Turing Machines

Mathematical Machines

- "Mathematical" (as opposed to mechanical) machines
  - Turing Machines (potentially infinite-state)
  - Finite-state machines
  - Other categories (cf. CS 142, Theory of Computation)

What is a Turing Machine?

- A computational model thought to be *universal* from the viewpoint of **functions** that can be computed
- Proposed by Alan M. Turing as a means of discussing such functions
- Universality generally accepted by computer scientists based on Turing's argument (see text) and other evidence

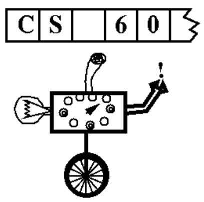
Alan M. Turing (1912-1954)

Turing Milestones

- 1936: Essay on computability
- 1940: Machine ("the Bombe") for decrypting the German *Enigma* code machine
- 1943: Participated in design and construction of an electronic computer (the "Colossus")
- 1949: First paper on proving correctness of programs
- 1950: Paper on AI ("the Turing test")
- 1951: Biological pattern formation ("morphogenesis")

The Enigma (from NSA museum)

Another Enigma?



Play about Turing

- "Breaking the Code" by Hugh Whitemore
- Played in London, New York, LA
- Public TV version

Turing Web Pages

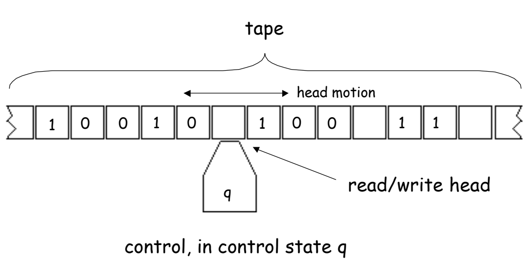
- Home Page (by Andrew Hodges)
- Home Page (Sheffield University)
- Clark University
- From History of Math
- History of Mathematics Archive
- Internet Scrapbook
- University of Manchester

Turing Machine Details

- The *tape*: an unbounded amount of memory. Consists of *cells*, each containing exactly one of a pre-convened set of characters (such as '0', '1', ' ' blank)

- The control: a finite amount of memory, the *control states*. Defines *control functions*.

Turing Machine



More about the Tape

- Only a finite portion of the tape is "non-blank" at any time.

- New cells are added at either end "as needed".

The Complete State of a TM

- The complete state of a TM is determined by:
  - The control state
  - The symbol currently under the head
  - The sequence of symbols to the right of the head
  - The sequence of symbols to the left of the head

Control Functions

- The control determines the following, given any combination of control state (q) and symbol under the head (s):
  - A new control state (q')
  - A new symbol to be written (s')
  - A head motion m (Left, Right, or None)
- Call the control partial-function f, so that
  f(q, s) = (q', s', m)

Sequential Operation

- The machine begins in a specified starting control state, with initial tape contents, and the head positioned at a standard place with respect to the contents.
- The machine goes through a sequence of states until it arrives at a halting state.

Halting Convention

- If f(q, s) is unspecified, then the TM is said to have ***halted*** in the current state.

5-tuple notation

- The control partial-function f, so that
  f(q, s) = (q', s', m)
  is often written as a set of 5-tuples of the form:
  (q, s, q', s', m)

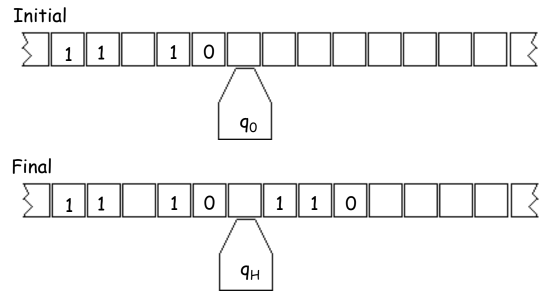
Various TM Categories

- **Transducer**: Starting with the initial tape contents, produce a new tape contents
- ***Acceptor or Classifier***: Starting with the initial tape contents, halt in either an accepting state or a rejecting state
- ***Generator***: Starting with an empty tape, generate the elements of some sequence on the tape.

