COSC 341
Theory of Computing
Lecture 1
What is computing?

Stephen Cranefield
stephen.cranefield@otago.ac.nz

Today: paper details, history, finite state machines

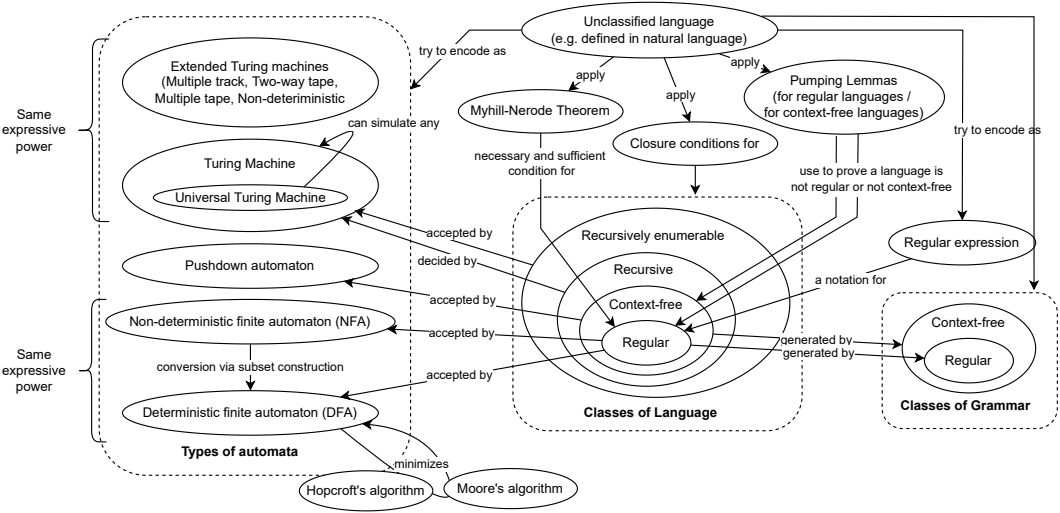
Paper details

- The main source of information for the paper is the website at https://cosc341.cspages.otago.ac.nz/. Blackboard will be used only for assignment submissions and grading.
- Assessment is in the form of two problem sets (10% each), a group report and presentation (10%) and the final exam (70%).
- Two lectures per week: Monday 11–11:50am and Tuesday 12–12:50pm.
- One tutorial per week: Friday, 3–4:50pm (with an alternative day and time on the week of Good Friday: Tuesday, 2–3:50pm)
- You are encouraged and expected to work together during the tutorials. However, work on the problem sets must be entirely your own. See the course outline on this paper's website for information about the university's academic integrity policies and procedures. If in doubt - ask first!

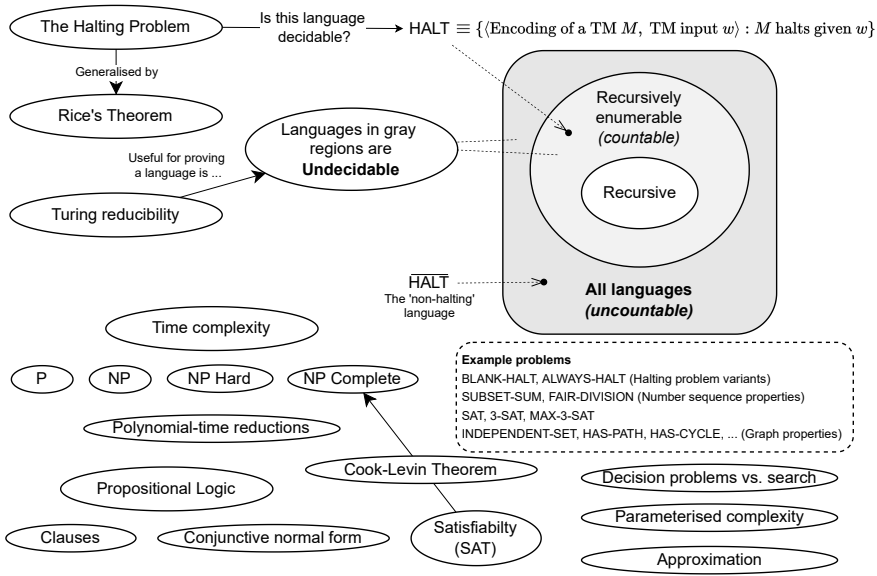
High level course overview

- Why do we need a theory of computing?
- A limited, but important, model of computing (finite state machines)
- A slightly less limited model of computing (pushdown automata)
- **The** model of computing (Turing machines)
- What can be computed? What can't?
- What can be computed efficiently? What can't? Do we know? (P, NP and NP-complete)
- What about approximation?
- The future?

