COMP30026 Models of Computation

Introduction and Welcome

Harald Søndergaard

Lecture 1

Semester 2, 2017

Welcome to COMP30026 Models of Computation

Teaching staff:

Lectures and subject coordination:

- Harald Søndergaard

Tutors:

- Matt Farrugia-Roberts (Head Tutor)
- Rob Holt
- Les Kitchen
- Dongge Liu
- Mak Nazecic-Andrlon
- Julian Tran

The COMP30026 LMS site

- Week-by-week plan for lectures, tutes, assignments, ...
- Important announcements.
- A Subject Guide with information about assignments, exam, contact, and much more, providing more detail than the Handbook entry.
- Our discussion boards.
- Reading Resources covering the topics in this subject.
- Grok modules for Haskell.

COMP30026 Texts

Freely available as e-books through the library:

- O'Donnell, Hall and Page. *Discrete Mathematics Using a Computer*. Springer, 2006.
- Makinson. *Sets, Logic and Maths for Computing*. Springer, 2008.
- Singh. *Elements of Commutation Theory*. Springer, 2009.
- Parkes. *A Concise Introduction to Languages and Machines*. Springer, 2008.

Other:

- Doets and van Eijck. *The Haskell Road to Logic, Maths and Programming*. King's College Publ., 2004. Two copies in ERC.
- Sipser. *Introduction to the Theory of Computation*. Thomson, 2012. (Good but expensive!)

Chapter 1 from Sipser is in Readings Online anyway.

Lectures and Tutes

Lectures are on Tuesday and Fridays, in the Charles Pearson Theatre.

The lectures are recorded and made accessible via the LMS site (but read the disclaimer).

The slides used in lectures will be there as well.

Tutorials are very important in this subject. They start in Week 2.

Each week, by Thursday, I will try to have next week's tutorial exercises posted, on the LMS.

Other Support

Matt offers face-to-face consultation each Monday 15:20–16:10.
That happens in Doug McDonell 702.

But (preferably): Use the LMS's Discussion Forum.

Over to You—Introductions

Please introduce yourself to your neighbours.

Tell them where you are from, what degree program you are enrolled in, something about languages or programming languages that you speak, or anything else that is interesting.

Why Study Logic and Discrete Maths?

Logic and discrete mathematics provide the theoretical foundations for computer science.

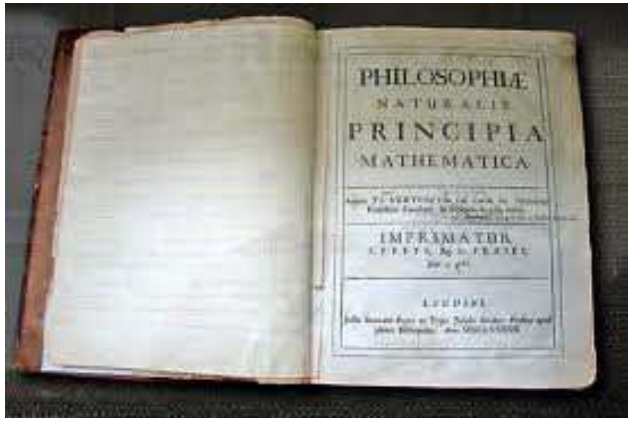
- Propositional logic has applications in hardware and software verification, planning, testing and fault finding, ...
- Predicate logic is essential to artificial intelligence, computational linguistic, automated theorem proving, logic programming, ...
- Algebra underpins theories of databases, programming languages, program analysis, data mining, ...

Why Study Logic and Discrete Maths?

- Integers and rational numbers: Planners and constraint solvers, operations research.
- Modular arithmetic, number theory: Verification, encryption and decryption.
- Graphs and trees: Efficient data structures and their algorithms; information retrieval.
- Finite-state automata: Controllers/counters, pattern matching, computational linguistics.
- Formal languages, grammars: Parsing, semantics, language-based security.

