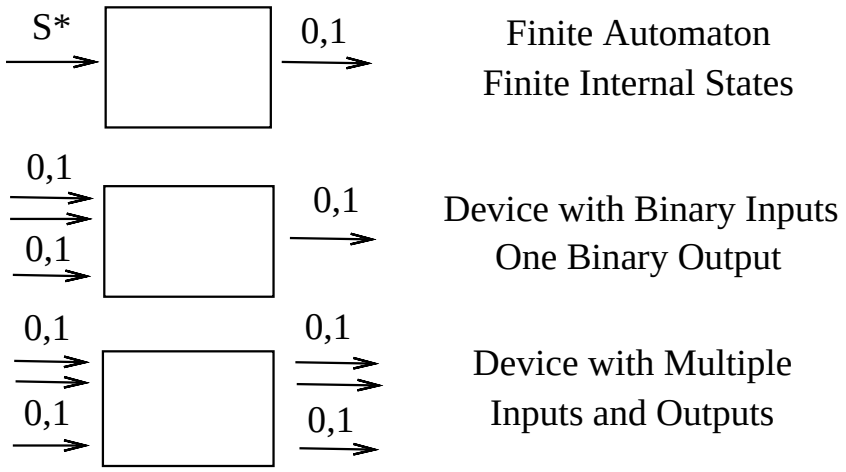
1 Deterministic Finite Automata



A finite automaton M is a device with finitely many internal states that receives input and responds "yes" or "no" depending on whether the input sequence seen so far is in the language or not.

K is the (finite) set of internal states.

Σ is the input alphabet.

It is assumed that time is discrete. The state at time $t+1$ is a function of the state at time t and the input at time t.

$s \in K$ is the initial state (at time $t = 0$).

If $q \in K$ is the state at time t and $a \in \Sigma$ is the input at time t, $\delta(q, a)$ tells which state the automaton will go into in time $t + 1$.

The output at time t is either "yes" or "no" depending on the inputs received so far.

$F \subseteq K$ is the set of accepting states, indicating that a string in the language has been seen.

The output is "yes" at time t if the state q at time t is in F, "no" otherwise.

If after seeing an input string, the automaton is in an accepting state, we say the automaton *accepts* the input.

Such an automaton M is reprsented by the 5-tuple

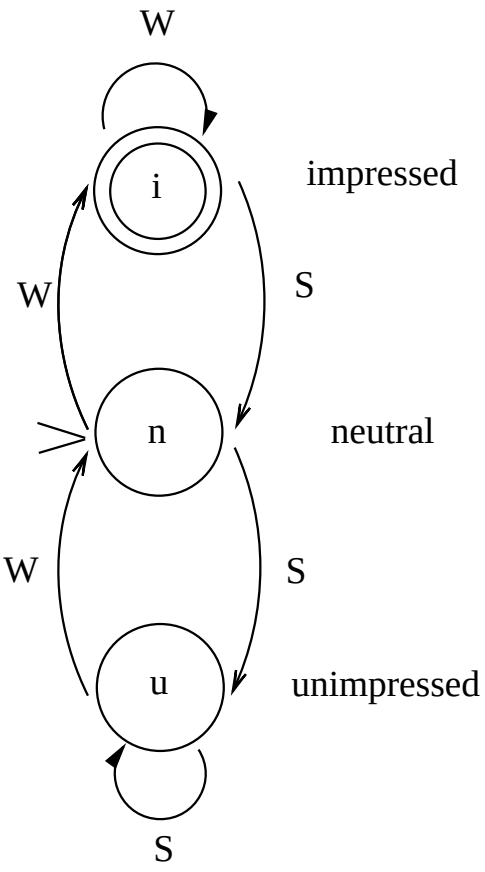
$$(K, \Sigma, \delta, s, F).$$

1.1 Automata as Graphs

It is often more convenient to represent automata by graphs. The states are circles, the arrows indicate δ, the start state is indicated by a "carat" pointing to it, and the accepting states have double circles.

1.2 Interview Automaton

Here is an example.



An interviewer has three states, impressed, neutral, and unimipressed. Initially the interviewer is neutral. If the student says something witty, then the interviewer has a better opinion. If the student says something stupid, the interviewer has a worse opinion.

The sequence WSSW is not accepted. The sequence WSW is accepted.