COSC341 concepts, weeks 1 to 7



COSC341 concepts, weeks 8 to 12



Machines first

- In some previous iterations of this paper we spent at least two weeks (re-)introducing the mathematical concepts and language necessary to talk about the theory of computation.
- That wasn't very interesting!
- So we'll start from a notion of machine, think about what we want to be able to say about it, and introduce the maths as needed.
- If, at any point, the notation or concepts are unclear, **stop me**.

A backwards history of computing

- The resurgence of AI: transformers, large language models etc. (2024)
- The cloud, big data, deep networks, GPUs (2021)
- Hand-held devices (2000)
- The internet (1991)
- Supercomputers and personal computing (1980)
- Mainframes (1960)
- The first general purpose digital computers (1950)
- Electronic data processing (1930)
- Babbage's analytical engine (1830 - vapourware unfortunately)
- Punch cards (Hollerith 1880, Jaquard 1800)

Why a theory of computing?

To try to answer the big questions:

- What can some systems do that others can't?
- What (if anything) can't be computed at all?
- What (if anything) can't be computed efficiently?
  Motivation: See this cartoon

To understand the importance of abstraction, simulation, and reduction.

Alan Turing (1950)

In section 5 of the paper that introduces the *imitation game*, or Turing test, he writes:

> *As we have mentioned, digital computers fall within the class of discrete state machines. But the number of states of which such a machine is capable is usually enormously large. For instance, the number for the machine now working at Manchester it about* $2^{165,000}$ *. . .*
>
> *This special property of digital computers, that they can mimic any discrete state machine, is described by saying that they are universal machines.*

What mathematicians know in their bones

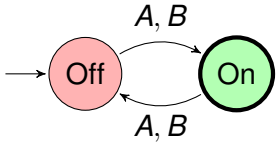
To understand how machines with $2^{165,000}$ or even $2^{200,000,000,000}$ possible states work, think about machines with two or three states (or possibly a few more) and then generalise.

Zero, one and two are qualitatively different concepts. All numbers greater than or equal to two are alike (O.K. sometimes three is the break point between few and many).

Two buttons, one light

- Our front lounge has a central light.
- Flipping either switch changes the state of the light.
- This is a computing device! It takes input (a switch flip) and produces output (the corresponding change in light state).
- How can we model this system?

Two buttons one light

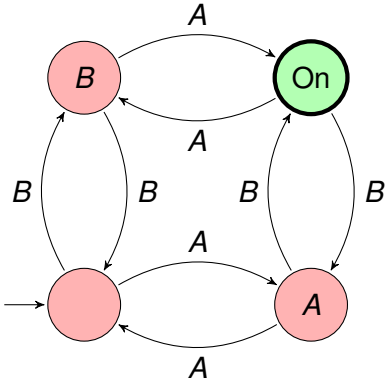


- The light starts out in the **Off** state.
- In either state, making an input of either A or B switches to the other state.
- We consider the **On** state to be *accepting* – any sequence of inputs that leads to this state is considered successful.
- The successful inputs are all strings consisting of characters from $\{A, B\}$ that have an odd number of characters.

Two different buttons, one light

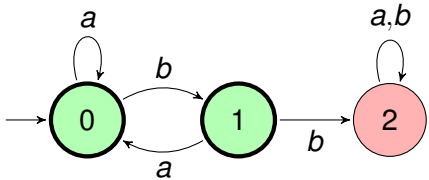
- What about the system where both switches need to be flipped an odd number of times to turn the light on?
- Now we'll need more than two states since basically we need to record the parity of both the number of A flips and the number of B flips.

Two different buttons, one light



- The light starts out in the lower left state.
- We consider the **On** state to be *accepting* – any sequence of inputs that leads to this state is considered successful.
- The successful inputs are all strings consisting of characters from $\{A, B\}$ that have an odd number of each letter.

What does this machine do?



- The machine starts in the state labeled 0.
- There are two buttons to press – *a* and *b*. These define the alphabet of the machine.
- Each button press causes a transition according to the labelled arrows.
- Which sequences of button presses leave us in an accepting state (coloured green above)? These sequences (strings) are the language accepted by the machine.

Try this

Design a machine whose accepted language is “zero or more copies of *a* followed by exactly one *b*.”

Our first definition!

A deterministic finite state automaton (Deterministic finite state automaton. (DFA)), **A**, consists of the following:

- A finite set Σ called its alphabet,
- A finite set $\mathcal{S}$ called its states,
- A function $T : \mathcal{S} \times \Sigma \to \mathcal{S}$ called its transition function,
- A single element $s \in \mathcal{S}$ called its start state,
- A subset $A \subseteq \mathcal{S}$ called its final states or accepting states.

We'll explore this definition in more detail next time!