COMP30026 Models of Computation

Turing Machines

Harald Søndergaard

Lecture 18

Semester 2, 2017

Some Deep Questions

What are the fundamental capabilities of a computer?

What is "computable"?

Are there limits, in principle, to what we can compute?

What is an algorithm?

How are algorithms and functions related?

Algorithmic Problems

Mathematicians have this proud history of inventing algorithms and posing algorithmic challenges.

Here, for example, is the famous tenth problem from a list of 23 posed by David Hilbert in 1900:

> *Find an algorithm that determines whether a polynomial has an integral root.*

Alan Turing

Alan Turing was born in 1912. At that time, a "computer" was a (usually female) human employed to do tedious numerical calculations.

Legacy: "Turing machine", the "Church-Turing thesis", "Turing reduction", the "Turing test", the "Turing award", and much more.

One of Turing's great accomplishments was to put "computability" on a firm foundation and to establish that certain important problems do not have an algorithmic solution.

Undecidable Problems

70 years after Hilbert posed his tenth problem, it was shown that there is no algorithm for it.

This fact, however, could only be shown once the world had a formal definition of what an algorithm is.

In the terminology that we will use later, we say that

$$\{p \mid p \text{ is a polynomial with integral root}\}$$

is a language (or problem) which is undecidable.

The Entscheidungsproblem

Around 1930, first-order logic had been found to have a sound and complete axiomatisation (by Kurt Gödel).

What was then considered the foremost outstanding problem of mathematical logic was the so-called Entscheidungsproblem: whether there is a decision procedure for first-order logic.

If the answer was yes (as was commonly assumed) then every theory that can be formalised in first-order logic would have an algorithm for deciding the truth of assertions in that theory.

Inspired by Gödel, Turing set out to prove that the answer was no.

For this, he needed a rigorous definition of "algorithm".

Turing Machines and the Universal Computer

Turing proceeded by defining a bare-bones "computer".

He did this by analysing what a person actually does, step by step, when following the rules of an algorithm.

The result has since been known as the Turing machine.

Importantly, Turing realised that a Turing machine can operate on strings that represent other Turing machines, and in particular one can simulate the operation of another.

In fact, a single, "universal" machine can be constructed which can simulate any given machine.

Turing Machines and Modern Computers

It is now clear that Turing's ideas had a profound impact on the design of "stored-program" computing technology in the 1940s and 1950s.

Turing spent most of the WWII years working in Bletchley Park, decoding German military communications.

This work involved building electronic computers, and after the war, Turing continued to be involved in practical engineering, working on the ACE project.

We Have Many Models of Computability

Turing machines (A. Turing, 1936)

Lambda calculus (A. Church, 1936)

Partial recursive functions (S. Kleene, 1936)

Post systems (E. Post, 1943)

Markov algorithms (A. Markov, 1954)

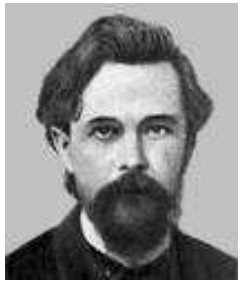
While programs

Register machines

Horn clauses

⋮



The Church-Turing Thesis

The class of computable functions is exactly the class of functions that can be realised by

⟨insert your favourite model here⟩

External evidence: All the above models are "equivalent" in spite of the fact that they all look very different, and were developed independently.