This automaton can be represented by the 5-tuple $(K, \Sigma, \delta, s, F)$ where K is $\{i, n, u\}$, Σ is $\{W, S\}$, s is n, F is $\{i\}$, and δ is defined by $\delta(n, W) = i$, $\delta(n, S) = u$, $\delta(i, W) = i$, $\delta(i, S) = n$, $\delta(u, W) = n$, and $\delta(u, S) = u$. In general, if there is an arrow from state s to state t labeled with a, then $\delta(s, a) = t$.

F can be empty; then the automaton doesn't accept any strings. If the automaton gets into a state in F and receives more input, it just keeps going. The start state may or may not be an accepting state.

1.3 Formalism for Automata

A *configuration* of a finite automaton M is an element of $K \times \Sigma^*$. It tells the current state and the inputs to be read in the future.

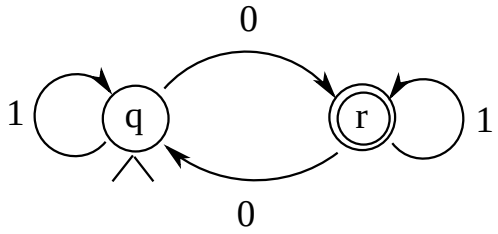
$(q, w) \vdash_M (q', w')$ if the automaton M can pass from the configuration (q, w) to (q', w') in one step. We say (q, w) *yields* (q', w') *in one step.*

For the interview automaton M, $(n, WSSW) \vdash_M (i, SSW) \vdash_M (n, SW) \vdash_M (u, W) \vdash_M (n, \epsilon)$.

We write $\vdash_M^*$ for the reflexive transitive closure of $\vdash_M$. Thus for the interview automaton, $(n, WSSW) \vdash_M^* (n, \epsilon)$. Also, $(n, WSSW) \vdash_M^* (n, WSSW)$. We say $(n, WSSW)$ *yields* (n, ϵ), as an example.

A string $w \in \Sigma^*$ is *accepted* by a finite automaton M if there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, \epsilon)$. The string ϵ is accepted if the start state is an accepting state. Also, the *language recognized by* M, $L(M)$, is $\{w \in \Sigma^* : M \text{ accepts } w\}$. We say that M *recognizes* the language $L(M)$.

Here is another example:

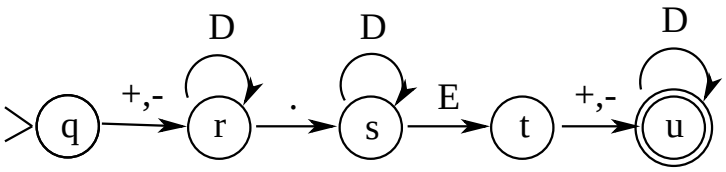


1.4 Problems

What is the 5-tuple for this automaton?

Can you give a finite automaton to recognize the set of binary strings that have an even number of zeroes and an even number of ones?

Finite automata can be helpful for the lexicographic phase of compilation. Here is an approximate automaton for recognizing floating point numbers:



Finite automata can be simulated on spread sheets.

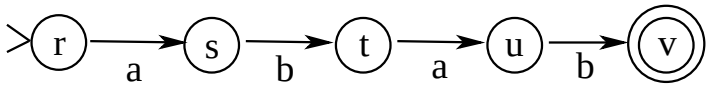
Applications of finite automata: String searching, protein sequence similarity testing, and hardware design.

How to construct a finite automaton for a language L over Σ:

1. Divide all strings over Σ into a finite number of "equivalent" subsets.
2. Choose a state for each subset.
3. Put arrows between the states corresponding to the subsets.
4. The start state is the subset containing the empty string.
5. The accepting states are the subsets that are in L.

Illustrate this procedure on the automaton for the binary strings with an even number of zeroes and an even number of ones.

Problem 2.1.3 (b) page 60: Add arrows to the following automaton so that it accepts the strings over $\{a, b\}$ that contain the substring *abab*.



Now do it for the set of strings that end with *abab*.

What is an efficient way to implement finite automata on a computer?

Now see how automata can be implemented on spread sheets.

Note that automata represent structures that arise naturally in many areas, even if they are not called automata. Therefore a knowledge of automata can help to understand and prove properites in such areas and also to generate efficient algorithms and code.