COSC 341
Theory of Computing
Lecture 3
What is computation?

Stephen Cranefield
stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Are we really doing computing when accepting languages?

- In what sense, if any, does a language acceptor (e.g. a DFA) solve computational problems?
- All it does is recognise the language it accepts.
- At a stretch we could think of it as providing a method mapping `String` to `boolean`.
- Is that really enough to represent a general class of computational problems?

Example: Computing the square root of 2

- Suppose we have a device (not necessarily a DFA) that computes (only) the square root of 2. We turn it on and it displays $1.4142\ldots$.
- Can we view that as a language recognition problem?
- Suppose we accept all strings that are prefixes (or consecutive roundings) of $\sqrt{2}$:

$$1$$
$$1.4$$
$$1.41$$
$$1.414$$
$$\ldots$$

- Is this a good enough model of computation?

Example: Recognising any square root

- How can we generalise this so that our machine can recognise any square root? ($\sqrt{5}, \sqrt{17}, \ldots$)
- Do we still have a language recognition problem?
- Consider a calculator:
  - Enter the digits of the input number, e.g. 241
  - Press the "$\sqrt{}$" button
  - The digits of the answer are shown.
  - Suppose we view this input-output sequence as a language, e.g.:

$$2\,4\,1\,\sqrt{}\,1$$
$$2\,4\,1\,\sqrt{}\,1\,5$$
$$2\,4\,1\,\sqrt{}\,1\,5\,.$$
$$2\,4\,1\,\sqrt{}\,1\,5\,.\,5$$

- Is this a good enough model of computation?
- No, because we have to guess a possible answer for the machine to verify.

From verifier to computer

- Can we combine our square root verifier with another machine to produce a square root computer?
- Yes, we could run a “generate numbers” machine (leaving the bounds of language recognisers) to produce inputs to verify.
  - Loop until accepted:
    1. Generate an(other) output character
    2. Append it to the input
    3. Verify it
    4. If rejected, go to step 1 move on the next output character
    5. If accepted, accept the current sequence
- Clearly this is computable (but horribly inefficient).
- Claim: this means that a model of language acceptance is good enough for thinking about what we can and can’t compute.
- Note: we can generalise this beyond computing mathematical functions, e.g. the language of database description, query then answer.

What constrains a reasonable model of computation?

For today's purposes just one thing:

Each device (or method or . . .) that is allowed as a 'language recogniser' must be described by a finite string (i.e., program) over some finite alphabet Σ_d

The uncomputable exists

Theorem

*No reasonable model of computation allows **every** language to be recognised.*

Proof 1: a cardinality argument.

- There are a countable number of device descriptions, i.e. the finite sequences of characters over Σ_d can be put into a list $(d_1, d_2, d_3, \dots)$. One way is to list all 1-character programs in lexicographic order, then the 2-character ones, and so on. Note: many of these 'programs' may be syntactically incorrect.
- Let's assume our devices recognise languages over the same finite alphabet (Σ_L). As for Σ_d, we can list the finite strings over Σ_L in some order $\{(s_1, s_2, \dots)\}$. We can now represent each string by its index in this list.
- Each language L over Σ_L is a set of these strings, which we can represent as a infinite binary sequence, e.g. $b_L = (b_1, b_2, b_3, b_4, \dots)$, where $b_i = 1$ if the string with index i is in the language, and 0 otherwise.
- The set of sequences of this form can be viewed as fractions in base 2: $0.b_1b_2b_3\dots$, which are the real numbers between 0 and 1: an uncountable set.

The uncomputable exists

Theorem

*No reasonable model of computation allows **every** language to be recognised.*

Proof 1 continued

- Our countable list of devices can only recognise a countable subset of this uncountable set, therefore there must be an uncountable number of languages that our devices cannot recognise

Details: see https://legacy.earlham.edu/%7Epeters/writing/infapp.htm (especially Theorems 3, 16 and 17).

The uncomputable exists

Theorem

*No reasonable model of computation allows **every** language to be recognised.*

Proof 2: construction of a language L different from $L_1, L2, \ldots$

- As before, put our device descriptions into a sequence $(d_1, d_2, d_3, \ldots)$.
- List the languages each of these devices recognises, in the corresponding order: $(L_1, L2, L_3, \ldots)$. If d_i is syntactically incorrect, set $L_i = \{\}$
- Similarly, list each string over Σ_L in some order $(s_1, s_2, s_3, \ldots)$
- We consider whether each s_i is contained in each L_j (see the table on the next slide).
- We use the diagonal elements of the table to choose whether each $s_i \in L$. L is constructed to differ from each L_i: it will contain s_i if and only if L_i does not contain it.
- Therefore language L does not equal any L_i in the list and must not be recognised by any of our enumerated devices.

The uncomputable exists

Theorem

*No reasonable model of computation allows **every** language to be recognised.*

Proof 2: construction of a language L different from $L_1, L2, \ldots$

L_j / s_i	L_1	L_2	L_3	L_4	$\cdots$
s_1	$\in$	$\notin$	$\notin$	$\in$	$\cdots$
s_2	$\notin$	$\notin$	$\in$	$\notin$	$\cdots$
s_3	$\notin$	$\notin$	$\in$	$\in$	$\cdots$
s_4	$\in$	$\in$	$\in$	$\notin$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Construct $L = \{s_i : i \in \mathbb{N}, s_i \notin L_i\} = \{s_2, s_4, \ldots\}$