****

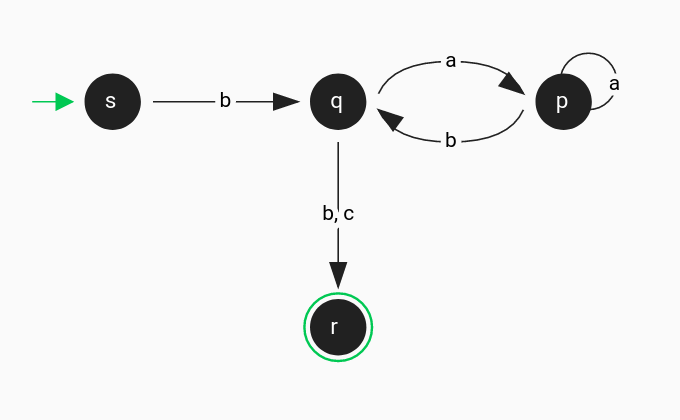
Department of Computing and Informatics

**COMP 221: AUTOMATA THEORY**

**Deterministic Finite Automatons (DFAs)**

The first type of finite-state machine is the **Deterministic Finite Automaton (DFA).** The word deterministic has quite a value here. It means that you can transition from one state to the next while not being in multiple states concurrently. Think the example with the traffic signals: they can’t be in green and red state at the same time.

Let's take a look at the automaton below:



Deterministic automaton with four states and five transitions.

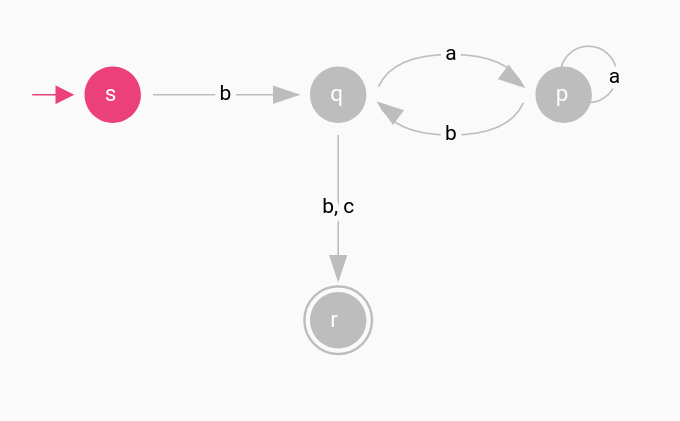
The above DFA contains 4 states. Starting state is s and accepting state is r. The alphabet contains the symbols a, b, and c.

Observe that for each state, there is only one transition arrow for each symbol. In some cases, a transition arrow can contain multiple symbols, like the one from q to r, but this does not break the requirement as there is no other exiting transition from q with the same symbols (b and c).

In deterministic finite automatons, the transition from one state to another can happen only if the input matches the symbol(s) of the transition. For instance, we can move from s to q only if the current symbol of the input is b.

**A running example**

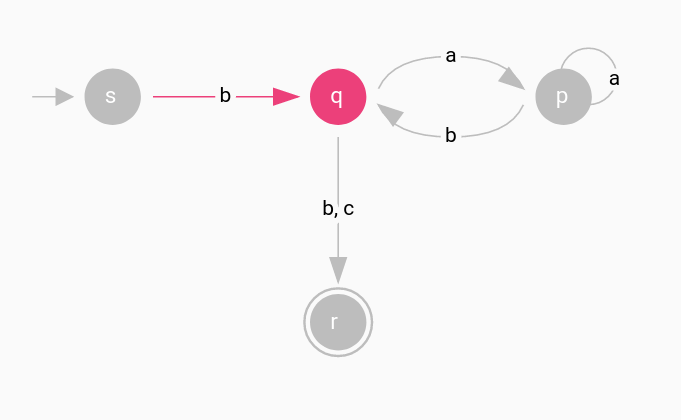
Let us see what the above DFA outputs for the input **baabb**.



Running a deterministic automaton.

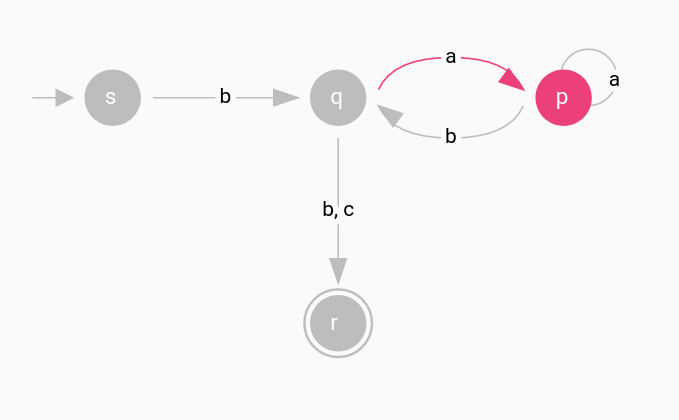
When the execution begins, the starting state is being activated. To run the automaton, for each symbol in the input, from left-to-right, we’re going to see if there is an exiting transition from the active state that match this symbol.

