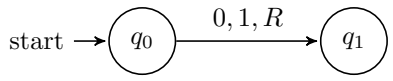
tms.1 Representing Turing Machines

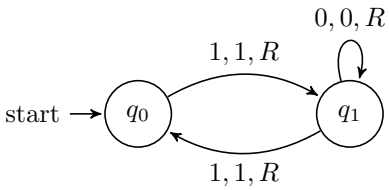
tms:tms:rep:sec

Turing machines can be represented visually by *state diagrams*. The diagrams are composed of state cells connected by arrows. Unsurprisingly, each state cell represents a state of the machine. Each arrow represents an instruction that can be carried out from that state, with the specifics of the instruction written above or below the appropriate arrow. Consider the following machine, which has only two internal states, q_0 and q_1, and one instruction: explanation



Recall that the Turing machine has a read/write head and a tape with the input written on it. The instruction can be read as *if reading a blank in state q_0, write a stroke, move right, and move to state q_1*. This is equivalent to the transition function mapping $\langle q_0, 0\rangle$ to $\langle q_1, 1, R\rangle$.

Example tms.1. *Even Machine*: The following Turing machine halts if, and only if, there are an even number of strokes on the tape.



The state diagram corresponds to the following transition function:

$$\delta(q_0, 1) = \langle q_1, 1, R\rangle,$$
$$\delta(q_1, 1) = \langle q_0, 1, R\rangle,$$
$$\delta(q_1, 0) = \langle q_1, 0, R\rangle$$

The above machine halts only when the input is an even number of strokes. Otherwise, the machine (theoretically) continues to operate indefinitely. For any machine and input, it is possible to trace through the *configurations* of the machine in order to determine the output. We will give a formal definition of configurations later. For now, we can intuitively think of configurations as a series of diagrams showing the state of the machine at any point in time during operation. Configurations show the content of the tape, the state of the machine and the location of the read/write head. explanation

Let us trace through the configurations of the even machine if it is started with an input of 4 1s. In this case, we expect that the machine will halt. We will then run the machine on an input of 3 1s, where the machine will run forever.

The machine starts in state q_0, scanning the leftmost 1. We can represent the initial state of the machine as follows:

$$\triangleright 1_0 1110\ldots$$

The above configuration is straightforward. As can be seen, the machine starts in state one, scanning the leftmost 1. This is represented by a subscript of the state name on the first 1. The applicable instruction at this point is $\delta(q_0, 1) = \langle q_1, 1, R\rangle$, and so the machine moves right on the tape and changes to state q_1.

$$\triangleright 11_1 110\ldots$$

Since the machine is now in state q_1 scanning a stroke, we have to "follow" the instruction $\delta(q_1, 1) = \langle q_0, 1, R\rangle$. This results in the configuration

$$\triangleright 111_0 10\ldots$$

As the machine continues, the rules are applied again in the same order, resulting in the following two configurations:

$$\triangleright 1111_1 0\ldots$$

$$\triangleright 11110_0\ldots$$

The machine is now in state q_0 scanning a blank. Based on the transition diagram, we can easily see that there is no instruction to be carried out, and thus the machine has halted. This means that the input has been accepted.

Suppose next we start the machine with an input of three strokes. The first few configurations are similar, as the same instructions are carried out, with only a small difference of the tape input:

$$\triangleright 1_0 110\ldots$$

$$\triangleright 11_1 10\ldots$$

$$\triangleright 111_0 0\ldots$$

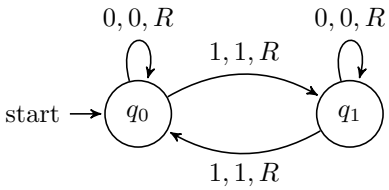
$$\triangleright 1110_1\ldots$$

The machine has now traversed past all the strokes, and is reading a blank in state q_1. As shown in the diagram, there is an instruction of the form $\delta(q_1, 0) = \langle q_1, 0, R\rangle$. Since the tape is infinitely blank to the right, the machine will continue to execute this instruction *forever*, staying in state q_1 and moving ever further to the right. The machine will never halt, and does not accept the input.

explanation

It is important to note that not all machines will halt. If halting means that the machine runs out of instructions to execute, then we can create a machine that never halts simply by ensuring that there is an outgoing arrow for each symbol at each state. The even machine can be modified to run infinitely by adding an instruction for scanning a blank at q_0.

Example tms.2.



explanation

Machine tables are another way of representing Turing machines. Machine tables have the tape alphabet displayed on the x-axis, and the set of machine states across the y-axis. Inside the table, at the intersection of each state and symbol, is written the rest of the instruction—the new state, new symbol, and direction of movement. Machine tables make it easy to determine in what state, and for what symbol, the machine halts. Whenever there is a gap in the table is a possible point for the machine to halt. Unlike state diagrams and instruction sets, where the points at which the machine halts are not always immediately obvious, any halting points are quickly identified by finding the gaps in the machine table.

Example tms.3. The machine table for the even machine is:

	0	1
q_0		$1, q_1, R$
q_1	$0, q_1, 0$	$1, q_0, R$

As we can see, the machine halts when scanning a blank in state q_0.

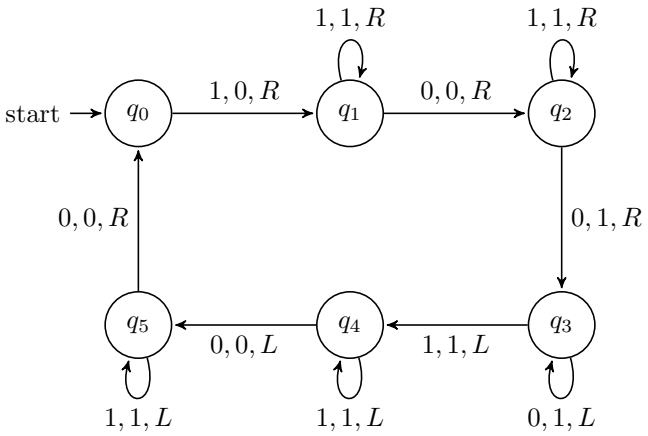
explanation

So far we have only considered machines that read and accept input. However, Turing machines have the capacity to both read and write. An example of such a machine (although there are many, many examples) is a *doubler*. A doubler, when started with a block of n strokes on the tape, outputs a block of $2n$ strokes.

tms:tms:rep:ex:doubler

Example tms.4. Before building a doubler machine, it is important to come up with a *strategy* for solving the problem. Since the machine (as we have formulated it) cannot remember how many strokes it has read, we need to come up with a way to keep track of all the strokes on the tape. One such way is to separate the output from the input with a blank. The machine can then erase the first stroke from the input, traverse over the rest of the input, leave a blank, and write two new strokes. The machine will then go back and find the second stroke in the input, and double that one as well. For each one stroke of input, it will write two strokes of output. By erasing the input as the machine goes, we can guarantee that no stroke is missed or doubled twice. When the

entire input is erased, there will be $2n$ strokes left on the tape.



Problem tms.1. Choose an arbitary input and trace through the configurations of the doubler machine in Example tms.4.

Problem tms.2. The double machine in Example tms.4 writes its output to the right of the input. Come up with a new method for solving the doubler problem which generates its output immediately to the right of the end-of-tape marker. Build a machine that executes your method. Check that your machine works by tracing through the configurations.

Problem tms.3. Design a Turing-machine with alphabet $\{0, A, B\}$ that accepts any string of As and Bs where the number of As is the same as the number of Bs *and* all the As precede all the Bs, and rejects any string where the number of As is not equal to the number of Bs or the As do not precede all the Bs. (E.g., the machine should accept $AABB$, and $AAABBB$, but reject both AAB and $AABBAABB$.)

Problem tms.4. Design a Turing-machine with alphabet $\{0, A, B\}$ that takes as input any string α of As and Bs and duplicates them to produce an output of the form $\alpha\alpha$. (E.g. input $ABBA$ should result in output $ABBAABBA$).

Problem tms.5. *Alphabetical?:* Design a Turing-machine with alphabet $\{0, A, B\}$ that when given as input a finite sequence of As and Bs checks to see if all the As appear left of all the Bs or not. The machine should leave the input string on the tape, and output either halt if the string is "alphabetical", or loop forever if the string is not.

Problem tms.6. *Alphabetizer:* Design a Turing-machine with alphabet $\{0, A, B\}$ that takes as input a finite sequence of As and Bs rearranges them so that all the As are to the left of all the Bs. (e.g., the sequence $BABAA$ should become the sequence $AAABB$, and the sequence $ABBABB$ should become the sequence $AABBBB$).

Photo Credits

Bibliography