Discrete vs Continuous Mathematics

discrē'te *a*. separate, individually distinct, discontinuous



There is traditionally a strong emphasis on continuous mathematics (calculus, analysis) in science and engineering education.

As the world has gone digital, discrete mathematics has become more important, not only for engineering disciplines, but for all the disciplines that now utilise "computational" thinking: computational biology, computational linguistics, computational neuroscience, ...

Hybrid systems combine discrete and continuous behaviour (robotics, x-by-wire).

Why Study Automata and Formal Languages?

Primarily because of the applications.

But there's an aesthetic dimension too: the theories of different language classes are very pleasing—if you like maths, you will like regular algebra, for example, for its beauty.

Moreover, automata theory is a perfect stepping stone on the way to computability theory. Important problems (that are simple to state) which arise from dealing with grammars and automata will provide perfect examples of decidability and undecidability.

Why Study Computability?

Because it is important to have an understanding of the limitations of algorithmic problem solving.

It is important to know that there are simple functions that are not computable.

It is important to know that there are simple questions that are not decidable.

Computability is full of philosophically challenging ideas, and exciting (sometimes surprising) results.

Topics Covered in COMP30026

- Propositional and predicate logic;
- Sets, functions, and relations;
- Proof principles, induction and recursion, well-ordering;
- Regular languages, finite-state automata, context-free languages;
- Computability and decidability.

This subject also involves programming in that beautiful functional programming language, Haskell.

Sometimes logic/mathematical concepts become clearer when we implement them and play around with them that way.

Further Ahead

Two very important fields of theoretical computer science:

- Computability Theory (which kinds of problems can and cannot be solved with different types of computing devices?)
- Complexity Theory (what is the inherent hardness of different kinds of computational problems?)

We will cover elementary computability theory, but sadly we will not have time for complexity theory.

Complexity theory (and computability theory in more detail) are covered in COMP90057 Advanced Theoretical Computer Science.

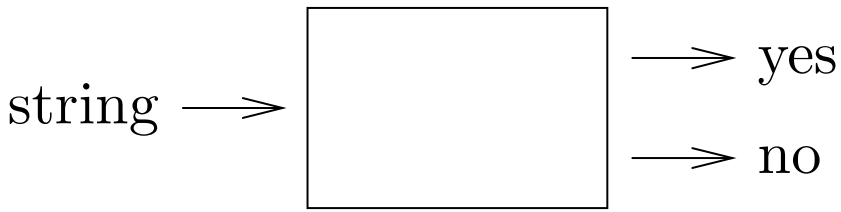
Formal Languages

Closely related to the theory of computation.

