Formalising Algorithms

To prove a lower bound on the complexity of a problem, rather than a specific algorithm, we need to prove a statement about all algorithms for solving it.

In order to prove facts about all algorithms, we need a mathematically precise definition of algorithm.

We will use the *Turing machine*.

> The simplicity of the Turing machine means it's not useful for actually expressing algorithms, but very well suited for proofs about all algorithms.

Turing Machines

For our purposes, a Turing Machine consists of:

- K — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$.
- $s \in K$ — an initial state;
- $\delta : (K \times \Sigma) \to (K \cup \{a, r\}) \times \Sigma \times \{L, R, S\}$

  A transition function that specifies, for each state and symbol a next state (or accept acc or reject rej), a symbol to overwrite the current symbol, and a direction for the tape head to move (L – left, R – right, or S - stationary)

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition
A *configuration* is a triple (q, w, u), where $q \in K$ and $w, u \in \Sigma^\star$

The intuition is that (q, w, u) represents a machine in state q with the string wu on its tape, and the head pointing at the last symbol in w.

The configuration of a machine completely determines the future behaviour of the machine.

Computations

Given a machine $M = (K, \Sigma, s, \delta)$ we say that a configuration (q, w, u) *yields in one step* (q', w', u'), written

$$(q, w, u) \to_M (q', w', u')$$

if

- $w = va$;
- $\delta(q, a) = (q', b, D)$; and
- either $D = L$ and $w' = v$ $u' = bu$
  or $D = S$ and $w' = vb$ and $u' = u$
  or $D = R$ and $w' = vbc$ and $u' = x$, where $u = cx$. If u is empty, then $w' = vb\sqcup$ and u' is empty.

Computations

The relation $\rightarrow_M^\star$ is the reflexive and transitive closure of $\rightarrow_M$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \rightarrow_M c_{i+1}$, is called a *computation* of M.

The language $L(M) \subseteq \Sigma^\star$ *accepted* by the machine M is the set of strings

$$\{x \mid (s, \triangleright, x) \rightarrow_M^\star (\mathrm{acc}, w, u) \text{for some } w \text{ and } u\}$$

A machine M is said to *halt on input* x if for some w and u, either $(s, \triangleright, x) \rightarrow_M^\star (\mathrm{acc}, w, u)$ or $(s, \triangleright, x) \rightarrow_M^\star (\mathrm{rej}, w, u)$

Decidability

A language $L \subseteq \Sigma^\star$ is *recursively enumerable* if it is $L(M)$ for some M.

A language L is *decidable* if it is $L(M)$ for some machine M which *halts on every input*.

A language L is *semi-decidable* if it is recursively enumerable.

A function $f : \Sigma^\star \rightarrow \Sigma^\star$ is *computable*, if there is a machine M, such that for all x, $(s, \triangleright, x) \rightarrow_M^\star (\mathrm{acc}, f(x), \varepsilon)$

Example

Consider the machine with δ given by:

	$\triangleright$	0	1	$\sqcup$
s	$s, \triangleright, R$	$s, 0, R$	$s, 1, R$	$q, \sqcup, L$
q	$\mathrm{acc}, \triangleright, R$	$q, \sqcup, L$	$\mathrm{rej}, \sqcup, R$	$q, \sqcup, L$

This machine will accept any string that contains only 0s before the first blank (but only after replacing them all by blanks).

Multi-Tape Machines

The formalisation of Turing machines extends in a natural way to multi-tape machines. For instance a machine with k tapes is specified by:

- K, Σ, s; and
- $\delta : (K \times \Sigma^k) \rightarrow K \cup \{a, r\} \times (\Sigma \times \{L, R, S\})^k$

Similarly, a configuration is of the form:

$$(q, w_1, u_1, \ldots, w_k, u_k)$$

Complexity

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f(n))$ if there is a machine $M = (K, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f(n))$.

Similarly, we define $\mathsf{SPACE}(f(n))$ to be the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

In defining space complexity, we assume a machine M, which has a read-only input tape, and a separate work tape. We only count cells on the work tape towards the complexity.

Nondeterminism

If, in the definition of a Turing machine, we relax the condition on δ being a function and instead allow an arbitrary relation, we obtain a *nondeterministic Turing machine*.

$$\delta \subseteq (K \times \Sigma) \times (K \cup \{a, r\} \times \Sigma \times \{R, L, S\}).$$

The yields relation $\to_M$ is also no longer functional.

We still define the language accepted by M by:

$$\{x \mid (s, \triangleright, x) \to_M^\star (\mathrm{acc}, w, u) \text{ for some } w \text{ and } u\}$$

though, for some x, there may be computations leading to accepting as well as rejecting states.

Computation Trees

With a nondeterministic machine, each configuration gives rise to a tree of successive configurations.

$$\begin{tikzcd}
& (s,\triangleright,x) \arrow[dl] \arrow[d] \arrow[dr] & & \\
(q_0,u_0,w_0) \arrow[d] & (q_1,u_1,w_1) \arrow[d] \arrow[dr] & (q_2,u_2,w_2) \arrow[dr] & \\
(q_{00},u_{00},w_{00}) & (q_{10},u_{10},w_{10}) & (q_{11},u_{11},w_{11}) & (\mathrm{rej},u_2,w_2) \\
\vdots & & \vdots & \\
(\mathrm{acc},\ldots) & & &
\end{tikzcd}$$

Decidability and Complexity

For every decidable language L, there is a computable function f such that

$$L \in \mathsf{TIME}(f(n))$$

If L is a semi-decidable (but not decidable) language accepted by M, then there is no computable function f such that every accepting computation of M, on input of length n is of length at most $f(n)$.