COMP30026 Models of Computation

Setting the Stage

Harald Søndergaard

Lecture 1

Semester 2, 2018

Welcome to COMP30026 Models of Computation

Teaching staff:

Lectures and subject coordination:

- Harald Søndergaard

Head tutor:

- Matt Farrugia-Roberts

Tutors:

- Keith Lee, Max Lee, Dongge Liu, Yang Lu, Dalzy Mendoza, Victor Phan, 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 Mondays and Tuesdays, in the Derham Theatre.

The lectures are recorded and made accessible via the LMS site. Note, however, that the quality of recordings will vary, and some lecture activities cannot be captured.

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

By Thursday each week, expect the next week's tutorial exercises to be posted on the LMS.

To get value out of the tutes, please please turn up well prepared.

Other Support

Harald has a consultation hour most Thursdays from 12:00 (Doug McDonell 8.16).

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.

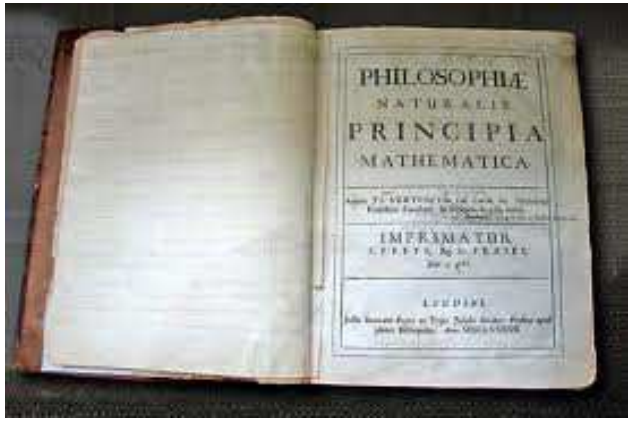
- Propositional logic has applications in hardware and software verification, planning, testing and fault finding, ...
- Predicate logic is essential to artificial intelligence, computational linguistic, automated theorem proving, logic programming, ...
- Algebra underpins theories of databases, programming 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 if "computation" is our business, we need to a good understanding of what it even means.

It allows us to grasp 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.

Let's Play a Game

Calculation, or computation, is ultimately manipulation of (combinations of) symbols. Consider these three symbols: I, M, U, and a string rewriting game with the following rules:

1:	Mx	$\Rightarrow$	Mxx
2:	xI	$\Rightarrow$	xIU
3:	xIIIy	$\Rightarrow$	xUy
4:	xUUy	$\Rightarrow$	xy

(The left-hand sides are string patterns using string variables x and y.)

The challenge is to find, for two given strings s and t, a sequence of rewrite steps that, starting from s, take us to t.

Your concrete challenge: Starting from MI, generate MU.

Let's Play a Game

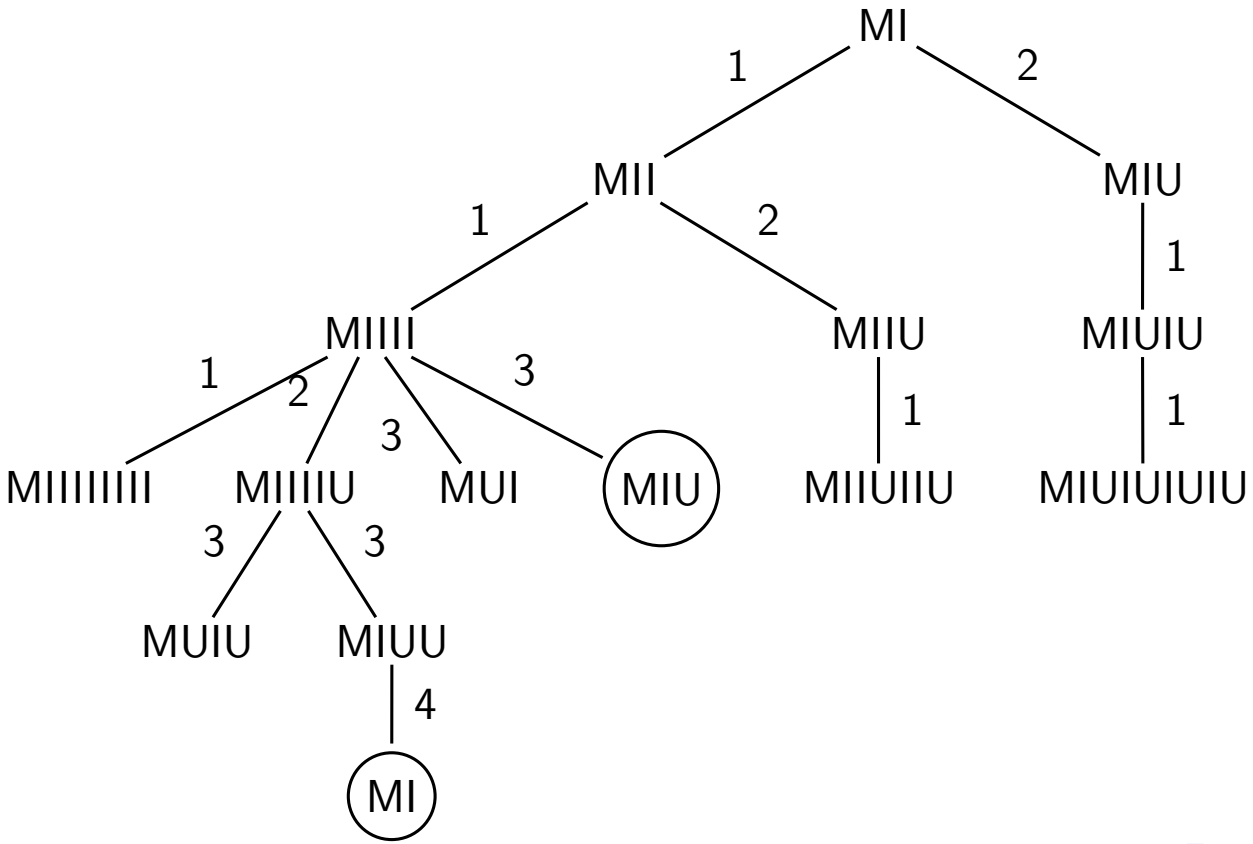
1: $Mx \Rightarrow Mxx$
2: $xI \Rightarrow xIU$
3: $xIIIy \Rightarrow xUy$
4: $xUUy \Rightarrow xy$