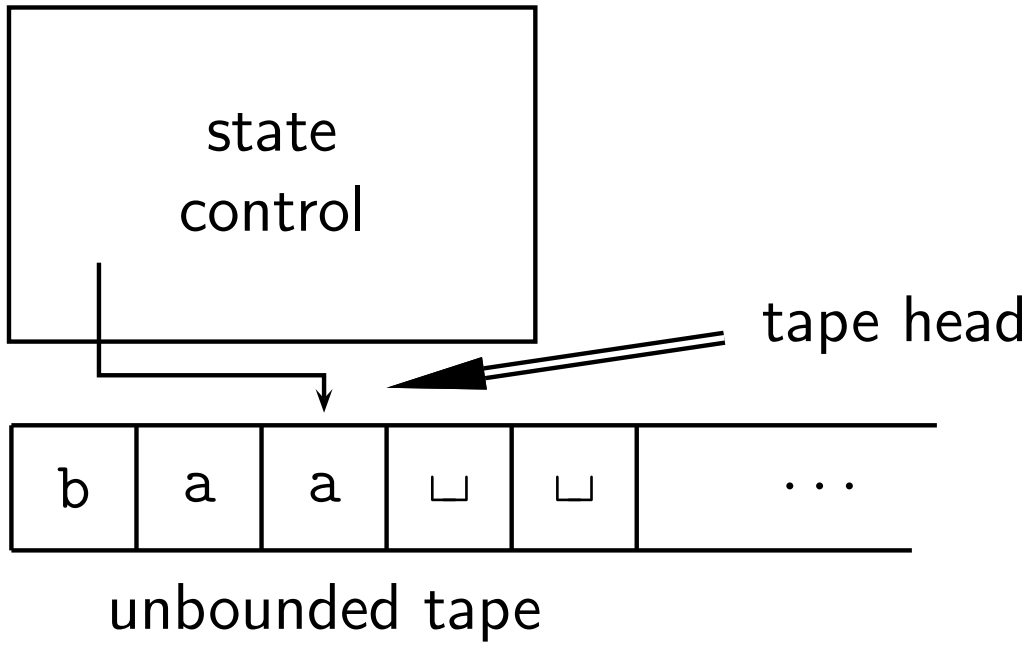
Internal evidence: It seems that no matter how we "extend" any of them, we fail to get something that is more powerful.

Turing Machines

A Turing machine has an unbounded tape through which it takes its input and performs its computations.

Unlike our previous automata it can:

- both read from and write to the tape, and
- move either left or right over the tape.



The machine has distinct accept and reject states, in which it accepts/rejects irrespective of where its tape head is.

Turing Machines Formally

A Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

- Q is a finite set of states,
- Γ is a finite tape alphabet, which includes the blank character, $\sqcup$,
- $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$ is the input alphabet,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- q_0 is the initial state,
- q_a is the accept state, and
- q_r $(\neq q_a)$ is the reject state.

The Transition Function

A transition $\delta(q_i, x) = (q_j, y, d)$ depends on two things

1. current state q_i, and
2. current symbol x under the tape head

It consists of three actions

1. change state to q_j,
2. over-write tape symbol x by y, and
3. move the tape head in direction d.

