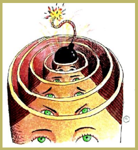
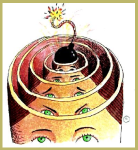


Lecture 37: Turing Machines

Aims:

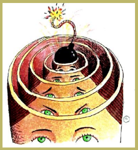
- To describe Turing machines, which are an important formal model of computation.



37.1. Introduction

- So far, we've presented algorithms and programs using DECAFF, MOCCA and Java. And we've tried not to make any assumptions about the computer on which these algorithms and programs will be executed: we've just assumed we'd use a 'standard' computer.
- In the next few lectures, we will be looking at devices of the simplest imaginable kind. They're not real, practical computers. They are a mathematical idealisation ('a formal model'). They're primitive in contrast to today's languages and computers. Nevertheless, they are powerful enough to execute any algorithm.
- The formal model of computation that we will study is called a *Turing machine* in honour of *Alan Turing* who invented it in 1936.

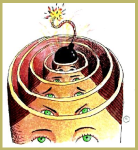
- Why do we want a formal model of computation?
  - We've been making some pretty bold claims about computability. And you might be wondering whether these claims only apply if you use certain languages or certain types of computers. It would be nice to show that these claims apply to all reasonable languages and computers. Turing machines offer a precise



notion of computation, enabling us to show that certain problems truly are non-computable.

– We wish to compare different programming languages and different types of computers. We want to know whether one is more powerful than another. (By power, we mean: can one solve more problems than another?) Turing machines offer a precise yardstick, against which we can compare other ways of doing computations.

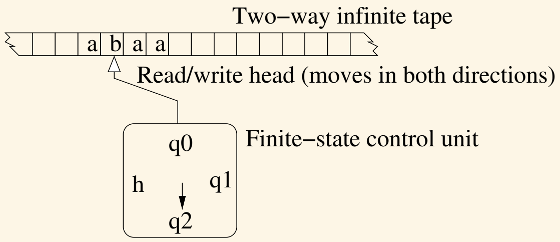
– We've been making some pretty bold assumptions when considering algorithm complexity and problem complexity. E.g. we've ignored the fact that arithmetic operations take longer when applied to numbers that exceed the size of a single word of memory. Turing machines offer a clearly-defined notion of a single indivisible computational step, enabling us to study algorithm complexity and problem complexity without unnecessary assumptions.



37.2. Turing Machines

37.2.1. Definition

- A *Turing machine* consists of a finite-state control unit and a tape that is infinite in both directions and divided into cells, each of which can hold one symbol:

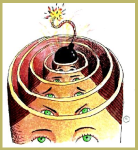

Two–way infinite tape
a
b
a
a
Read/write head (moves in both directions)
Finite–state control unit
q0
h
q1
q2

- At each step, the control unit performs two actions in a way dependent on:
  - the current state, and
  - the tape symbol currently scanned by the read/write head.

  The two actions will be:

  1. either
     - write a symbol into the cell that is under the read/write head (overwriting the one already there)

     or
     - move the read/write head one cell to the left or right;
  2. put the control unit into a new state.



- Only a certain finite set of symbols can be written on the tape. Call this the *tape alphabet*, Σ:
  - Σ always includes $\textvisiblespace$ ('blank'), and
  - Σ always excludes L and R.
- There will be a finite set of *states*, $Q = \{q_0, q_1, \ldots, q_n, h\}$:
  - q_0 will always be the *start state*, and
  - h will be the *halt state* (when the computation enters state h, the computation halts).
- We will use L and R to indicate that the read/write head moves one cell left/right.
- The actions are specified by a transition function, δ. For the mathematically minded, δ is a function

$$\text{from } Q \setminus \{h\} \text{ to } (\Sigma \cup \{L, R\}) \times Q$$

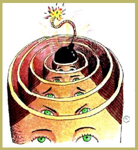
37.2.2. Simple Examples

- **Example 1** Consider a machine where $\Sigma = \{a, b, \textvisiblespace\}$ and $Q = \{q_0, h\}$ and δ is as follows:

δ	a	b	$\textvisiblespace$
q_0	L, q_0	L, q_0	$\textvisiblespace, h$

Class Exercise: What does this machine do?

- **Example 2** Consider a machine where $\Sigma = \{a, \textvisiblespace\}$ and $Q = \{q_0, q_1, h\}$ and δ is as follows:



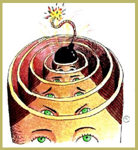
δ	a	$\sqcup$
q_0	$\sqcup, q_1$	$\sqcup, h$
q_1	—	L, q_0

(In this table, '—' is used to mean that this can't arise. I.e. this particular machine can never find itself in a situation where the control unit is in state q_1 and the cell of the tape under the read/write head contains an a.

