1 Nondeterministic Finite Automata

Suppose in life,

- whenever you had a choice, you could try both possibilities and live your life.
- At the end, you would go back and choose the one that worked out the best.
- Then you could decide who to marry, which job to accept, or which answer to give on an exam knowing the future consequences.

This is the idea of nondeterminism for finite automata. It's not practical to build devices like this, but it does help in applications of finite automata to make use of this concept.

- In a deterministic automaton, in a state s with an input a there is one and only one state t that it can go to in the next time instant.
- In a nondeterministic automaton, there can be many states $t_1, t_2, \ldots, t_n$ that it can go to, or possibly no states at all.
- Also, nondeterministic automata have the possibility to go from one state to another without reading any input, if there is an ϵ transition between the two states.

1.1 Formalism

Formally,

- instead of a transition function δ there is a transition relation Δ with $\Delta \subseteq K \times (\Sigma \cup \{\epsilon\}) \times K$.
- If $(q, u, p) \in \Delta$ then in state q, reading a u, the automaton can go to state p.
- Also, u can be ϵ, in which case the automaton can go from state q to state p without reading anything.

Therefore

- a nondeterministic finite automaton can be represented as a quintuple $(K, \Sigma, \Delta, s, F)$, much as before.
- It can also be represented as a graph; the difference is that there can be more than one arrow labelled with the same symbol coming out of a state q, or possibly none at all.

The computation of a nondeterministic automaton is similar to that of a deterministic automaton, except that

> for configurations, $(q, w) \vdash_M (q', w')$ iff there is a u in $\Sigma \cup \{\epsilon\}$ such that $w = uw'$ and $(q, u, q') \in \Delta$.

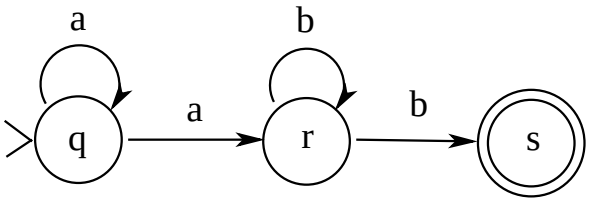
Note that u can be ϵ. Then $\vdash_M^*$ is defined as before, and we say

- M accepts w if there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, e)$.
- Also, $L(M)$, the language recognized by M, is the set of strings accepted by M.

Deterministic finite automata can be regarded as a special case of nondeterministic finite automata, in which there are no ϵ transitions and Δ is a function.

1.2 Example

Here is an example:



Why is this automaton nondeterministic? What language does it recognize? What are K, Σ, Δ, s, and F for it?

This automaton rejects the strings *aa* and *aba*.

Consider the computation of this automaton on the input *aabb*. There is more than one possibility:

$q \xrightarrow{a} q \xrightarrow{a} r \xrightarrow{b} r \xrightarrow{b} s$ leads to acceptance.

$q \xrightarrow{a} r$ leads to failure.

Because there is at least one computation sequence leading to acceptance the string *aabb* is accepted by this automaton.

To show that a nondeterministic automaton rejects an input string, it is necessary to examine all computation sequences.

- There could be exponentially many possible computations on a given input.
- This makes simulating the automaton very expensive.
- To increase the efficiency, we can summarize all computations of this automaton on this input using sets this way:

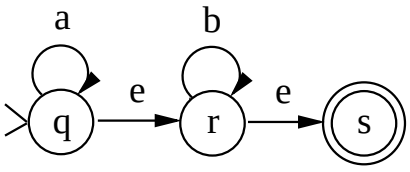
$$\{q\} \xrightarrow{a} \{q, r\} \xrightarrow{a} \{q, r\} \xrightarrow{b} \{r, s\} \xrightarrow{b} \{r, s\}$$

1.2.1 Problem

Find an equivalent deterministic finite automaton.

1.3 Example

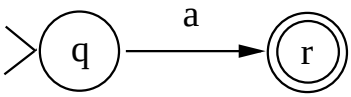
Here is a nondeterministic automaton with ϵ arrows:



Simulate this automaton on inputs aa and ba. The ϵ arrows make this harder to do. Give K, Σ, Δ, s, and F for this automaton. What language does it recognize?

1.4 Example

Note that a nondeterministic automaton may not be able to do anything. Consider the following example:



If it receives a b as input, it can't do anything.