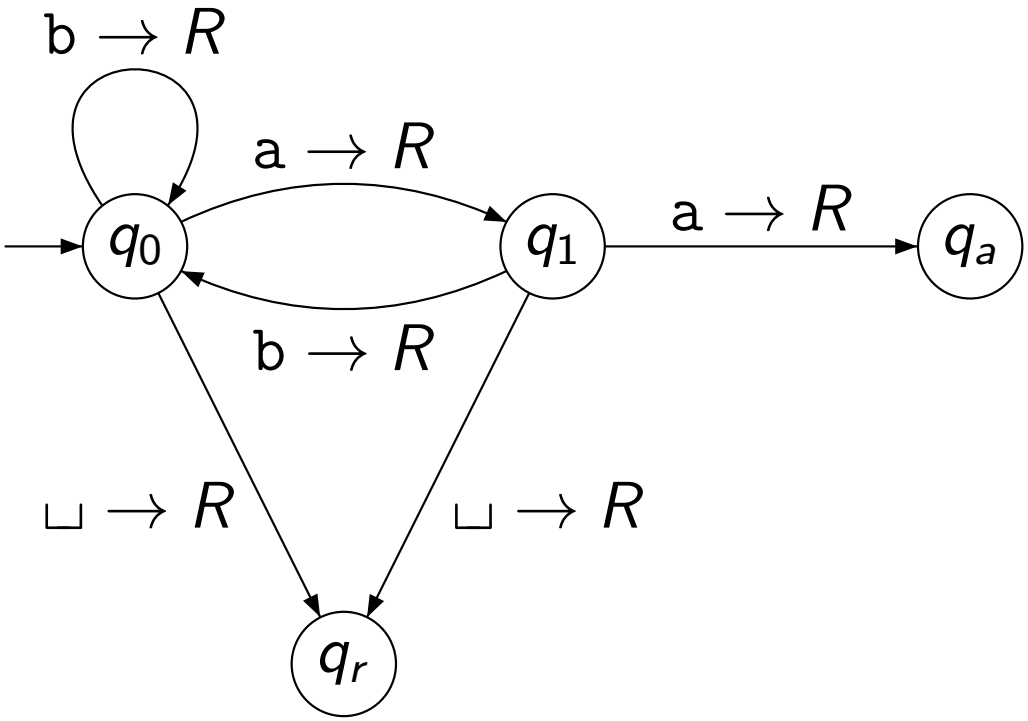
Drawing Turing Machines

We can have a graphical notation for Turing machines similar to that for finite automata.

On an arrow from q_i to q_j we write

- $x \to d$ when $\delta(q_i, x) = (q_j, x, d)$, and
- $x \to y, d$ when $\delta(q_i, x) = (q_j, y, d), y \neq x$.

Turing Machine Example 1

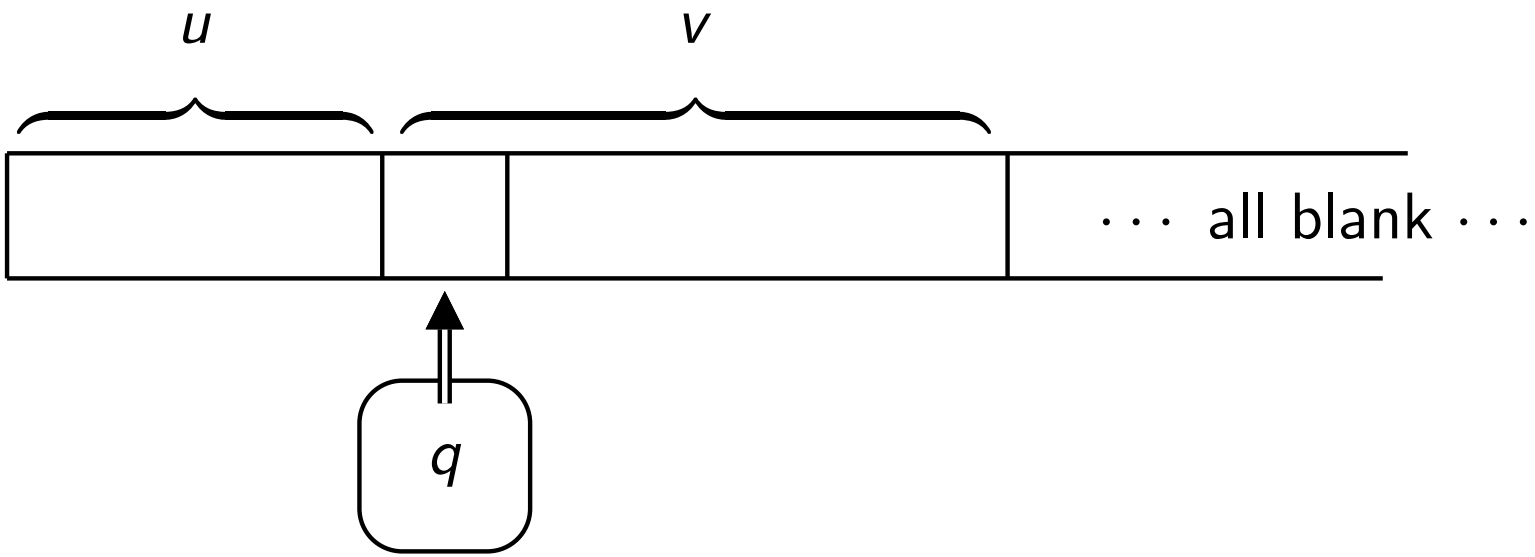


This machine recognises the regular language $(\mathtt{a} \cup \mathtt{b})^*\mathtt{aa}(\mathtt{a} \cup \mathtt{b})^*$.

