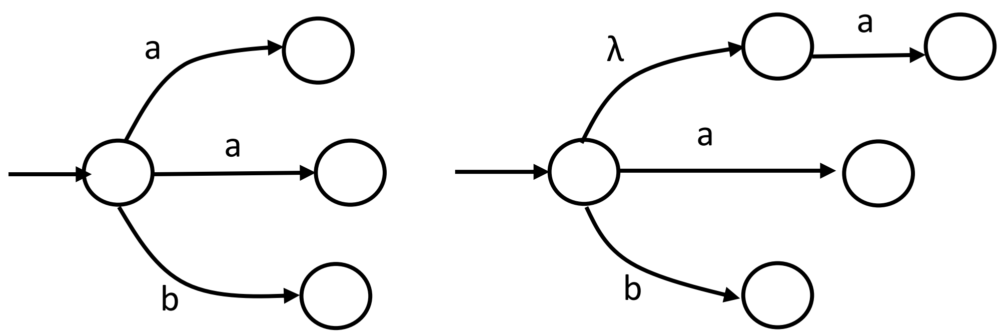
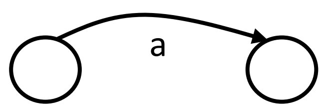
Non-Deterministic Finite Automata

Non-Deterministic Finite Automata (NFA)

A non-deterministic finite automaton allows choices. This means there may be multiple arcs with the same label from a given state. It also allows λ-label on an arc (i.e., a spontaneous transition), as in the following:



It also relaxes the requirement of completeness. For example, if A = { a, b}, the following is acceptable in an NFA:



no transition on b

Since we have choices the next state is not necessarily uniquely determined for a given input! This means that the transition function is now described as:

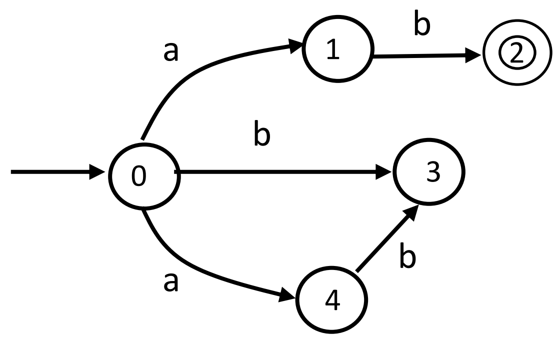
$$\delta: \quad Q \times (\Sigma \cup \{ \lambda \}) \rightarrow 2^Q \quad \text{for example } \delta(q, a) = \{q', q''\}$$

Remember that $2^Q \equiv \wp(Q)$ is the powerset of Q (i.e. the set of all subsets of Q).

Implication of the Changes

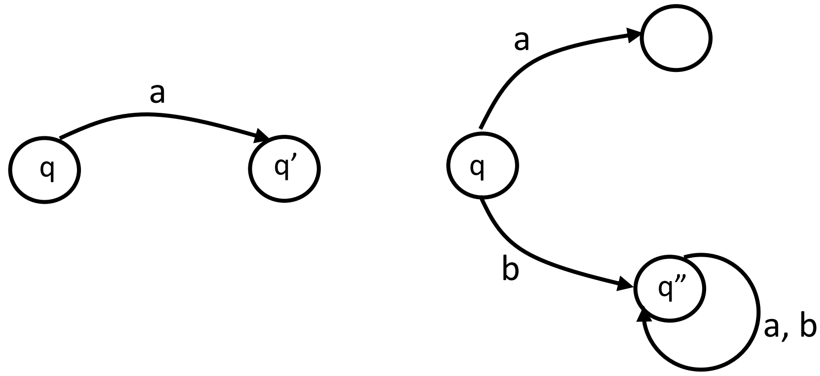
1. Allowing choices means that we need to redefine "acceptance":

   An NFA accepts string w if **some** choice of path leads to an accepting state, **even if many other path choices exist which do not end in an accepting state**.



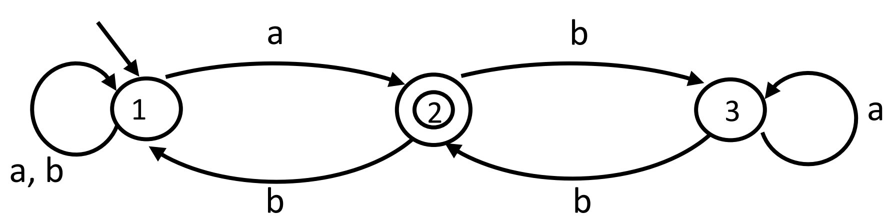
   This machine accepts **ab** since there is a path (0, 1, 2) leading to accept state 2, although there is another path (0, 4, 3) that does not. It does not accept **aa**