Class Exercise: Why is this case impossible?

Mathematicians would not be happy with the idea of leaving this undefined. δ is supposed to be a function, which should be defined for all possible inputs. Mathematicians would just put some arbitrary stuff into this cell of the table, e.g. a, q_0, safe in the knowledge that it will never be used. I think it's clearer for us if we put '—' in.)

Class Exercise: What does this machine do?

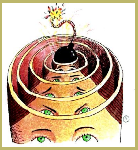


Module Home Page
Title Page
Page 7 of 9
Back
Full Screen
Close
Quit

37.3. Configurations and Traces

- The non-blank symbols that we place on the tape prior to the computation can be thought of as the Turing machine's input.
- We'll assume, *unless otherwise stated*, that the Turing machine always starts in state q_0 and the read/write head will be positioned over the *rightmost non-blank*.
- Because it can write onto the tape, a Turing machine can output an answer onto the tape. In general, we won't be overly concerned by whether the answer is written to a blank part of the tape or whether it overwrites some or all of the input.
- The *configuration* that a machine is in is given by:
  - the symbols on the tape,
  - the position of the read/write head, and
  - the current state.
- In fact, we'll split the symbols on the tape into three strings:
  - the symbols to the left of the read/write head up to and including the blank to the left of the leftmost non-blank;
  - the symbol currently being scanned by the read/write head, and
  - the symbols to the right of the read/write head up to and including the blank to the right of the rightmost non-blank.
- Here are some example configurations:

$$\begin{array}{cc}
\langle \textvisiblespace abb, b, bba\textvisiblespace, q_0\rangle & \langle \textvisiblespace ab, b, bbba\textvisiblespace, q_0\rangle \\
\langle \textvisiblespace a, b, bbbba\textvisiblespace, q_0\rangle & \langle \textvisiblespace, a, bbbbba\textvisiblespace, q_0\rangle \\
\langle \textvisiblespace, \textvisiblespace, abbbbba\textvisiblespace, q_1\rangle & \langle \textvisiblespace, \textvisiblespace, \textvisiblespace abbbbba\textvisiblespace, h\rangle
\end{array}$$



- With this notation, we can *trace* the operation of a Turing machine.
- We simply show a sequence of configurations, e.g.

$$\langle \sqcup abb, b, bba\sqcup, q_0\rangle \leadsto \langle \sqcup ab, b, bbba\sqcup, q_0\rangle \leadsto \langle \sqcup a, b, bbbba\sqcup, q_0\rangle$$

- E.g. We'll trace Example 2 using the following initial configuration:

$$\langle aaaa, a, \sqcup, q_0\rangle$$

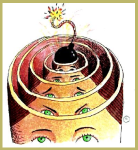
δ	a	$\sqcup$
q_0	$\sqcup, q_1$	$\sqcup, h$
q_1	—	L, q_0

$$\begin{array}{lcrcccc}
 & \langle & \sqcup aaa, & a, & \sqcup, & q_0 & \rangle \\
\leadsto & \langle & \sqcup aaa, & \sqcup, & \sqcup, & q_1 & \rangle \\
\leadsto & \langle & \sqcup aa, & a, & \sqcup, & q_0 & \rangle \\
\leadsto & \langle & \sqcup aa, & \sqcup, & \sqcup, & q_1 & \rangle \\
\leadsto & \langle & \sqcup a, & a, & \sqcup, & q_0 & \rangle \\
\leadsto & \langle & \sqcup a, & \sqcup, & \sqcup, & q_1 & \rangle \\
\leadsto & \langle & \sqcup, & a, & \sqcup, & q_0 & \rangle \\
\leadsto & \langle & \sqcup, & \sqcup, & \sqcup, & q_1 & \rangle \\
\leadsto & \langle & \sqcup, & \sqcup, & \sqcup, & q_0 & \rangle \\
\leadsto & \langle & \sqcup, & \sqcup, & \sqcup, & h & \rangle
\end{array}$$

Acknowledgements

In preparing this material, I have used [Har92], [Jun] and [LP81].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.



References

[Har92] D. Harel. *Algorithmics: The Spirit of Computing*. Addison-Wesley, 2nd edition, 1992.

[Jun] A. Jung. Models of Computation (Course Notes).

[LP81] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, 1981.