1.5 Problem

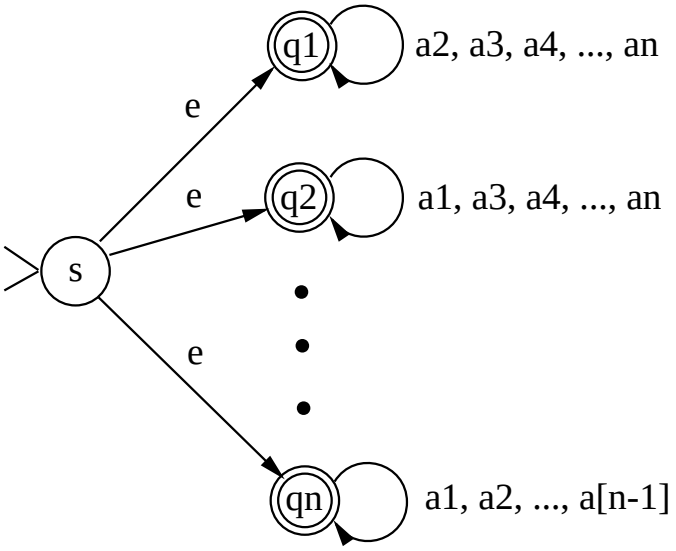
Give a nondeterministic finite automaton recognizing the set of strings over $\{a, b\}$ having $abab$ as a substring, and make it as simple as possible.

Also, do it for the set of strings over $\{a, b\}$ ending with $abab$.

1.6 Complexity of Nondeterministic Automata

Sometimes nondeterministic automata can be much simpler than any equivalent deterministic automaton. (We say two automata M_1 and M_2 are *equivalent* if $L(M_1) = L(M_2)$).

- Let Σ be $\{a_1, a_2, \ldots, a_n\}$ and let L be the set of strings w that do not contain all the symbols $\{a_1, a_2, \ldots, a_n\}$.
- That is, at least one symbol must be missing from w.
- This language can be recognized by the following $n+1$ state nondeterministic finite automaton.
- Any equivalent deterministic automaton has at least 2^n states:



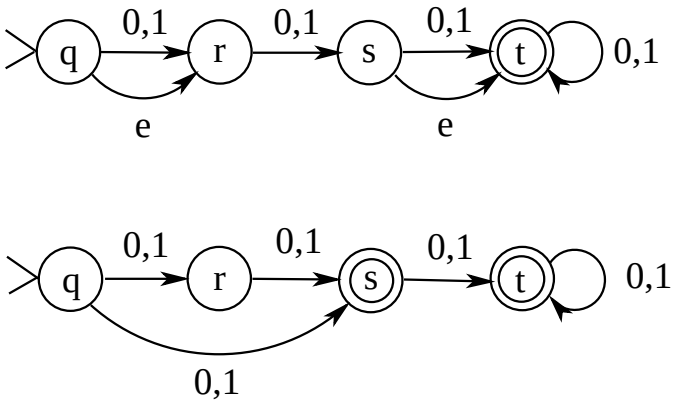
1.6.1 Problem

Consider this automaton for $n = 3$ and using the set idea, simulate this automaton on the inputs $a_1a_2a_3$ and on a_1a_3.

1.7 egrep and grep

- In Unix, grep works directly with a nondeterministic automaton and simulates it using sets of states.
- egrep constructs an equivalent deterministic automaton and simulates it.
- Thus egrep can take a long time on complicated queries, because the automaton may have many states.
- But if the string searched is very long, this can pay off.

1.8 Omitting ϵ arrows



How can one eliminate ϵ arrows from the automaton given earlier for the set of strings missing at least one symbol?

Theorem 1.1 *For each nondeterministic finite automaton M there is an equivalent nondeterministic finite automaton M'' without ϵ arrows and having the same number of states.*

Proof: We obtain automaton M' from M by applying two transformation rules as often as possible until they can be no more applied. If in M we have

$$q \xrightarrow{\epsilon} r \xrightarrow{a} s$$

for $a \in \Sigma$ then in M' we have also

$$q \xrightarrow{a} s.$$

Also, if in M we have

$$q \xrightarrow{\epsilon} r$$

and $r \in F$ then in M', q is also added to F if it is not already there. Let M'' be M' with all ϵ arrows deleted.

Claim: M'' is equivalent to M, that is, $L(M'') = L(M)$.

Proof of claim: Clearly M' is equivalent to M. We know $L(M) \subseteq L(M')$ because M' has all transitions that M has and possibly more. However, $L(M') \subseteq L(M)$ because every accepting computation of M' can be simulated by an accepting computation of M.