The first symbol of the input **baabb** is b. The current active state is s. Is there any exiting transition from s that contains the symbol b? Yes, the one that connects s to q.



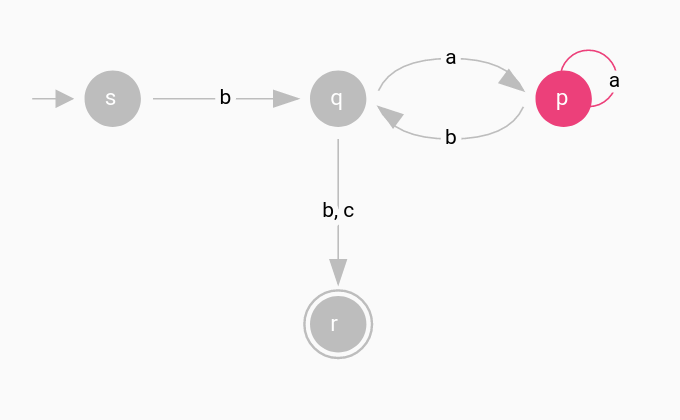
Running a deterministic automaton.

The current active state is q. The next symbol of the input **baabb** is a. Is there any exiting transition from q that match the symbol a? Yes, the one that connects q with p.



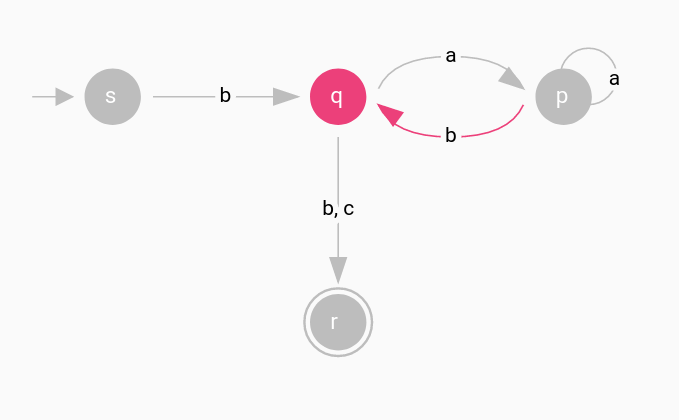
Running a deterministic automaton.

The current active state is p. The next symbol of the input **baabb** is a. Is there any exiting transition from p that match the symbol a? Yes, the one that connects p with itself.



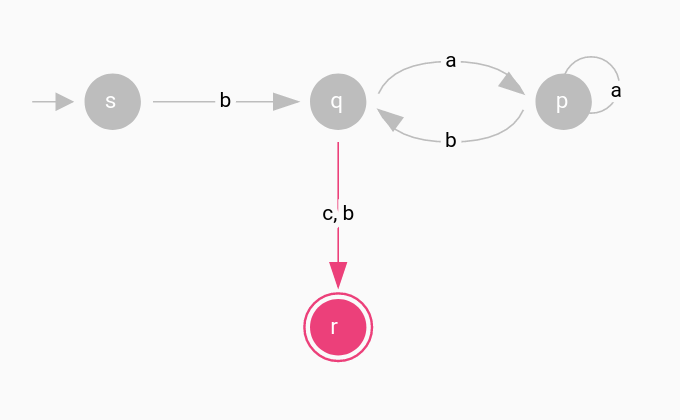
Running a deterministic automaton.

The current active state is p. The next symbol of the input **baabb** is b. Is there any exiting transition from p that match the symbol b? Yes, the one that connects p with q.



Running a deterministic automaton.

The current active state is q. The next symbol of the input **baabb** is b. Is there any exiting transition from q that match the symbol b? Yes, the one that connects q with r.



Running a deterministic automaton.

At this point we have traverse all the symbols of the input. The final active state is r. Is r an accepting state? Yes it is! That means the output is *accept*. In case that after traversing the input the final state was not an accepting state, the output would be *reject*.

**Formal definition**

The formal definition of a DFA.

