WHAT IS A TURING MACHINE?

Foundations

Turing machines are used to investigate the nature of computation - computation that can be carried out by humans or machines. The initial fundamental question is: "Can one give a precise definition corresponding to the intuitive notion of being *computable*; i.e. of being able to be carried out using calculations that anyone can do given simple instructions and unlimited paper?; i.e. of being able to be specified by an *algorithm* ?"

Alan Turing (1912-1954) was an influential English mathematician who worked on the foundations of computing. It has been claimed that he was responsible for the first complete design of a stored program computer. In addition to his work on computing and mathematics, he played an important role in the British codebreaking work during World War 2. The Association for Computing Machinery (the major professional organisation of computer scientists in the U.S.A.) has recognised his contribution to computer science by naming their major annual award the Turing Award.

Interest in computation and algorithms goes back at least two thousand years, but it is only since the 1930's that there has been a serious attempt to determine the limits of what is computable - the study of *computability* . In 1936, Turing introduced the machines we call Turing machines in order to argue that all detailed sets of instructions that can be carried out by a human calculator can also be carried out by a suitably defined simple machine. Other researchers working in the 1930's looked at computability from different points of view - giving alternative formalisations to attempt to capture the (intuitive) notion of *computable function* - which all have turned out to be equivalent; i.e. it has been proved that any function computable by any of the proposed mechanisms can be computed using a Turing machine, and vice-versa. These remarkable results show that these definitions have managed to give the vague concept of "computability" a very precise meaning.

A Turing machine can be thought of as a computing device stripped to its essentials. It will be vastly less efficient than a modern computer, but, in compensation, much less complex. Such a device needs to be able to receive *input* , say on paper, to carry out *intermediate calculations* - say using "scratch paper" - that serve to record things for later reference in the calculation but which need not be shown when the final result is to be displayed, and to *display* the final result. At different times the device can invoke one of a number of "subroutines" which we can think of as different mental states. Calculations are not to require ingenuity or mathematical invention. The instructions for the device must all be fully determined and finitely described before any particular computation to which it is applied is selected; i.e. all possible types of steps are to be specified in advance. A typical calculation can be thought of as just replacing symbols by symbols (including erasing symbols, as we consider the blank to be one of the possible symbols). This replacement involves *reading* and *writing* symbols on the paper. The device can *move* to all parts of the paper to do computations. The amount of scratch paper can be thought of as unlimited in that we always allow the device to request more paper as needed. A fixed *finite alphabet* of symbols is part of the definition of each machine, as is the specification of the finitely many *states* .

One of the key outcomes of Turing's analysis of computation is the assertion that *every* algorithm can be programmed on some Turing machine. We come back to this point in the section 'What Turing machines can do'.

Turing machine organisation

We now describe the characteristics of a Turing machine. This description is rather informal as it is only intended to explain the general idea. More careful definitions may be found in textbooks on the Theory of Computation and the interested reader is urged to consult such a book. The reference list at the end of this section suggests some suitable books.

- The machine consists of a *control unit* , a *linear tape* that is divided into squares and a (read-write) *head* that scans one square of the tape at a time.

- The control unit can be set to one of a finite number of distinct internal states q_1, ..., q_m . At the start of each computation the control unit is in a specified *starting state* . The tape, at any given time, is finite but is potentially infinite in both directions in the sense that we assume a new blank square will be appended to the tape whenever needed. Symbols written on the tape are taken from a finite alphabet s_1, ..., s_k . Termination of a computation is signalled by the machine being set to one of a specified set of *final states* .

- The *program* for a given machine is a finite set of instructions, each instruction being of the form:

```
if currently in state q_i and scanning symbol s_j
    then
        replace s_j by s_k
        and move one square in direction D  /*D may be Left or Right*/
        and set machine to state q_p
```

i.e. each instruction can be abbreviated to (q_i, s_j, s_k, D, q_p) where D is one of L or R, q_p may be the same as q_i, and s_j may be the same as s_k.

- *Execution* of the program proceeds in a step by step fashion. Initially the control unit is set in the starting state and the tape head is in some specified position on the tape (commonly scanning the leftmost non-blank symbol). Then applicable instructions are obeyed until there is no further appropriate instruction and then the machine halts. It is possible for a Turing machine to keep running and never halt. If the machine halts in a final state, it is said to *accept* the input string - shown by the **ATMS** program as a "thumbs up" sign in the speedo dial. Otherwise, on termination, the program shows thumbs down.