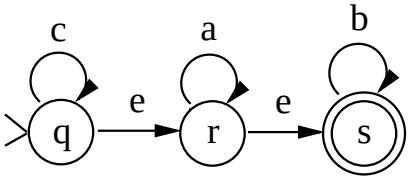
Finally, $L(M') = L(M'')$. Clearly $L(M'') \subseteq L(M')$ because M' has more arrows. But $L(M') \subseteq L(M'')$ because any sequence of moves of M' with ϵ arrows can be simulated by a sequence of moves of M''. Thus $L(M') = L(M'')$.

- For, consider a string w that is accepted by M'. Look at the sequence of states that the automaton goes through in accepting w, and in particular, at the subsequences consisting entirely of ϵ transitions.
- If $s_i, s_{i+1}, \ldots, s_{i+k}$ is a sequence of states with ϵ transitions in between them in this computation sequence, then, if s_{i+k} is an accepting state, the rules will make s_i an accepting state too, so the ϵ transitions are not needed in this computation sequence.
- If there is some arrow from s_{i+k} to s_{i+k+1} labeled with a symbol a that is used in the accepting computation, then the rules will add an arrow from s_i to s_{i+k+1} labeled with a, so the ϵ transitions are not needed in this computation sequence.
- So w is still accepted by M''.

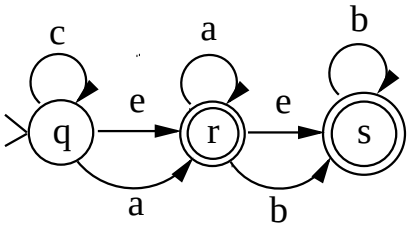
Putting these results together, $L(M'') = L(M)$.

1.8.1 Example

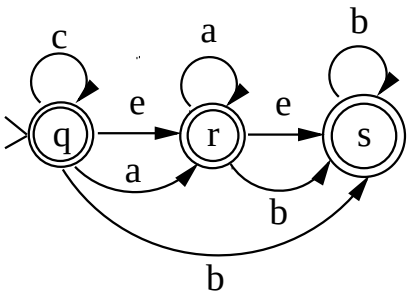
Consider this automaton with ϵ arrows:



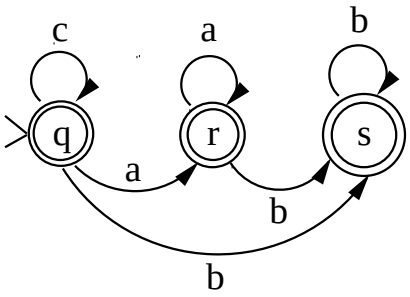
Now, to eliminate the ϵ arrows, first we apply the rules to add a b arrow from state r to state s, and make state r an accepting state. We also add an a arrow from state q to state r:



Now, we can apply the rules again. We add a b arrow from q to s. We also make state q an accepting state.

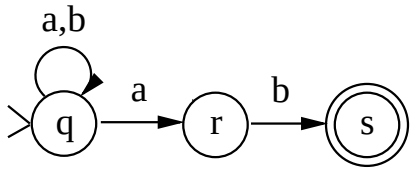


Now no more rules can be applied. So we delete the ϵ arrows and obtain the following automaton, having no ϵ arrows but equivalent to the starting automaton.

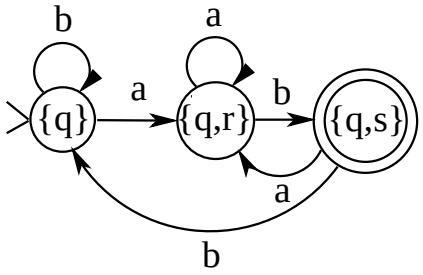


1.8.2 Continuing with the proof

Now we will show that every nondeterministic automaton can be converted to an equivalent deterministic automaton. For example, the following nondeterministic automaton M_N:



can be converted to this deterministic automaton M_D:

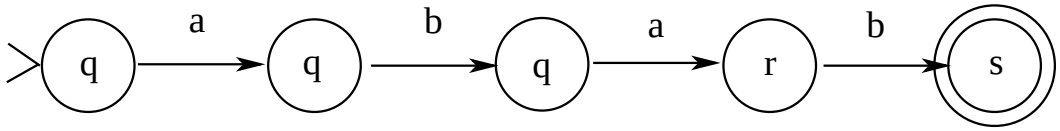


How M_D is obtained from M_N:

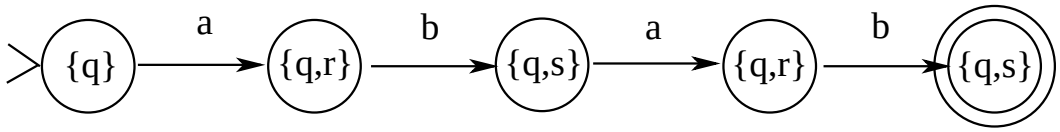
- The states of M_D are sets of states of M_N; only those that are reachable from the start state need to be kept.
- The input alphabet of M_D is the same as that of M_N.
- The start state of M_D is the set consisting of the start state of M_N.
- The accepting states of M_D are the sets containing at least one accepting state of M_N.
- If $\{q_1, q_2, \ldots, q_n\}$ is a state of M_D and a is in Σ, then there is an a arrow in M_D from $\{q_1, q_2, \ldots, q_n\}$ to the set of r such that in M_N there is an arrow from some q_i to r.

