COMP30026 Models of Computation

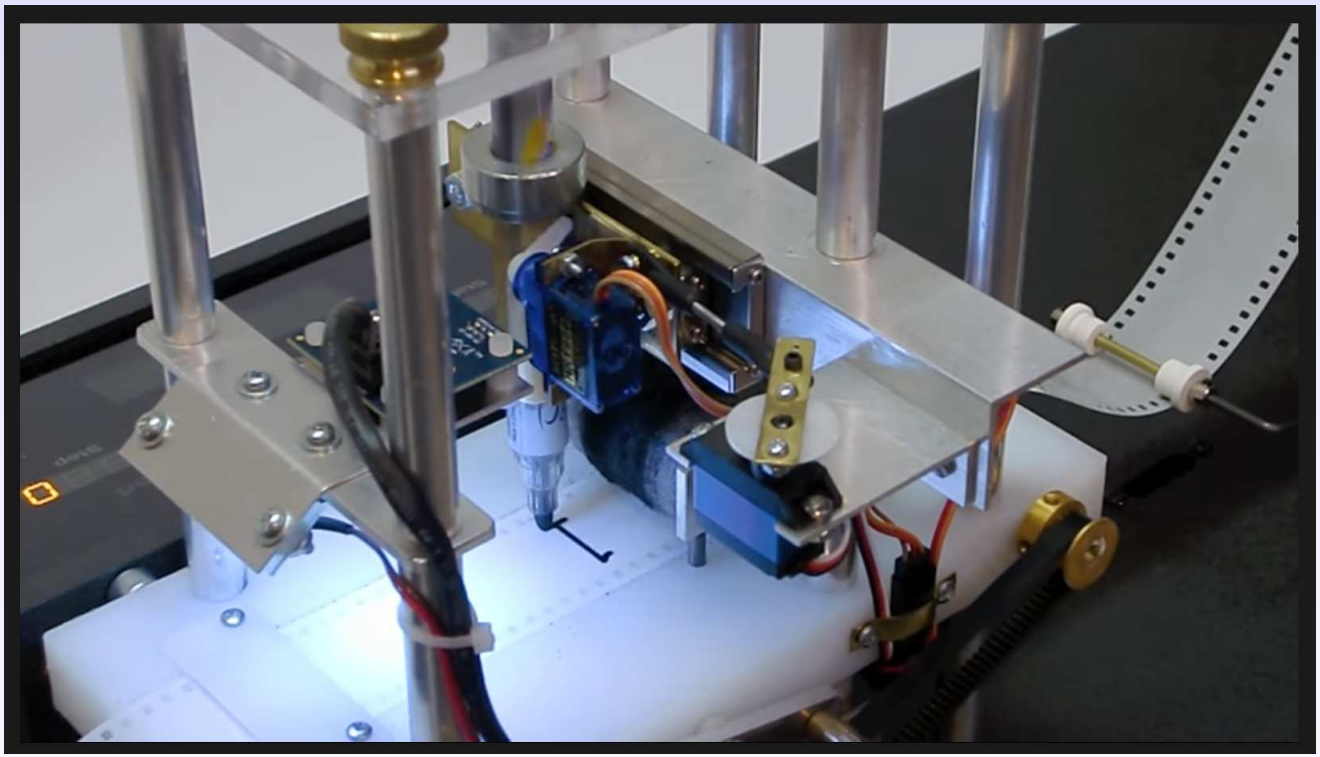
Setting the Stage

Harald Søndergaard

Lecture 1

Semester 2, 2020

COMP30026 Models of Computation



Topics Covered in COMP30026

- Propositional and predicate logic;
- Sets, functions, and relations; proof principles;
- Regular languages, finite-state automata, context-free languages;
- Computability and decidability.

You will learn the basics of a beautiful functional programming language, Haskell. Nobody learns programming from lectures; instead we use an interactive learning environment called Grok.

We use Haskell to reinforce the learning of concepts that we study.

Why Study Logic and Discrete Maths?