2. Incompleteness : Missing transitions are equivalent to a transition to a sink (q").



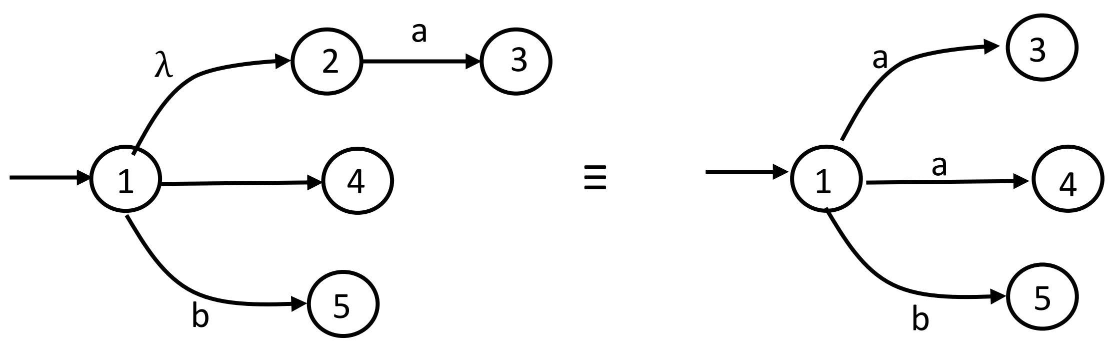
   Example:

   The following automaton

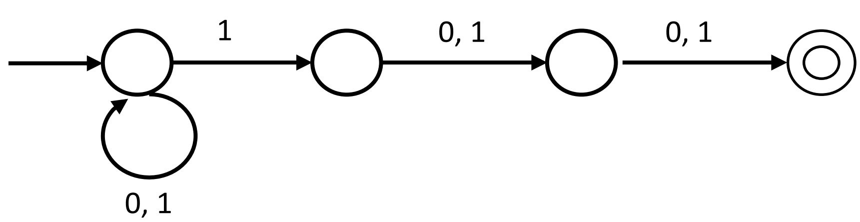


accepts the strings aabaa & abab (at least one path to 2)
rejects the string bbaab (no path to 2)
Note: NFA's are not very useful in a practical way since, in real-life problems, we don't want to be non-deterministic but they are an essential step in proving equivalence between RE and DFA. They are also often easier ti construct and, given the existence of an algorithm to convert a NFA to a DFA, it still allow the construction of an equivalent DFA.

3. Allowing λ labels means that we make a transition without consuming an input character.



Example:
Automaton which recognizes the language of all strings on {0, 1}which contain a 1 in the third position from the end:

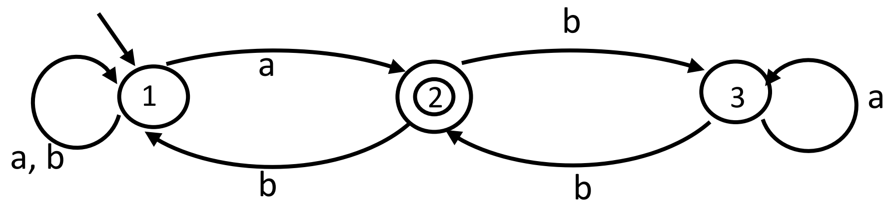


In other words, one has only to make the "right" transition at the "right" time.

Note: deterministic automata are a subset of NFA. However, when asked to provide an NFA, a DFA is not an acceptable answer!

Your turn:

1. Given the following NFA:



are the following strings accepted, indicate how you reach your conclusion?

a. aaaa

b. babbb

c. babab

2. Construct NFA's for the following (try for as few states as you can):

a. L = { 0 } with two states. Alphabet is {0, 1}.

b. L = { w ∈ {0, 1}* | w ends with 00} with three states.

c. L = { λ } with one state.

d. strings that finish with ab with no more than 3 states.

e. strings with at least one occurrence of ab with no more than 3 states.

f. strings that start with ab, or end with ab, or both with no more than 6 states.

How to Think of an NFA?

Think of it as a black box:

Input a string (once only), one character at a time. the result is accept or not accept (this is no different from a DFA). Note that machines accept strings but don't reject string (they may wander forever without giving an answer).

But, how does it work inside??

1. Parallel Computation or cloning, each following a different path through the states.

2. Guessing the right choice of the next state since *no amount of guessing can lead to accepting an unacceptable string*.

3. Deterministic Simulation:
   a. sequential trial-and-error
   b. systematic trial-and-error: backtracking
   c. systematic "parallel": breadth-first traversal

We will see below that NFA's are not any more powerful than DFA's (i.e. for every NFA there is a corresponding DFA) but it is sometimes easier to construct an NFA first. Hence there is no language recognized by an NFA which is not also recognized by some DFA.

Relation between NFA and DFA

DFA are a subset of NFA

NFA allow us to have a choice at some states but if we do not elect to have such choice we still have an NFA. Hence, a DFA is also an NFA (the reverse, of course is not true). Hence the following is a legitimate question is:

Are NFA's more "powerful" than DFA?

As already stated the answer is no and it is demonstrated by the following theorem.