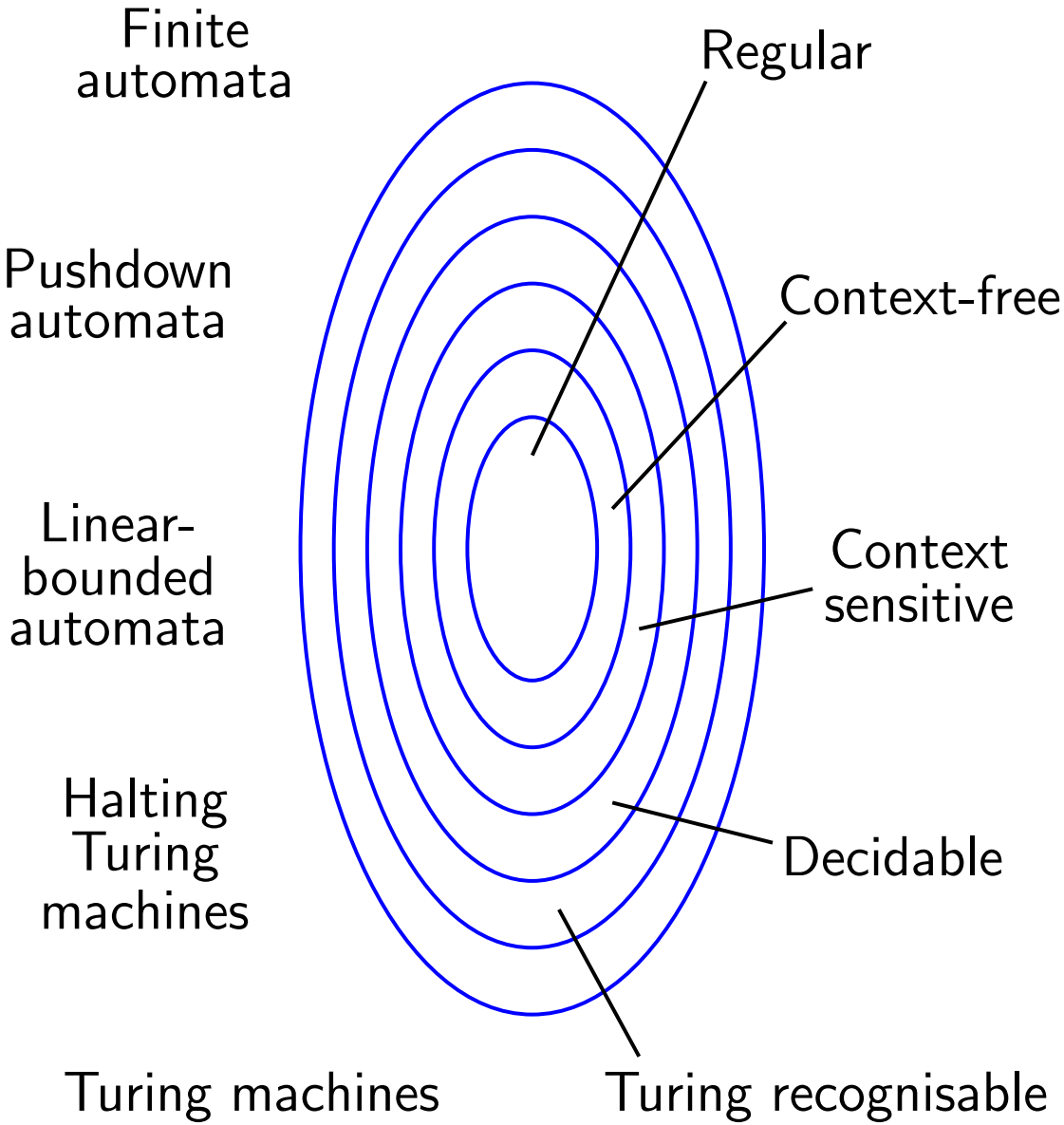
Here: language = set of strings.

We can classify languages according to what sort of rules (grammars) are allowed in describing how strings are generated.

We can then consider a machine's ability to act as a recogniser for a language.



Machines vs Languages



Some of the Objectives

- Use fundamental concepts and formalisms for reasoning about computation;
- Reason formally about models of computation, simple specifications and programs;
- Apply discrete mathematical techniques to problems in computer science;
- Design state machines for a range of computational problems.

Theory and Practice

"There is nothing more practical than a good theory."

Kurt Lewin

"The theory I do gives me the vocabulary and the ways to do practical things that make giant steps instead of small steps when I'm doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they're just purely inspired by other theories. There's this symbiotic relationship . . ."

Donald Knuth

Intro Puzzle

Huey, Dewey and Louie are being questioned by their uncle. Here is what they say:

> *Huey: "Dewey and Louie had equal share in it; if one is guilty, so is the other."*
> *Dewey: "If Huey is guilty, then so am I."*
> *Louie: "Dewey and I are not both guilty."*

Their uncle, knowing that they are cub scouts, realises that they cannot tell a lie.

Has he got sufficient information to decide who (if any) are guilty?

Next Up

The tool that we need to solve this puzzle is propositional logic.

Next Tuesday we will get started on that topic. To prepare, read the relevant chapter from O'Donnell, Hall and Page, or alternatively, Makinson's chapter 8.

On Friday I will give a Cook's tour of Haskell. To prepare, try out `https://tryhaskell.org/` and check out other resources from `https://www.haskell.org/documentation`.