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

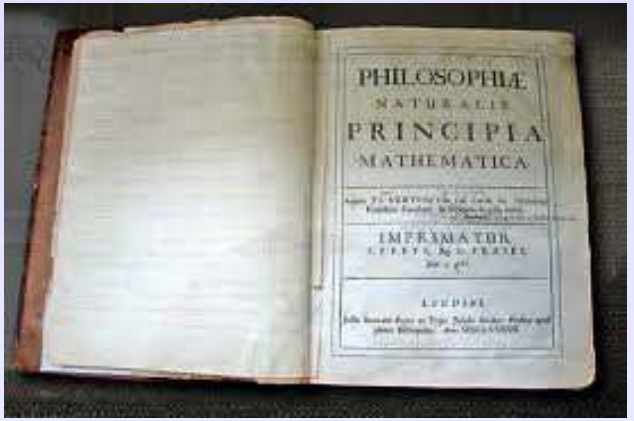
- Propositional logic has applications in hardware and software verification, planning, testing and fault finding, ...
- Predicate logic is essential to artificial intelligence, computational linguistic, automated theorem proving, logic programming, ...
- Algebra underpins theories of databases, programming 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 gained importance, not only for engineering disciplines, but for all those 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?

For one thing, because of the many 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 and easy-to-state problems which arise from dealing with automata and grammars provide excellent examples of decidability and undecidability.

Why Study Computability?

Because if "computation" is your business, you really should have a solid understanding of what it means.

It allows us to grasp the limitations of algorithmic problem solving.

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

There are important problems that do not have algorithmic solutions.

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

Further Ahead in Your Study

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 only cover elementary computability theory, and 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.

Further Ahead in Your Working Life

Technological innovations appear with dizzying speed.

But while the practice of computing changes fast, the theoretical underpinnings of computation change very slowly.

The tools and concepts we study here will remain relevant in the foreseeable future.

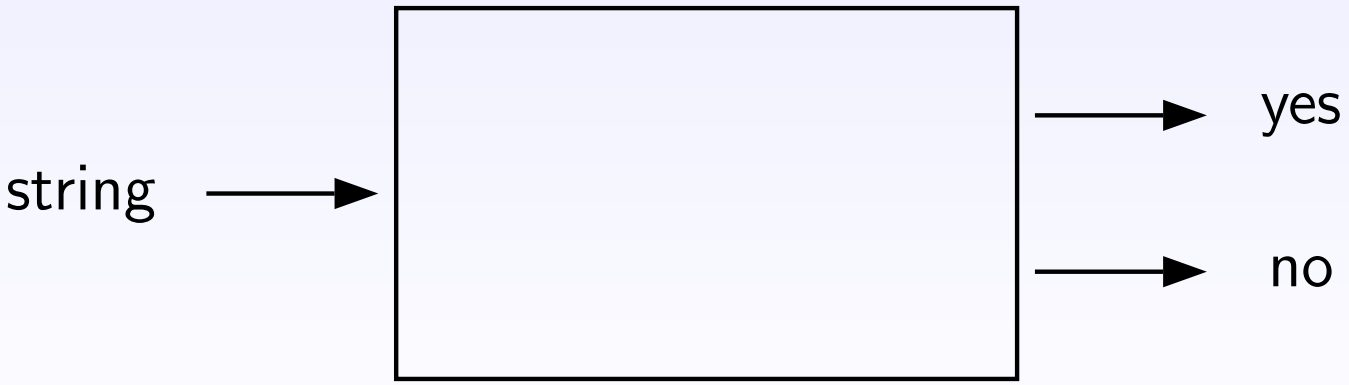
Formal Languages

Closely related to the theory of computation.