$MI \stackrel{*}{\Rightarrow} MU?$

Let's Play a Game

1: $Mx \Rightarrow Mxx$
2: $xI \Rightarrow xIU$
3: $xIIIy \Rightarrow xUy$
4: $xUUy \Rightarrow xy$

$MI \overset{*}{\Rightarrow} MU?$



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.

You will learn the basics of a beautiful functional programming language, Haskell.

We want to use Haskell to reinforce the learning of logic and discrete maths concepts.

Nobody learns programming from lectures; instead we will use an interactive learning environment called Grok.

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.

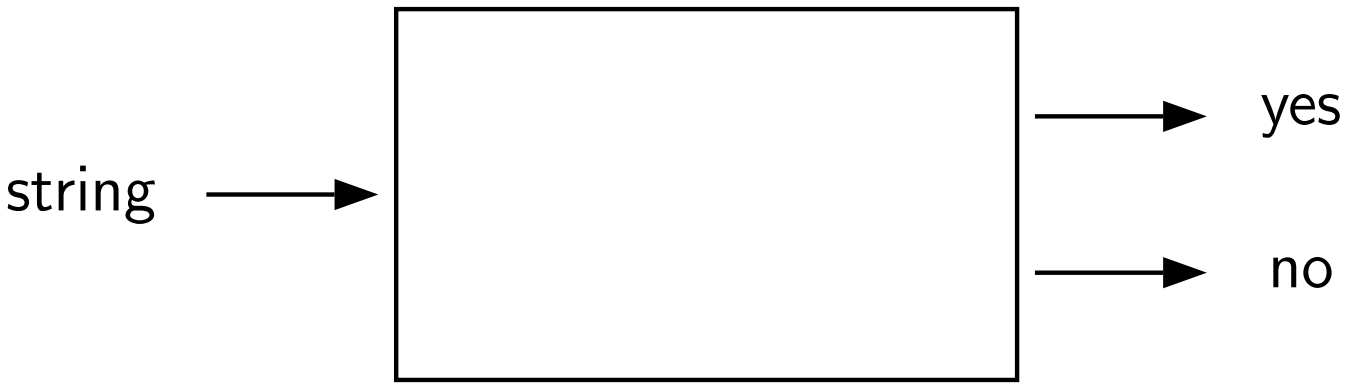
Formal Languages

Closely related to the theory of computation.