In class, give the algorithm to obtain M_D from M_N more explicitly. Show the set of eight subsets as states. Show the start state and the set of accepting states. Then illustrate on a few states how to do the arrows. Write the elements of the state and for each one which states it can go to with a given symbol in M_N. Then take the union of the results and draw the arrow in M_D. Finally, delete the unreachable states.

The equivalence of M_N and M_D can be shown by converting every accepting computation sequence for M_N into an accepting computation sequence for M_D, and vice versa. For example, the following computation for M_N:



corresponds to the following computation for M_D:



and vice versa.

Go forwards (left to right) in the computation constructing the bottom sequence from the top one, to map the top sequence into the bottom sequence, showing that every top state is an element of the corresponding bottom state.

Thus the final state s of the top sequence, which is an accepting state of M_N, is an element of the final state $\{q, s\}$ of the bottom sequence, which makes $\{q, s\}$ an accepting state of M_D. Thus the bottom sequence is also an accepting sequence.

Go backwards (right to left) constructing the top sequence from the bottom one, to map the bottom sequence into the top sequence, again showing that the top state is an element of the bottom state.

- Start with an accepting state s of M_N that is an element of the final state $\{q, s\}$ of the bottom sequence.

- Then go backwards to some state r of M_N that caused s to be added to the final state of the bottom sequence. Put this state in the top sequence.
- Continue going back this way, constructing the top sequence from the bottom one, and showing that each top state is an element of the bottom state.
- Finally, the first state q of the top sequence is an element of the first state $\{q\}$ of the bottom sequence, so the first state of the top sequence has to be the start state.

Thus if M_N accepts w then M_D accepts w and vice versa. This means that $L(M_N) = L(M_D)$ so M_N and M_D are equivalent.

This same construction can be used for any nondeterministic finite automaton, which shows that

> for every nondeterministic automaton with n states and no ϵ arrows, there is a determistic automaton with at most 2^n states.

This is stated formally in the following theorem.

Theorem 1.2 *Every n state nondeterministic finite automaton without ϵ arrows can be converted to an equivalent deterministic finite automaton with at most 2^n states.*

Proof: Suppose M_N is $(K, \Sigma, \Delta, s, F)$. Let M_D be $(K_D, \Sigma, \delta, \{s\}, F_D)$ where

$$K_D = 2^K$$

$$F_D = \{Q \subseteq K : Q \cap F \neq \emptyset\}$$

$$\delta(Q, a) = \{q' \in K : (q, a, q') \in \Delta \text{ for some } q \in Q\}.$$

Then accepting computations of M_N and M_D can be mapped onto each other as in the example.

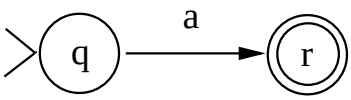
For the example M_N given above, because M_N has three states, M_D would have eight states. However, only three of these states are reachable from the start state, so only three of them are shown in the example M_D given above.

Because ϵ arrows can be eliminated from a nondeterministic automaton without increasing the number of states, we obtain the following result:

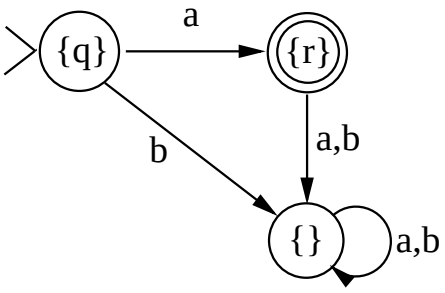
Theorem 1.3 *Every n state nondeterministic finite automaton (even if it has ϵ arrows) can be converted to an equivalent deterministic finite automaton with at most 2^n states.*

In fact, there is an algorithm to convert nondeterministic finite automata to equivalent deterministic finite automata.

Let's apply this construction to the automaton given above:



Suppose the alphabet is $\{a, b\}$. The final automaton is the following:



The state with the empty set in it arises because some inputs do not lead anywhere in the nondeterministic automaton.