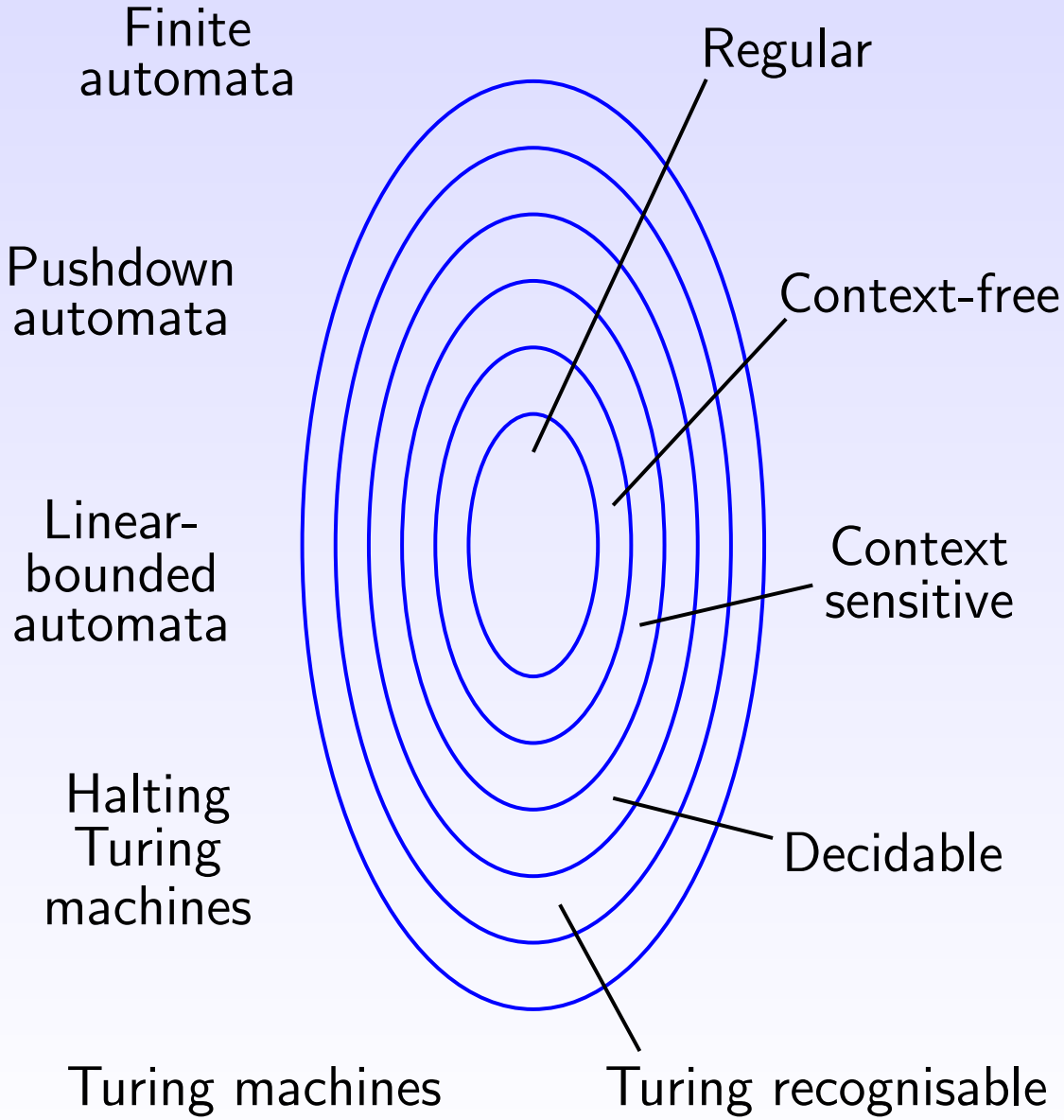
Here: language = set of strings.

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

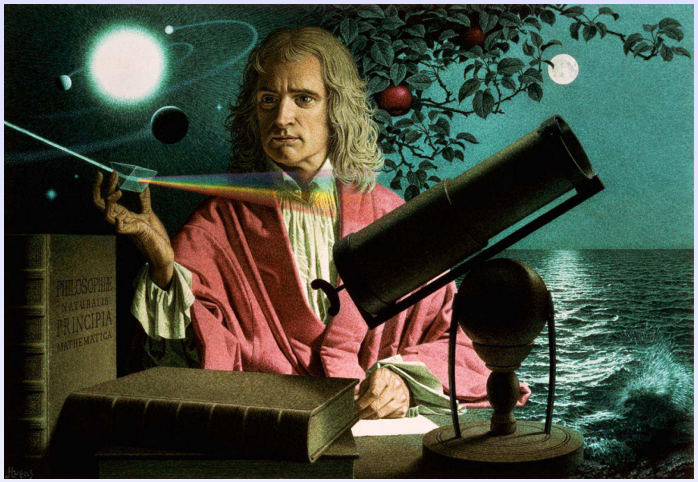
We can then consider a machine's ability to act as a recogniser for a language.



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, we 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.

About Theory and Practice

"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

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

Kurt Lewin

The Week 1 Lecture

I plan to say a few more words about the subject but mainly about practical issues of running it.

We will have time to tackle a few fun problems.

Most of the lecture time will be used to show you a bit of Haskell programming.

We will play around with some code, just to give you a feel for the language.

To Prepare

Lectures and tutes will be delivered via Zoom.
You will find the Zoom link to lectures on Canvas.

By Friday each week, the next week's teaching materials will be available. That will include videos to watch before the lecture, as well as tutorial exercises to work on before your tute.

Tutorials are very important in this subject. They start in Week 1!
To get value out of the tutes, turn up well prepared.

As soon as possible, become familiar with Haskell by going through the Grok modules. Find other useful Haskell resources at `https://www.haskell.org/documentation`.