Examples of TM Categories

- **Transducer**: Multiply two binary numerals
- **Acceptor or Classifier**: Determine whether or not a binary numeral is prime
- **Generator**: Starting with an empty tape, generate numerals for the primes, each separated by a blank

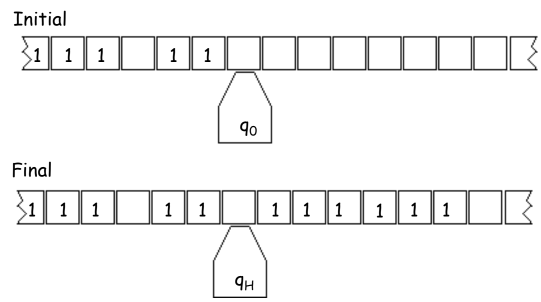
TM Multiplying Example



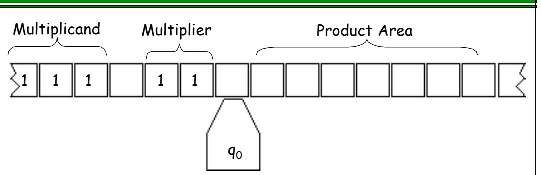
Simplifying TM Programming

- Allow symbols to be erased
- Use extra symbols: can always convert to fewer symbols later (by encoding the larger set)
- Use symbols with/without markers: these in effect are just a larger symbol set
- Use special encodings, such as 1-adic encoding (number n is n 1's or n+1 1's)

1-adic Multiplying



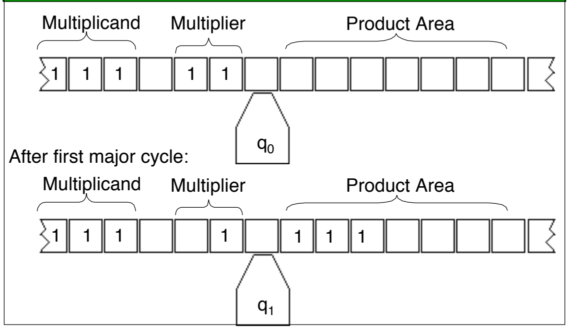
1-adic Multiplier Structure



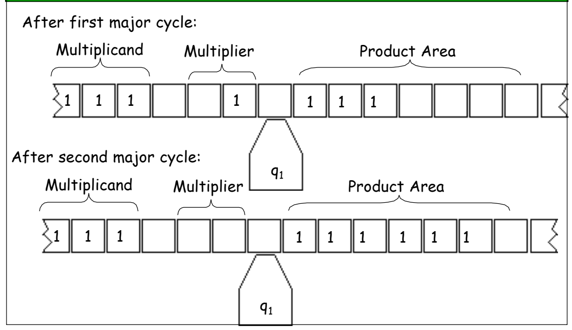
1-adic Multiplier Scheme

- Check whether the multiplicand is 0 (no 1's); if so, return to home position and halt.
- For each 1 in the multiplier:
  - Copy the multiplicand to the right of the accumulated product
  - Erase the leftmost 1 in the multiplier
- until all multiplier 1's have been erased
- Then restore the multiplier 1's and halt.

1-adic Multiplier In Operation



1-adic Multiplier In Operation



How to tell when done?

- Each time the multiplicand is copied, the leftmost 1 of the multiplier will be set to blank (it will be restored at the end).
- Moving left from q_1, if there is a 1 then the multiplier has not been decimated.

How to copy the multiplicand?

- This is tricky, because the multiplicand can be arbitrarily long; we cannot "count" arbitrarily-high in the control of the machine alone.
- During copying, make each 1 of the multiplicand into a 0. At the end of copying, turn all 0's back to 1's.
- The machine is done copying when there are no 1's left.