Examples

Running the example machines on the distribution disk, using a variety of different tapes, should give you an idea of the general behaviour of Turing machines.

Variations

• Our Turing machine description allowed the designer of a Turing machine to use any finite alphabet of symbols for the tape. One of the basic theorems of the theory states that an alphabet with only two non-blank characters, e.g. 0, 1 and B (blank), is sufficient to build a machine to compute any function. [The **ATMS** program allows the designer to use any finite alphabet, but this theorem says that for each machine using an alphabet with more than two non-blank symbols, there is another (generally more complex) 'binary' Turing machine which only uses symbols 0, 1 and B and which computes exactly the same function; i.e. given the same inputs the machines produce the same outputs (generally the binary machine takes longer to complete the computation, and the binary machine starts with the input provided binary coded on its tape).]

• Our Turing machine description was in terms of a two way infinite tape. An alternative definition uses only a one way infinite tape (with the convention that trying to move past the end of the tape causes the machine to halt). Another basic theorem states that these two classes of Turing machines are equivalent, i.e. any function which can be computed by a Turing machine operating on two way tape can also be computed by some (other) Turing machine operating on a one way tape, and vice versa. [The **ATMS** program allows the creation of machines that operate on one-way tapes. See the **SETUP** menu.]

• There are a number of other variations which all turn out to lead to equivalent classes of machines. For example, some writers allow the machine on a single move to change state or move but not both [-the **ATMS** program allows the creation of such machines. See the **SETUP** menu.] Other variations involving the shape of the tape, the number of tape tracks, the number of tape heads, etcetera have not been implemented.

• A most important variation is to allow the machine to be *non-deterministic* , i.e. at each step to have a finite number of choices of moves that it can make. [The **ATMS** program allows the creation of such machines - see the **SETUP** menu.] Another basic theorem states that the classes of functions computable by deterministic and by non-deterministic Turing machines are the same. A more delicate question, to which the answer is not (yet) known, is how much longer, in general, it must take a deterministic machine to perform a computation compared with an equivalent (i.e. computes the same function) non-deterministic machine.

The 'stability' of the class of functions computable by Turing machines under all these variations is further evidence of the importance of this class of functions, and further evidence of the importance of Turing machines.

What Turing machines can do

Turing machines enable a precise account of what it means for a function over the natural numbers {0,1,2,...} to be *computable* . The initial content of the tape is the input to the function. If the machine halts, the final content of the tape is the output, or value, of the function. Turing machines can fairly easily be designed to compute functions such as addition and multiplication. Using an alphabet with at least the symbols, say # and +, represent 0 by #, 1 by ##, 2 by ###, etcetera. Use ##+##### as initial contents of the tape to request the value of 1+4, etcetera. Then a Turing machine that starts with such a tape, performs a computation, and halts with the final

contents of the tape representing the sum of the input digits (in this case ######) is said to *compute* addition. See the example

machine Unary_Adder provided with the distribution disk. So addition is said to be a (*Turing*) *computable function* .

What other kinds of functions are computable by Turing machines? It turns out that any function that can be computed by, say, a Pascal program (actually an idealised Pascal program in that one needs to assume it has access to unlimited storage) can also be computed by a Turing machine. Of course, since the Turing machine can only carry out very simple steps at a time, a Turing machine to compute some function will be more complex than the corresponding Pascal program (e.g. consider addition, as above) but, ignoring the issue of complexity, we can say that Turing machines and (idealised) Pascal programs have the same computing power.

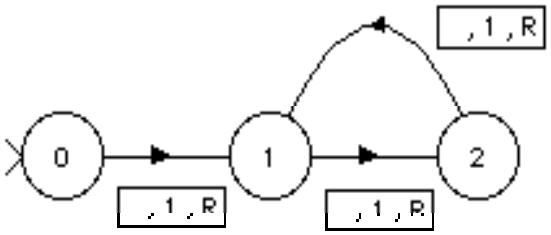
From the 1930's there have been many attempts to give a precise definition corresponding to the intuitive notion of *computable* . In each case it has been possible to prove that the functions computable according to the new definition were exactly the same as those computable according to the previously suggested definitions. As the the evidence accumulated, it became widely believed that Turing-computable functions corresponded exactly to the (intuitively) computable functions. As we are trying to match up an imprecise (intuitive) notion with a formal notion (Turing computability), it is not possible to prove that they are the same, but we have to rely on experimental evidence. The belief that any intuitively computable function is computable by a Turing machine (and so by any of the other mechanisms known to be equivalent to Turing machines) is referred to as *Church's hypothesis* , and sometimes as the *Church-Turing hypothesis* ; i.e. according to Church's hypothesis, Turing machines embrace all possible types of computation.