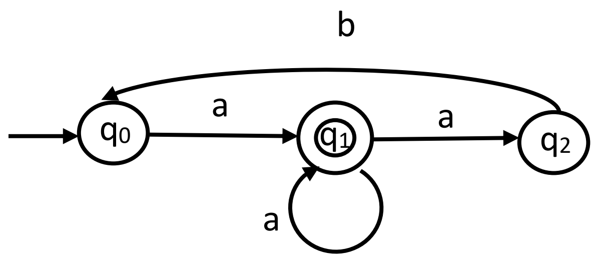
Theorem: For every NFA one can build an equivalent DFA (I.e. one which accepts the same language).

We will use a constructive proof.

The general idea is as follows: let us call the NFA M and its states g_1, g_2, etc., and the corresponding DFA D and its states q, q', q", etc.

- All states of D will be subsets of states of M (i.e we might have q' = { g_5, g_{10}}.)
- For every state q = {g_1, g_2, etc.} in D we keep track of all possible states one can get to. Let us assume q = {g_1, g_2} and that from g_1 upon receiving a particular symbol, let us say "a" we can get to g_4 or g_8. In a general sense, these two states are "equivalent" transition from g_1 upon receiving an a. Similarly, if from g_2 upon receiving an a you can go to g_6, g_9, or g_{20} those states are "equivalent." The union of those two subsets includes all the states that one can reach from q upon receiving an a. Let us call q' this union (i.e., the set { g_4, g_8, g_6, g_9, g_{20}}), so q' is a state of D. The transition in D from q upon receiving an a will be q'. In other words, $\delta(q, a) = q'$.

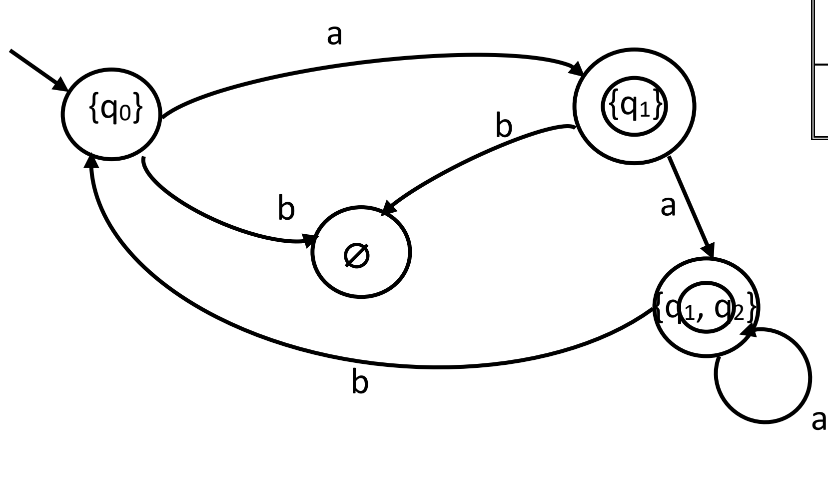
Let us first look at an example: consider the following NFA



	a	b
q_0	q_1	Ø
q_1	q_1, q_2	Ø
q_2	Ø	q_0

Applying the above method, we get the following transitions for the corresponding DFA:

	a	b
{q_0}	{q_1}	Ø
{q_1}	{q_1, q_2}	Ø
{q_1, q_2}	{q_1, q_2}	{q_0}
Ø	Ø	Ø



General algorithm to convert an NFA into a DFA

Assume an NFA $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ which has no λ transition (there is an algorithm to remove λ transitions). We build an equivalent DFA $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ as follows:

1. ***$\{q_0\}$ is the initial state of M_D***. This is always the case and should always be the starting point.

2. Repeat the following steps until no more edge are missing:

   a. Take any state $\{q_i, q_j, ..., q_k\}$ of M_D that has no transition for some symbol, say "a" .

   b. Compute $\delta_N (q_i, a), \delta_N (q_j, a), ..., \delta_N (q_k, a)$.

   c. Form the union of all the results (remember they are sets), yielding a set such that $\{q_l, q_m, ..., q_n\}$. If this set does not already exist, add it to Q_D.

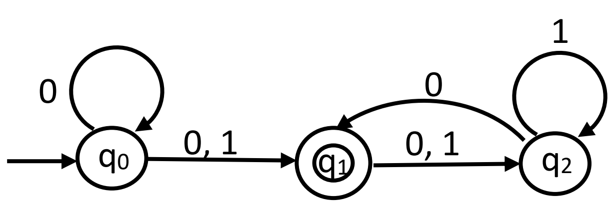
   d. Add to δ_D a transition from $\{q_i, q_j, ..., q_k\}$ to $\{q_l, q_m, ..., q_n\}$ under "a".

3. Every state of D that includes any $q_f \in F_N$ is identified as an accept state.

4. If M_N accepts λ, the state $\{q_0\}$ in D is also made a final state.

Your turn:

Convert the following NFA to a DFA



Transition table for NFA

	0	1
q_0		
q_1		
q_2		

The transition table for the DFA is as follows (remember where you should start):

	0	1

Leading to the following automaton: