Notes on Finite Automata

> Turing machines are widely considered to be the abstract proptotype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.
>
> In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.
>
> From the Introduction to *Finite Automata and their Decision Problems,* Rabin & Scott, IBM Journal, April 1959, pp114-25.S

0.1 The intuitive model and basic definitions

Again, we quote from Rabin and Scott:

> An automaton will be considered as a black box of which questions can be asked and from which a "yes" or "no" answer is obtained. The number of questions that can be asked will be infinite, adn for simplicity a question is interpreted as any arbitrary finite sequence of symbols from a finite alphabet given in advance. ...
>
> ... we shall ... think of the question as given on one-dimensional tapes. The machine will be endowed with a reading head which can read one square of the tape (i.e. one symbol) at a time, and then it can advance the tape one unit and read, say, the next square to the right.

> We assume the machine stops when it runs out of tape. So much for the external character of an automaton.
>
> The internal workings of an automaton will not be analyzed too deeply. We are not concerned with how the machine is built but with what it can do. The definition of the internal structure must be general enough to cover all conceivable machines, but it need not involve itself with problems of circuitry. The simple method of obtaining generality without unnecessary detail is to use the concept of *internal states.* No matter how many wiresor tubes or relays the machine contains, its operation is determined by stable states of the machine at discrete time intervals. An actual existing machine may have billions of such internal states, but the number is not important from the theoretical standpoint—only the fact that it is finite.
>
> ... if we make a table of all the transitions from a state and a symbol to a new state, then the whole action of the machine is essentially described.
>
> Finally, to get an answer from the machine, ... we need only distinguish .. those states in which the "yes" light is on ... The whole machine is described when a class of designated states corresponding to the "yes" answers is given. *ibid.*, Chapter I. p115.

Here is Rabin and Scott's formal definition of an automaton.[1]

Definition 1. A (finite) automaton over the alphabet Σ is a system

$$\mathfrak{A} = \langle Q, \Sigma, q_0, A, N \rangle,$$

where Q is a finite non-empty set (the *internal states* of $\mathfrak{A}$), the *alphabet* Σ is a set whose elements we call *symbols*, the *next-state function* N is a function defined on the Cartesian product $Q \times \Sigma$ of all pairs of states and symbols to give values in Q (defining *transitions* or *moves* of $\mathfrak{A}$), the *initial state* $q_0 \in Q$ is an element of Q (of $\mathfrak{A}$), and $A \subseteq Q$ is a subset of Q (the *final*, or *accepting* states of $\mathfrak{A}$).

This definition has stood the test of time.

For small automata, it is often helpful to draw a *transition graph* rather than listing the various elements of the formal definition. Each state is represented by a node in this graph, and there is an arrow—a *transition*—labelled a from state s to state t iff $t = M(s, a)$. We use circles to represent the nodes (states). We use

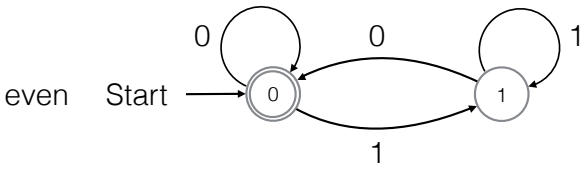
[1] The symbol $\mathfrak{A}$ is a capital A in the *Fraktur* or *Gothic* typeface. Typesetting in Fraktur—fractured, or broken type, known in German as *Gebrochene Schrift*—was common in German-speaking countries until the mid 20th century. Individual Fraktur letters are often used in mathematics: $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \mathfrak{D}, \mathfrak{E}, \mathfrak{F}, \mathfrak{G}, \mathfrak{H}, \mathfrak{I}, \mathfrak{J}, \mathfrak{K}, \mathfrak{L}, \mathfrak{M}, \mathfrak{N}, \mathfrak{O}, \mathfrak{P}, \mathfrak{Q}, \mathfrak{R}, \mathfrak{S}, \mathfrak{T}, \mathfrak{U}, \mathfrak{V}, \mathfrak{W}, \mathfrak{X}, \mathfrak{Y}, \mathfrak{Z}$.

an arrow, which looks just like an unlabelled transition coming from nowhere, to indicate the starting state, s_0, and a double ring around each accepting state $t \in F$.

To recap, states are nodes and transitions are labelled arrows in a directed graph. The start state is indicated by an arrow, coming from nowhere. An accepting state is indicated by a double border. Each transition is labelled with a symbol from the alphabet.

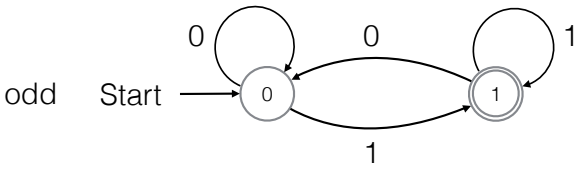
Examples 2. We start with two complementary examples.

2.1: Here is an automaton (we call it even) that accepts binary numbers (strings of zeros and ones) divisible by 2.



This machine is simple. Its alphabet is $\{0, 1\}$. The transitions labelled 0 end in state 0, and the transitions labelled 1 end in state 1. So this machine starts in state 0, then moves from state-to-state, as required to ensure that its state always corresponds to the last symbol read.

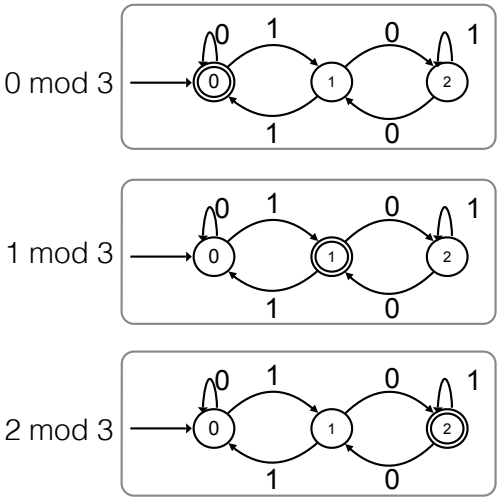
2.2: We can change the accepting state to create a machine (odd) that accepts exactly the odd binary numbers.



These examples are complementary in the sense that they give complementary answers: one answers, "Yes", when the other answers, "No", and *vice versa*. We can create a similar complement for any machine, simply by replacing $F \subseteq S$ by its complement, $\bar{F} = S \setminus F$, in the definition.

Taking stock, we have seen that a finite automaton can recognise the difference between odd and even binary numbers—this is not surprising: we can even say it would be odd if it could not!

Examples 3. Moving from two to three gives three more interesting examples.



These three machines share the same set of states, and the same transition function. These keep track of the value $\bmod 3$, of the binary number they have seen. If we write b for some sequence of binary digits, and $[\![b]\!]$ for the number it represents, then

$$[\![b\,0]\!] = 2 \times [\![b]\!] \qquad \text{and} \qquad [\![b\,1]\!] = 2 \times [\![b]\!] + 1$$

For example, if $b = 1\,1\,1$ then, $[\![b]\!] = 7$, and we have,

$$[\![1\,1\,1\,0]\!] = 14 \qquad \text{and} \qquad [\![1\,1\,1\,1]\!] = 15$$

Now $7 \bmod 3 = 1$ so the two equations, $14 \bmod 3 = 2$, and $15 \bmod 3 = 0$, correspond to the two transitions from state 1, to state 2 and state 0 respectively. These values correspond to the middle row of the full transition function igiven by the following table:

$[\![b]\!] \bmod 3$	$[\![b\,0]\!] \bmod 3$	$[\![b\,1]\!] \bmod 3$
0	0	1
1	2	0
2	1	2

where each entry in the $[\![b]\!] \bmod 3$ column corresponds to a different current state, and the two following columns give the next state determined by the input of a 0 or 1, respectively.

By a *tape* we shall understand any finite sequence of symbols from Σ. A machine processes an input tape by starting its start state, s_0, then following a series of transitions, from state-to-state, whose labels correspond to the sequence of symbols

on the tape.[2] The sequence of states visited by this process is called the *trace* of the computation. For example, for both machines in Example 2 above, the trace for an input x is just $0x$.

The set of all tapes is denoted by Σ^*, the set of finite sequences of symbols, which includes the empty tape with no symbols, denoted by Λ. For each input tape $x = (x_1, \ldots, x_n)$, where n is the length of x, the trace is a finite sequence of states, that is, an element of S^*.

Definition 4. For a machine M with alphabet Σ, and a tape $x = (x_0, \ldots, x_{n-1}) \in \Sigma^*$ the *trace* $\mathfrak{T}(M, x) \in S^*$ is the sequence $(s_0, \ldots, s_n)$ of states where s_0 is the starting state of M and the remaining steps are defined by induction:

$$s_{i+1} = M(s_i, x_i)$$

We also extend the function M, defined on $S \times \Sigma$, to a function M defined on $S \times \Sigma^*$.

$$M(s, \Lambda) = s$$

To process Λ we do nothing.

$$M(s, x\sigma) = M(M(s, x), \sigma)$$

To process $x\sigma$ first process x then step to the state determined by σ.

So, if the machine $\mathfrak{A} = (S, M, s_0, F)$ processes a tape x starting from s_0, it will terminate in the state $t = M(s_0, x)$, and give the answer "yes" just in case $t \in F$.

This is a natural definition. Every symbol can be viewed as a tape of length 1. The definition of the automaton determines the values of $M(s, \sigma)$, and this definition extends the function so that $M(s, xy) = M(M(s, x), y)$. To process xy first process x and then process y.

In presenting examples, we may present M as a *partial function*: for some state-symbol pairs we show no next state. We can view this, for the time being, as short-hand for a machine with a distinguished sink state, or *black hole* from which there is no escape. We don't show the black hole state, and for any state-symbol pair that is not shown the convention is that the next state is the black hole (which is not an accepting state). once a trace falls into a black hole, it cannot escape.

[2]Once a symbol has been read, the tape moves on. In this model of computation there is no going back to look at the same symbol twice.

Theorem 5. *No finite automaton can recognise the language*

$$\left\{0^n 1^n \mid n \in \mathbb{N}\right\},$$

that consists of strings that consist of some number of zeros follwed by the same number of ones.

Proof. Let M be a DFA with $k(>0)$ states. Let $n = 2 \times k$. Then any trace of the states $(x_1, \ldots, x_n)$ visited for an input of n zeros in succession, must visit some state more than once, say $s = x_p = x_q$, where $p < q$. Now consider the traces for the following two input sequences: let a_i be the trace of states visited for the sequence of q zeros followed by q ones; let b_j be the trace of states visited for the sequence of p zeros followed by q ones. By construction $a_q = b_p = s$. Beyond this state, both inputs continue with q ones, so, by induction on i, $a_{q+i} = b_{p+i}$ for all $i \leq q$. In particular, the two traces end in the same state, but one input has equal numbers of ones and zeros, while the other does not. □