What Turing machines cannot do

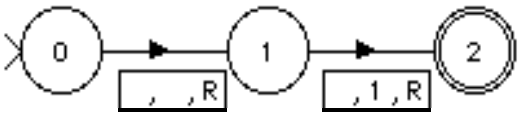
One of the key uses of a precise definition of computable function is to show that there are functions which are *non-computable* . Indeed, the motivation of Turing's original work on Turing machines was to show that a certain problem (to do with predicate calculus) was non-computable.

There are many non-computable (also called *unsolvable*) problems which arise in computer science. We shall briefly describe one non-computable function which deals with the operation of Turing machines. It is called the "busy beaver" problem. To explain this problem, we need the notion of the *productivity of a Turing machine* . Consider Turing machines that use only the symbol 1 and a blank symbol ( ). Initially the tape is blank. If the machine eventually halts, scanning the leftmost of an unbroken string of 1's on an otherwise blank tape, its *productivity* is said to be the number of 1's in that string. If the machine never halts, or halts with something other than an an unbroken string of 1's, then its *productivity* is said to be 0.

Example:
The following three state machine has productivity 0.



The next three state machine has productivity 1.



And the next three state machine has productivity 2.

Let p(n) = the productivity of the most productive n-state machine. The *busy beaver problem* is to design a Turing machine, which uses only the symbol 1 and the blank symbol, and which, started in the starting state, scanning the leftmost of a string of n 1's on an otherwise blank tape, eventually halts, scanning the leftmost of a string of p(n) 1's on an otherwise blank tape. It can be proved that there is no such Turing machine, i.e. the function p is non-computable.

The books listed later contain descriptions of many unsolvable problems which arise in computer science.

Feasible computation

Turing machines enable the study of functions that are computable, but, as presented above, without major concern for how many steps it may take before the machine halts, or how much tape is actually needed for a given computation. Since the mid-60's there has been a strong interest in identifying the classes of problems and functions that are *practical* or feasible to compute, as measured by the amount of time and/or space required for the computation. Deterministic and non-deterministic Turing machines also play an important role in such analyses. The interested reader is again referred to books such as those listed below.

References

History

J N Crossley (ed.), 'Reminiscences of Logicians', in Algebra and Logic, Springer Lecture Notes in Mathematics, Volume 450, 1975, pp3-62.

J N Crossley, 'Fifty Years of Computability', Logic Paper No. 62, Department of Mathematics, Monash University, Australia, August 1987.

Turing's original paper has been reproduced in:
M Davis, 'The Undecidable' Raven Press, 1965, pp115-154.

A. Hodges, 'Alan Turing: The Enigma', Simon and Schuster, 1983.

S C Kleene, 'Origins of Recursive Function Theory', Annals of the History of Computing, Volume 3, Number 1, January 1981, pp52-67.

M Davis, 'Why Godel didn't have Church's Thesis', Information and Control, Volume 54, 1982, pp3-24.

Theory

These books contain technical accounts of the theory of Turing machines, and relevant applications. There are many other useful books on these topics which have not been listed.

F S Beckmann, 'Mathematical Foundations of Programming', Addison Wesley, 1980

J D Hopcroft and J E Ullman, 'Introduction to Automata Theory, Languages, and Computation, Addison Wesley, 1979.

A J Kfoury, R N Moll, and M A Arbib, 'A Programming Approach to Computability', Springer Verlag, 1982.

J S Mallozzi and N J DeLillo, 'Computability with Pascal', Prentice-Hall, 1984.

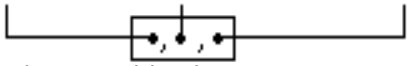
V J Rayward-Smith, 'A First Course in Computability', Blackwell Scientific, 1986.

QUICK REFERENCE

General Setup:

Get setup dialog box by selecting Setup in Operate menu.

Blank Symbol	Character typed is blank symbol.
Max computations	If enabled, limits no. of transitions executed.
Write AND move	Makes transition box: read symbol write symbol direction of tape head move(R or L)



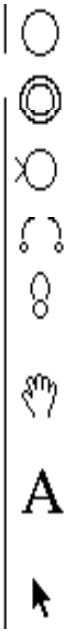
Write OR move	Makes transition box: read symbol

write symbol (not R or L) **or**
direction of tape head move(R or L).