We can leave the reject state q_r out with the understanding that transitions that are not specified go to q_r.

Turing Machine Configurations

We write uqv for this configuration:



On input aba, the example machine goes through 5 configurations:

$$
\begin{aligned}
& \epsilon q_0 \mathtt{aba} \qquad \text{(or just } q_0\mathtt{aba}\text{)} \\
\Rightarrow\ & \mathtt{a} q_1 \mathtt{ba} \\
\Rightarrow\ & \mathtt{ab} q_0 \mathtt{a} \\
\Rightarrow\ & \mathtt{aba} q_1 \sqcup \\
\Rightarrow\ & \mathtt{aba} \sqcup q_r
\end{aligned}
$$

Computations Formally

For all $q_i, q_j \in Q$, $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$, we have

$$\begin{array}{ll} uq_ibv \Rightarrow ucq_jv & \text{if } \delta(q_i, b) = (q_j, c, R) \\ q_ibv \Rightarrow q_jcv & \text{if } \delta(q_i, b) = (q_j, c, L) \\ uaq_ibv \Rightarrow uq_jacv & \text{if } \delta(q_i, b) = (q_j, c, L) \end{array}$$

The start configuration of M on input w is q_0w.

M accepts w iff there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that

1. C_1 is the start configuration q_0w,
2. $C_i \Rightarrow C_{i+1}$ for all $i \in \{1 \ldots k-1\}$, and
3. The state of C_k is q_a.

Turing Machines and Languages

The set of strings accepted by M is the language of M, $L(M)$.

A language A is Turing recognisable (or recursively enumerable, or just r. e.) iff $A = L(M)$ for some Turing machine M.

Three behaviours are possible for M on input w: M may accept w, reject w, or fail to halt.

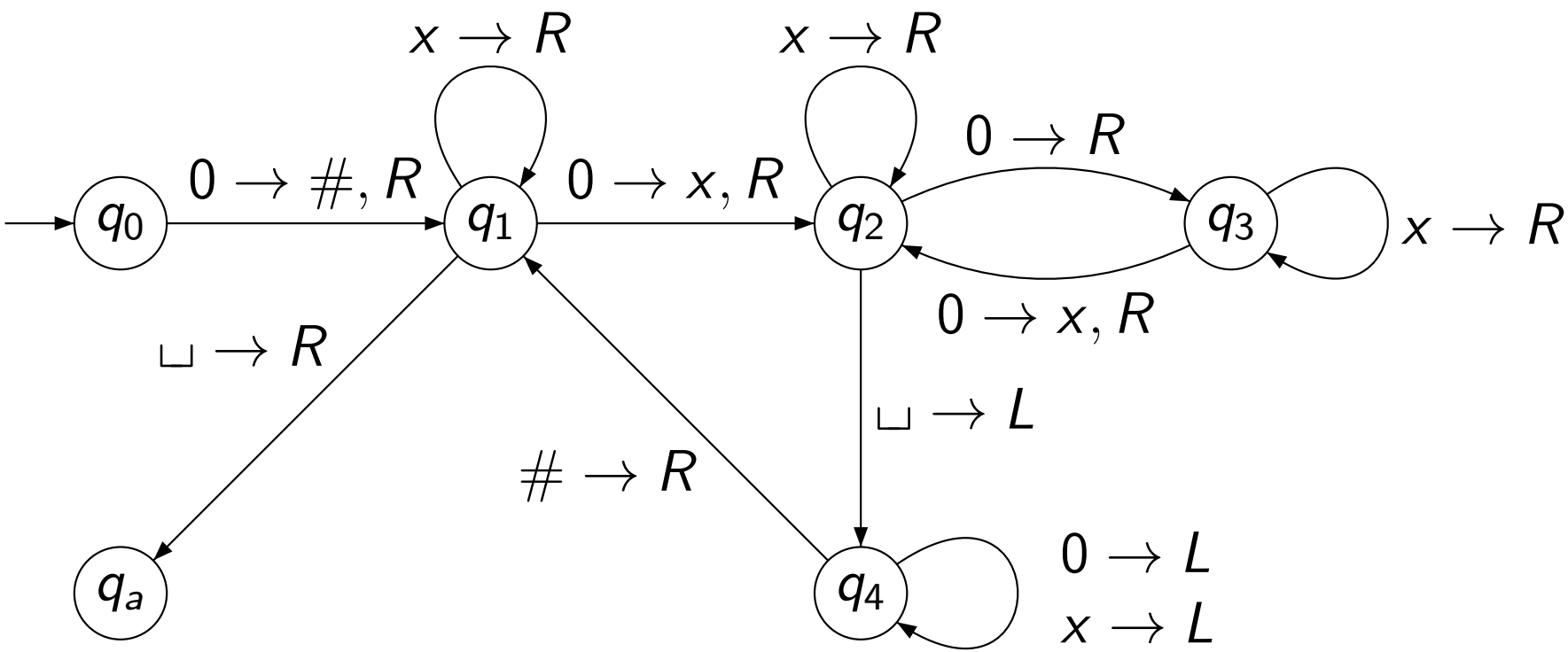
If A is recognised by a Turing machine M that halts on all input, we say that M decides A.