Completing the Machine (Sketch)

- We present this as an informal set of triples: (start state, what happens, end state)
  The "what happens" parts might require additional states to implement.
- (q_0, skip left over multiplier, q_1)
- (q_1, check multiplicand for 0, 0 ? q_2: q_3)
- (q_2, skip right over multiplier, q_H) major jump
- (q_3, skip right over multiplicand, q_4)
- (q_4, skip right over multiplier, q_5)

Completing the Machine (Sketch, part 2)

- (q_5, if blank to left, q_{18}) major jump
- (q_5, if non-blank to left, skip left over 1's, q_6)
- (q_6, move right, q_7)
- (q_7, move left to 1, q_8)
- (q_8, set (multiplicand bit) to 0, q_9)
- (q_9, move right to 1, q_{10})
- (q_{10}, set (multipliplier bit) to blank, q_{11})

Completing the Machine (Sketch, part 3)

- (q_{11}, move right to blank, q_{12})
- (q_{12}, move right to blank, q_{13})
- (q_{13}, write 1 (product), q_{14})
- (q_{14}, move left to blank, q_{15})
- (q_{15}, move left to 0 (multiplicand), q_{16})
- (q_{16}, convert 0's to 1's until blank, q_{17})
- (q_{17}, move right to blank, q_3) ← major jump
- (q_{18}, restore blanks in multiplier to 1, q_H)

Turing's Hypothesis

- Turing's Hypothesis is that for every computable function there is some Turing machine that computes it.
- Turing's argument was based on a direct appeal to intuition.

Turing's Hypothesis (2)

- In order to give a sound proof of the hypothesis, it would be necessary to characterize what it means to be computable.
- This would entail presenting another convincing notion of computability, which would have to be similarly argued.
- Most computer scientists and mathematicians accept Turing's notion as the notion.

Turing's Hypothesis (3)

- Other natural notions of computability have been proposed, such as systems of recursive functions.
- All such notions have been proved equivalent to Turing machines through appropriate encodings.

Non-Computable Functions

- There are more partial functions, say, from the natural numbers to themselves, than there are Turing machines:
  - The infinite set of functions N -> N is *not* countable (can't be enumerated).
  - The infinite set of Turing machines *is* countable (can be enumerated).

Divergence

- A Turing machine is said to **diverge** on an input if it never halts.
- Divergence is like a program that never terminates, e.g either due to an infinite loop or a search of an infinite space that can never yield an answer.
- If a machine diverges, the partial function it computes is undefined for this unput.

A Specific Non-Computable Function

- Let T_0, T_1, T_2, … be an enumeration of all Turing machines (over some specific alphabet, such as {0, 1, blank}).
- Similarly let x_0, x_1, x_2, … be an enumeration of all tapes for this alphabet.
- Define the partial function H as follows:

$$H(i) = \begin{cases} 1 \text{ if } T_i \text{ diverges on input } x_i \\ \text{undefined (i.e. intentionally diverges) if } T_i \text{ halts on } x_i \end{cases}$$

A Specific Non-Computable Function (2)

$$H(i) = \begin{cases} 1 \text{ if } T_i \text{ diverges on input } x_i \\ \text{diverges if } T_i \text{ halts on } x_i \end{cases}$$

- Claim: H is not computable by any Turing machine (in the sense that H(i) is the result of the machine on x_i).
- Suppose instead that H is computable by T_k for appropriate k. Then

$$H(k) = \begin{cases} 1 \text{ if } T_k \text{ diverges on input } x_k \\ \text{diverges if } T_k \text{ halts on } x_k \end{cases}$$

- **Does this look ok?**

Implications of the Halting Problem

- The "Halting Problem" is that of finding a TM that will tell whether the TM will diverge on its own description.
- The unsolvability of this problem does not hinge on TM's; it applies to any universal computing model (e.g. rex programs).

The Blank-Tape Halting Problem

- It can be shown that there is nothing special about requiring "its own description".
- Any standardized input, such as an **all blank tape**, will also suffice.
- The halting problem is still not solvable in this revised case.

The Reducibility Concept

- There are many other problems that are unsolvable.
- The standard way of proving that a given problem P is unsolvable is to "reduce" the halting problem to P.
- In other words, if P were solvable, then the halting problem would be also.
- The halting problem is not solvable, therefore P is not either.

There is no algorithm that will decide whether an arbitrary Program halts on an arbitrary input

- For example, the problem of devising an algorithm that determines whether an *arbitrary* Turing machine halts on an *arbitrary* input.
- The halting problem reduces to this problem, since if this problem were solvable, so would the halting problem be (as a special case).
- More on this in new Computer Science 81 (Computability and Logic)