Auto transition box addition	When enabled automatically adds a transition box to all arcs directly after drawing them.
One-way tape Two-way tape	selects one way or two way "infinite" tape.
Record history	If enabled allows deterministic TM to reverse. TM runs faster if disabled. No effect on non-deterministic TM.
Overview	If enabled scales TM picture to show entire machine while running.
Deterministic non-deterministic	Sets ATMS to simulate deterministic TMs only or non-deterministic ones as well.

Edit a TM:

Use tools:



Add (or convert to) a normal state.

Add (or convert to) a final state.

Add (or convert to) the initial state.

Drag arc between two different states.

Drag arc with same origin and destination out of a state.

Scroll drawing by dragging on editing window. Double click to show page.

Drag transition boxes out of arrows on arcs. Click in transition boxes to edit contents. Edit labels in states.

Drag states or transition boxes to new locations. Change shape of arcs by dragging arrow. Click in state or transition box or in arrow of arc to mark for deletion.

Holding down <option> converts other tools to arrow.

Delete arcs, states or transition boxes by marking with arrow tool, then press <backspace>.

Set Machine Title in Operate Menu:

Edits machine title and then puts it in bottom right hand corner of drawing.

Set Drawing Size in Windows menu

Set the maximum size of TM by dragging corner of black rectangle. Can be changed even if there is a TM already. Cannot be shrunk smaller than the present TM. Shows the page breaks as setup in page setup in File menu.

Show page in Windows menu or double click on hand tool:

Shows the entire TM. The grey frame can be moved around to select the view. Then press OK button.

Edit tape:

Use tools:

Click at insertion position, type. <Backspace> erases one character
Clear Tape in Operate menu clears entire tape.

Click on tape to set tape head position.

Use arrows on side of tape to :

Scroll tape fast/slow, right/left.

Run deterministic TM:

Check in Operate menu or <clover K> checks the TM for inconsistencies. If inconsistent, leaves a blob in a state or transition box close to the inconsistency. A check is automatically done before running.

Reset in Operate menu or <clover R> resets tape to last edited or saved state and TM into start state.

Go in Operate menu, Go button above dial or <clover G> start TM running, change button to Pause.
Pause button (same location as Go button) stops TM temporarily, changes button to Cont(inue).
Cont(inue) (same location as Go button) button continues TM running.

Run Turing machine while button held down. Click to single step.

Run Turing Machine in reverse, only when record history is on. (Use Setup in Operate menu.)

When tape is accepted "Thumbs up" symbol appears in speedo dial:

When tape is rejected "Thumbs down" appears in speedo dial:

Increase/decrease TM running speed by pressing buttons below speedo dial, when machine is stopped.

Run non-deterministic TM:

Press Go button or select Go in Operate menu or <clover G>:

ATMS puts up dialog box with buttons (possibly not all active):

View Shows state of the currently successfully explored TM, allows reversing (RR button) or forwarding (FF button) through states that

led to this accepting state. The Go button is changed to Next and is used to find the next solution.

Find next Tries to find next alternative accepting solution.

Cancel Stops any further action of TM.

If no possible path to acceptance can be found, "thumbs down" is displayed in dial.
Computation may take a long time (possibly of exponential order).
To repeat from start use Reset in Operate menu or <clover R>, before pressing Go.

Print TM:

Make sure printer is set up (Chooser in Apple menu).
Use Page Setup in File menu to set print page size and reduction or enlargement.
Use Print in File menu <clover P> to print.

TM to/from disk:

Select Save as... from File menu.

Click in Set File Type button to select in next dialog box:

Normal:to save TM and remembered tape in ATMS file format

Machine only: to save TM only in ATMS file format

Tape only: to save tape in ATMS file format.

Text: to save the complete TM as a standard set representation in text format (readable by Mac Write, ATMS can't use this.).

Pict: to save the TM picture in pict format (readable by MacDraw, ATMS can't use this.).

Select Save from File menu or <clover S>:

Same as Save as ... , but displays dialog box only when not previously saved in ATMS file format.

Select Close from File menu <clover W>:

Closes TM, asks to save if modified since last save.

Select Open from File menu <clover O>:

Puts up standard open box to select a file in ATMS format for opening. If selected file contains a TM, Close is executed first.

Others:

To move windows: Hold down <clover> and drag.
Quit in File menu <clover Q>
Edit menu used for desk accessories only. (NO cut, copy, paste or undo on TM editing).