COSC 341
Theory of Computing
Lecture 11
Introducing Turing machines

Stephen Cranefield
stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Turing machines

Revision: The model matters

A pushdown automaton cannot accept the language:

$$\textsc{Powers-of-2} = \{a^n \,:\, n = 2^k \text{ for some } k \geqslant 0\}$$

But, we saw that replacing the stack with a queue *would* allow us to recognise this language. So pushdown automata are *not sufficiently powerful* to be a general model of computing.

But where does this end?

Turing machines

- Alan Turing introduced an abstract version of a mechanical computer, now known as the *Turing machine* (TM).
- As far as anyone knows, this model and every other “sufficiently powerful” model of mechanical computing solve exactly the same set of problems.
- Because, among other things, any “sufficiently powerful” mechanical computer should be able to simulate any other mechanical computer.
- The idea that there is a a pool of sufficiently and equally powerful computing models (including the Turing machine) is called the *Church-Turing thesis*.

What is a Turing machine?

- **A finite alphabet**: input symbols (lower case), punctuation (usually things like #), blank (␣), and markers (upper case letters or symbols other than letters).
- **Finite control** i.e., a finite number of possible internal states.
- A single infinite (in one direction) **tape** divided into **cells**.
- Each cell can hold a single character from the alphabet.
- A **read-write head**, which can determine the contents of the cell it is reading (and write onto that cell).
- Initially, the input is on the tape (usually preceded by a blank or some "beginning of input" character for convenience).

How does a Turing Machine work?

Start with the read-write head over the leftmost cell and execute a sequence of atomic steps, each of which is:

- Read the current symbol on the tape
- Based on it and the current state
  - Write a symbol on the tape
  - Move the read-write head one place left, right, or leave it in place.
  - Change the internal state
- Halt if no transition is defined for the current conditions (and accept or not according to the state we're in)
- Crashes (moving the read-write head past the left-hand end of the tape) and infinite computations are possible and are considered to be non-accepting.

What is the transition function?

- Inputs: Current symbol, current state
- Outputs: New state, new symbol, new direction to move
- Deterministic? Yes, for the moment, i.e., at most one transition defined for the current symbol and state.
- Remember, no transition means halt.

A formal definition (not the only one)

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ and Γ are all finite sets and

- Q is the set of states,
- Σ is the input alphabet,
- Γ is the tape alphabet, where the *blank* symbol $\sqcup$ is in Γ and $\Sigma \subseteq \Gamma$,
- $Q \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$ is the transition function (L means "left", S means "stay" and R means "right"),
- $q_0 \in Q$ is the start state,
- $q_{\text{accept}} \in Q$ is the accept state,

Some versions also require a single rejection state $q_{\text{reject}} \in Q, q_{\text{reject}} \neq q_{\text{accept}}$.

Modified from M. Sipser, Introduction to the theory of computation (to remove the single rejection state and to allow spaces in the input and the tape head to stay in the current state)

And now the examples

We will use Anthony Morphett's Turing Machine simulator.

- Accept if the input is $w\#w$ for some $w \in \{a, b\}^*$.
- And many more.