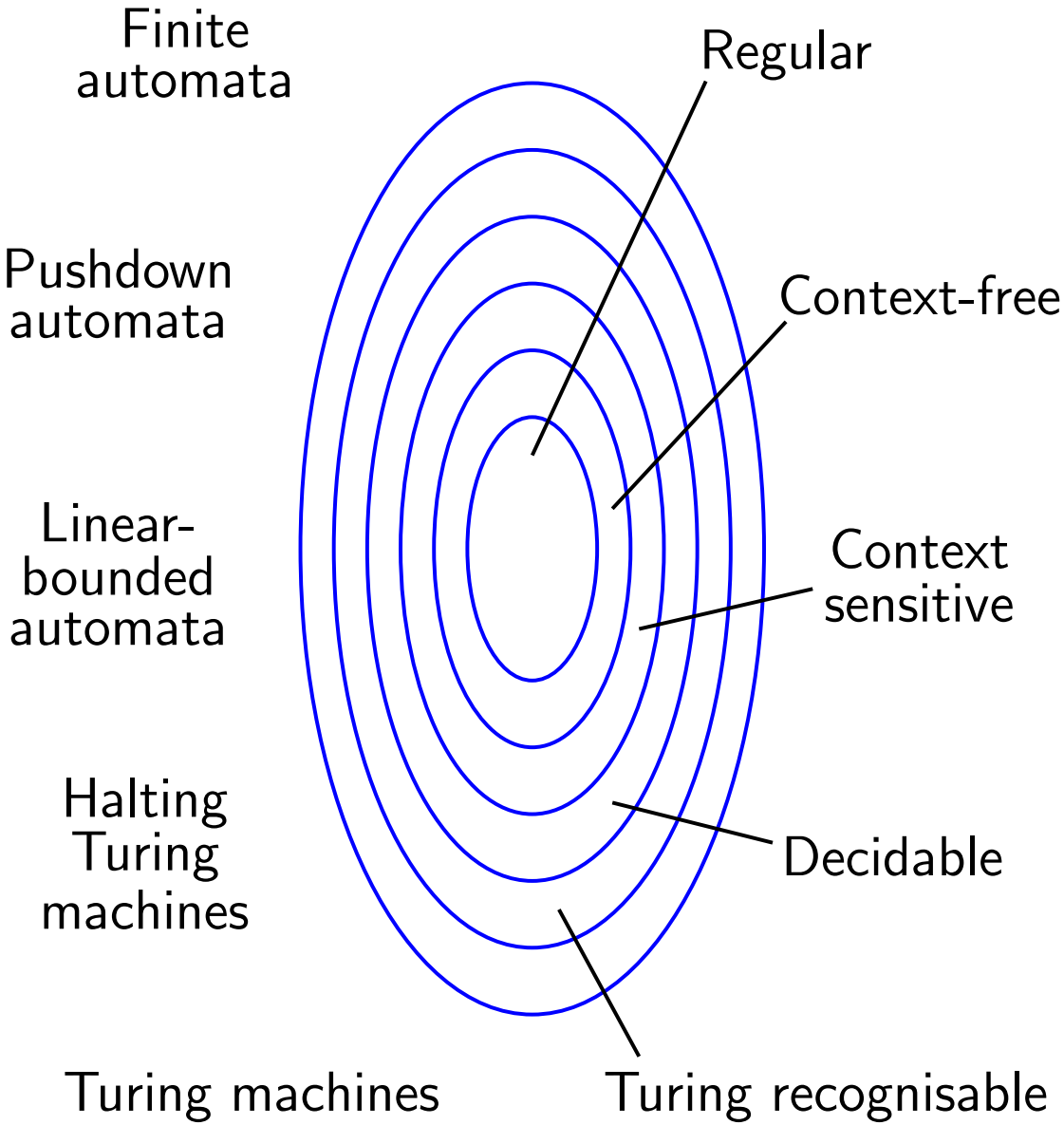
Here: language = set of strings.

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

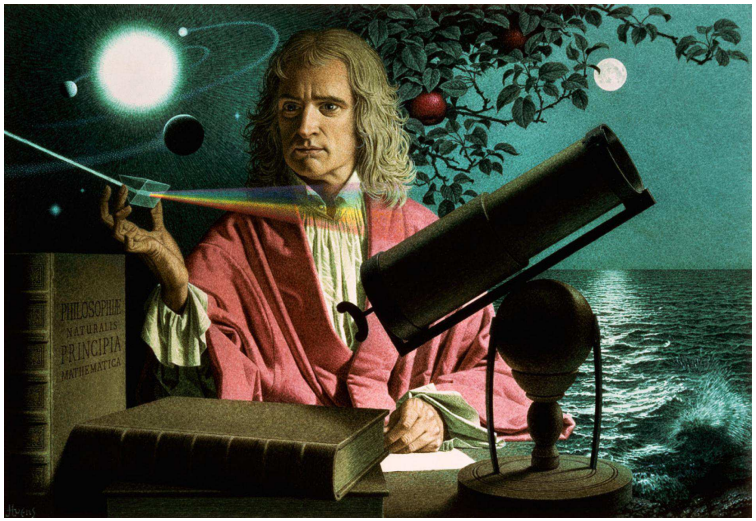
We can then consider a machine's ability to act as a recogniser for a language.



Machines vs Languages



Models and Their Use



We model aspects of the world to understand it better.

To understand how an aircraft will behave in different weather conditions, engineers may use a simplified representation in a wind-tunnel.

This involves abstraction.

The usefulness of a model is that it allows us to reason hypothetically and to predict events.

However, a good (scientific) model often adds something even more important: It can help explain the phenomena we observe, or it can reveal unexpected links between phenomena.

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

Coming Up ...

... A non-introduction to Haskell.

We'll play around with some code to get a feel for the language.

To learn basic Haskell (as you should do as soon as possible), get started on the Grok modules.

If you want a head start before tomorrow's lecture, browse the resources at `https://www.haskell.org/documentation`.

And Finally: An Ad from UROP