Finite Automata to Turing Machines

- Decidability
- Models of Computation
- The Halting Problem

Computability

❑ The Decidability Problem (Entscheidungsprobleim) was posed by David Hilbert at the turn of the century

Can there exist, at least in principle, a definite method by which all mathematical problems could be decided?

❑ Turing defined "definite method"

- invented the concept of algorithm
- described a general abstract machine that represented computation: The Turing Machine
- showed there were questions it could not answer:
  The Halting Problem

Invention versus Discovery

❑ A key question in the philosophy of mathematics

❑ Invention
- All mathematical results are invented by humans, there is no one mathematical truth, only lots of different logical systems that humans invent

❑ Discovery
- All mathematical truths exist in the Platonic Universe and are discovered, because there is a single mathematical truth

A Grand Unified Theory of Everything (e.g. Hawking) can only exist if there is a single mathematical truth in the universe

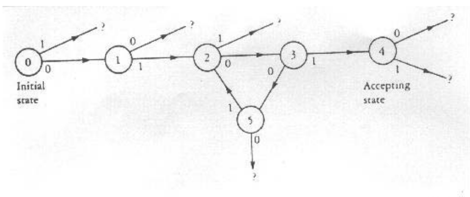
Models of Computation

❑ Key to Turing's approach to the Entscheidungsprobleim and the theoretical underpinning of Computer Science

❑ **Finite Automata** are the simplest models

- black box accepting two inputs; 0 and 1
- single light bulb provides output
- reset button
- machine operation is described by a set of internal states, including two special ones
  - initial state = no input yet
  - accepting state = lights the bulb

Finite Automata



❑ In general a finite automaton is a finite collection of states + a finite **alphabet** of input signals + a function which determines the new state from the current state and input

Finite Automata

❑ Input sequences which put the machine in its accepting state are called **words**

❑ The set of words accepted by a finite automaton is called the **language** of the automaton

❑ The language of the automaton shown here is described by the **regular expression**

01 (001)* 01

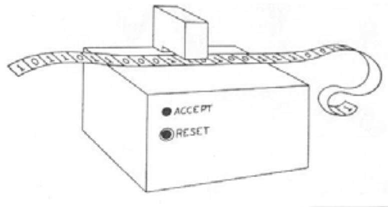
words begin and end 01, with any number of 001s in between

Recognising a word is equivalent to performing a computation, more powerful models of computation can recognise larger languages

The Chomsky Hierarchy

- ❑ Four models in order of increasing generality
  - • Finite Automata
  - • Pushdown Automata
  - • Linear Bounded Automata
  - • Turing Machines
- ❑ Any machine built to one of these models is a computer
- ❑ All computers have a finite number of states, each cycle of operation involves a transition from one to another, triggered by a specific symbol from the computer's alphabet
- ❑ Model input sequence as a tape containing one symbol per cell

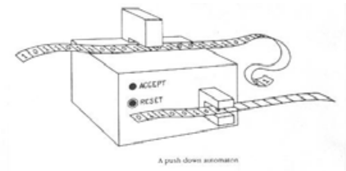
Finite Automata



❑ Tape is simply read from one end to the other, bulb either lights up or not, hit reset to restart. Symbols can be written to tape, but not read back in again.

Pushdown Automata

- ❑ FA plus auxiliary tape which can move in either direction
  - • read both tapes
  - • either moves auxiliary tape forward and writes to it or erases current symbol and moves auxiliary tape backwards
  - • not allowed to remove the first symbol from tape (a **<u>Stack</u>**)



Finite automata are a subclass of pushdown automata

Linear Bounded Automata

❑ Only have the main tape, but can move it in both directions

- • can re-read symbols they've written

BUT tape is limited by a linear factor *k*

❑ If a word of *n* symbols is input the machine can only use *kn* tape cells to determine if it is acceptable

❑ Pushdown automata can interpret, e.g. programming language syntax including nested brackets

❑ Linear bounded automata are effectively general (Turing) machines but with limited memory

Turing Machines

❑ One infinitely long read/writeable tape, moveable in both directions - has a start but no end

❑ In abstract terms a Turing Machine is

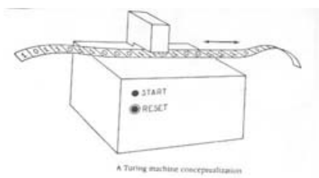
- • a finite collection of states
- • a finite alphabet of input signals
- • a function which determines the new state from the current state and input

but there is no accepting state - output is written to the tape

❑ Special states are the initial state as before and the **<u>halt state</u>**, reached when the computation has finished

Turing Machines

❑ More complex digital models have been proposed but turn out to be no more powerful than the Turing Machine



❑ Any model which formalises the concept of a computational procedure can be shown to be equivalent to the Turing Machine

- • only computations that can be executed by TMs can be called algorithms: **<u>The Church-Turing Thesis</u>**

The Universal Turing Machine

- The operations of the Turing Machine are the processes that constitute computation
- Any particular Turing Machine represents one algorithm
- The Universal Turing Machine
  - a Turing Machine which implements the Turing Machine algorithm
  - is supplied with input symbols and a set of instructions (on tape) specifying what a specific Turing Machine would do

<u>Since any algorithm can be implemented as a specific Turing Machine the UTM can perform any algorithm</u>

Back to Decidability

- Turing used the Universal Turing Machine idea to prove that in any general algorithm there were propositions which could not be decided
- Basic method was to assume Decidability was true and look for a paradox, e.g. The Liar's Paradox

"This statement is false."

  - If the statement is true, its content makes it false
  - If it is false, its content makes it true

Turing's paradox arose from **<u>The Halting Problem</u>**

The Halting Problem

- A Turing Machine is just a formal model of what is an algorithm
- A Universal Turing Machine is an algorithm for executing algorithms
- Any Turing Machine (including the UTM) is said to halt if the algorithm termintes
  - they only stop when the computation is finished

Suppose we define a Universal Turing Machine which halts if the Turing Machine it is executing does not halt…….

The Halting Problem

- What happens when we ask the UTM to execute itself, its own algorithm?
  - If the UTM halts, then the algorithm does not halt, i.e. the UTM does not halt.
  - If the UTM does not halt, then the algorithm does halt, i.e. the UTM does halt.

Contradiction!

- So using Turing's definition of a "definite method" there exists a problem which cannot be solved - **<u>the Universal Turing Machine cannot solve the Halting Problem</u>**

The Contribution?

- Alan Turing
  - Solved a fundamental problem in mathematics
  - Defined what it is to be an algorithm and explored the limits of computability
  - Provided an existence proof for the general purpose digital computer

………..and he didn't stop there

Things to Do & Think About

- The Halting Problem and its role in answering Hilbert's question is a key event in the history of computing. Make sure you understand the argument. There are pointers to alternative descriptions are on the module website.
- Turing was a well-known figure in certain parts of the UK computing community, but contributions were made by people with a variety of backgrounds in both Europe and the US. Can you find any **<u>evidence</u>** that the Turing machine influenced
  - the practical design of any computers?
  - any US developments?