A language is Turing decidable (or recursive, or just decidable) iff some Turing machine decides it.

Turing Machine Example 2

This machine decides the language $\{0^{2^n} \mid n \geq 0\}$.

It has input alphabet $\{0\}$ and tape alphabet $\{\sqcup, 0, x, \#\}$.



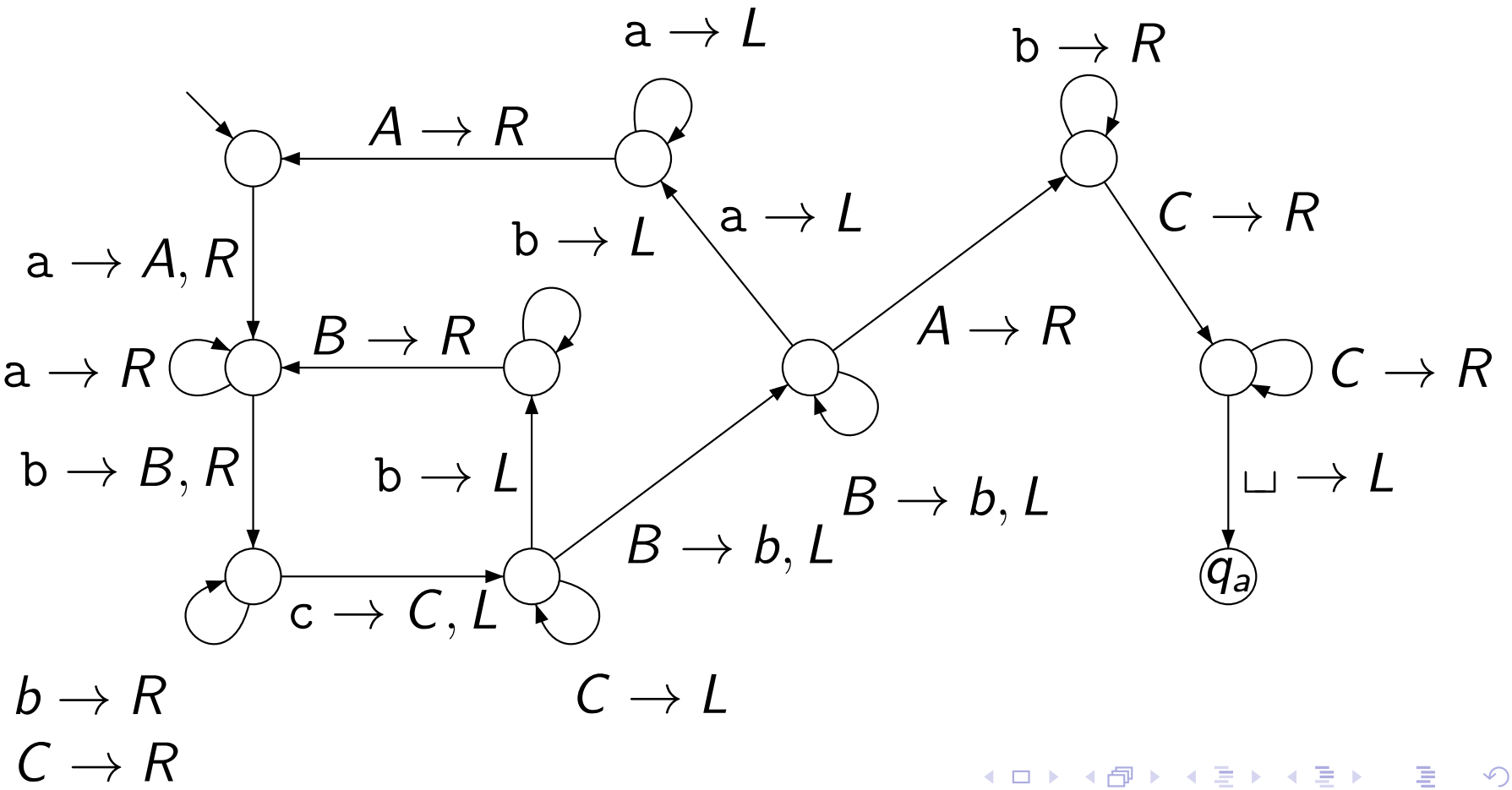
Running the machine on input 000:

