas of Computer Science, for example:

- 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.

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)

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 19-something.

The following UNIX-command will do the trick:

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

The command involves TWO (extended) regular expressions.

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,
```

How can we write software that quickly searches for this kind of patterns?

Not as easy as it seems!

You will learn the basic answer in this module.

Example: The Halting Problem (1)

Consider the following program:

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

Does it terminate for all values of $n \geqslant 1$? (where $n \in \mathbb{N}$)

Example: The Halting Problem (2)

For example, say we start with $n = 7$:

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

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 (3)

Thus 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 Alan Turing.

Alan Turing

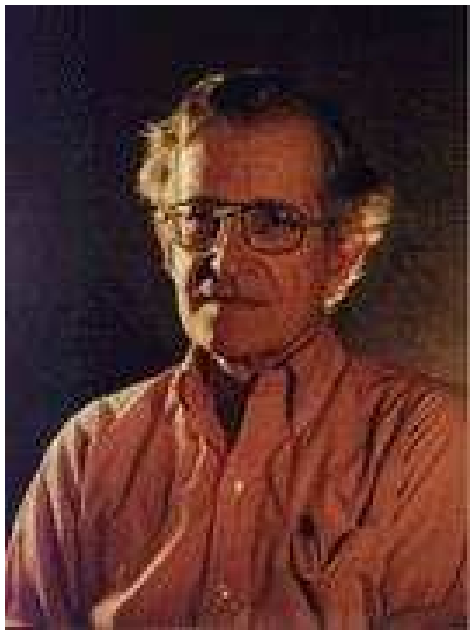
Alan Turing (1912–1954):

- British mathematician.
- Introduced an abstract model of computation, Turing Machines, to give a precise definition of what problems can be solved by a computer.
- Also famous for being instrumental in the success of British code-breaking efforts during the Second World War.

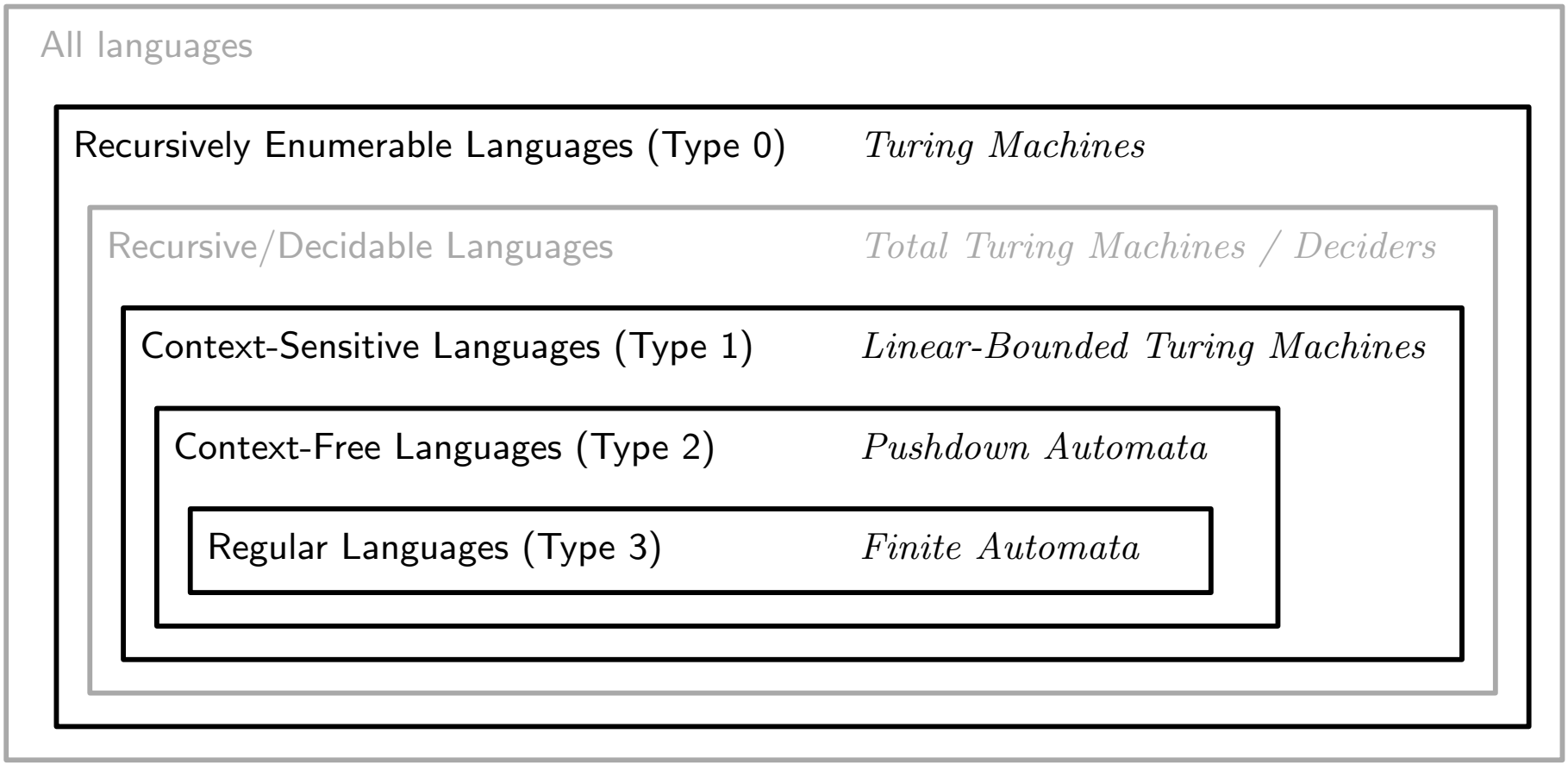
Noam Chomsky

Noam Chomsky (1928–):

- American linguist.
- Introduced Context Free Grammars in an attempt to describe natural languages formally.
- Introduced the Chomsky Hierarchy which classifies grammars and languages, and their descriptive power.
- Also famous for his political activism.



The Chomsky Hierarchy



Recommended Reading

- Introduction to Automata Theory, Languages, and Computation (3rd edition), pages 1–5.
- G52MAL Lecture Notes, pages 2–5.