$$q_0000 \Rightarrow \#q_100 \Rightarrow \#xq_20 \Rightarrow \#x0q_3\sqcup \Rightarrow \#x0\sqcup q_r$$

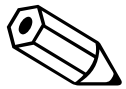
Turing Machine Example 3

This machine decides the language $\{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid k = i \cdot j, i, j > 0\}$.

Its tape alphabet is $\{\sqcup, \mathrm{a}, \mathrm{b}, \mathrm{c}, A, B, C\}$.



Exercise



Design a Turing machine with input alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ which decides the language

$$A = \{\mathtt{a}^i \mathtt{b}^j \mathtt{c}^k \mid i, j, k > 0 \wedge i + j \geq k\}.$$

The Versatility of Turing Machines

We can decide that a Turing machine produces output (not just accept/reject) through its tape. This way a Turing machine can be a general computing device, not just a language recogniser.

We can capture data other than strings via suitable representations.

For example, natural numbers can be represented as strings of (unary, binary, decimal, ...) digits, so a Turing machine can compute number theoretic functions $\mathbb{N} \to \mathbb{N}$.

Or, by suitable encoding, it can take multiple arguments, and/or return multiple results.

A Turing machine can also solve graph problems, once we decide on a suitable representation for graphs.

Robustness of Turing Machines

Different books use different definitions of Turing machines.

Most differences are minute and technical and aim at making the machines easier to program (for example, we may insist that machines start with a tape that has the first cell blank, and they try to leave that cell blank—to make it easier to compose machines).

Similarly, in addition to the two kinds of tape movement, we can allow a 'no move' option.

Turing machines are robust in the sense that such changes to the machinery do not effect what the machines are capable of computing.

Variants of Turing Machines

In an attempt to make the Turing machine more powerful we could add more radically to its features:

- Let its tape extend indefinitely in both directions.
- Let its tape have multiple tracks.
- Let there be several tapes, each with its independent tape head.
- Add nondeterminism.

However, none of this increases a Turing machine's capabilities as a language recogniser.

Coming to a Theatre Near You

Decidable and undecidable languages.