A deterministic automaton is a 5-tuple ⟨**Q**, **Σ**, **δ**, **q₀**, **F**⟩, where

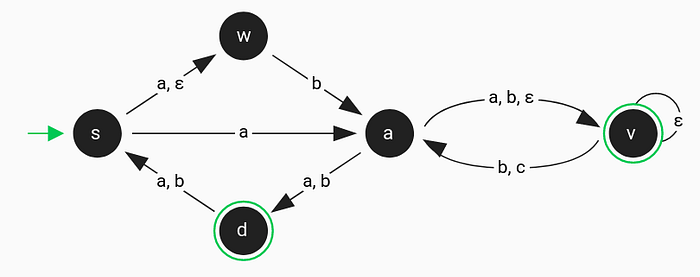
1. **Q** is a finite set of all states
2. **Σ** is a finite set of all symbols, the alphabet
3. **δ**: **Q** × **Σ** → **Q** is the transition function
4. **q₀** ∈ **Q** is the starting state
5. **F** ⊆ **Q** is the set of accepting states

Remember that we can move from one state to another only if the current input matches the symbol of the transition. For instance, consider that a finite automaton has a transition labeled 1 from a state p to a state q, then we can indicate the same thing with a transition function *δ(p, 1) = q*. This notion is just a kind of mathematical shorthand.

**Non-deterministic Finite Automatons (NFAs)**

The second type of finite-state machine is the **Non-deterministic Finite Automaton (NFA).** Compared to deterministic finite automatons, there are some major differences.

Consider the following automaton:



Non-deterministic automaton with five states and eight transitions.

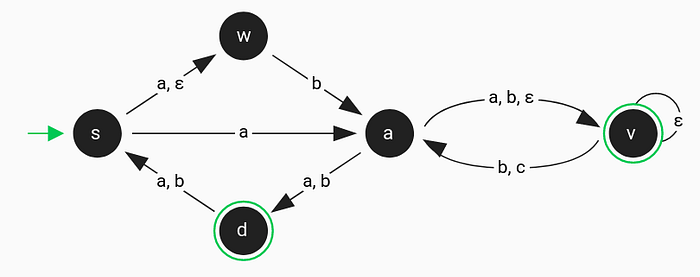
The above NFA contains 5 states. Starting state is s and accepting states are v and d. The alphabet contains the symbols a, b and c.

What is this strange label ε? It is a Greek character called **epsilon** and is not part of the alphabet. In non-deterministic automatons, transitions can be labeled either with a symbol from the alphabet or ε.

In non-deterministic automatons, there are cases where more than one transitions are applicable. In those cases, the automaton splits into multiple copies of itself and continues to run in non-determinism manner like before. We can see this behavior as a kind of parallel computation, where the machine is [forking](https://en.wikipedia.org/wiki/Fork%E2%80%93join_model) into several children, each processing independently.

Label ε is the first difference between deterministic and non-deterministic automatons. The second difference is the use of multiple transition arrows that contain the same symbols for each state in the NFA. For instance, state s has two exiting transitions labeled with the symbol a. This is the opposite of a DFA which has only one symbol per exiting transition for each state.

**A running example**



Non-deterministic automaton with five states and eight transitions.

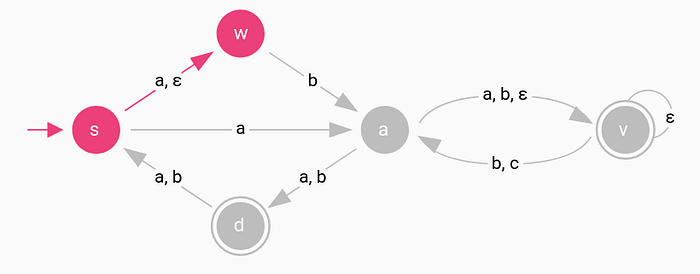
When the execution begins, the starting state is being activated. We’re going to check if there are any ε transition from s.

* Is there any ε transition from s? Yes, the one that connects s to w. Activate state w.

For the new state, we’ll do the same procedure until no more ε transitions exist.

* Is there any ε transition from w? No, there isn’t.

The initial active states are s and w.



Running a non-deterministic automaton.

For each symbol of the input, from left-to-right, we‘ll check if there are exiting transitions from the active states that match that symbol. Then we’ll follow the previous process and activate all the states that connect with *epsilon transitions*.

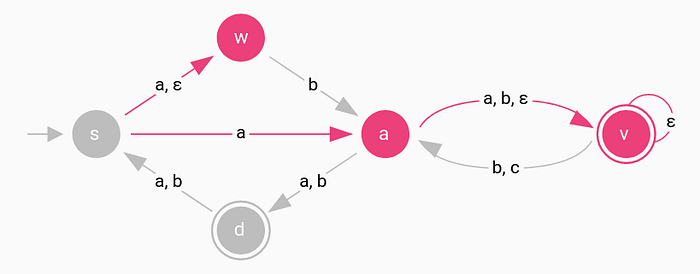
The first symbol of the input **aabc** is a. The current active states are s and w.

* Any exiting transition from s that contains the symbol a? Yes, the ones that connect s to w and a.
* Any exiting transition from w that contains the symbol a? No, there isn’t.

For the new activated states w and a, we’re going to follow all the *epsilon transitions* and repeat until we run out of them:

* Any ε transition from w? No, there isn’t.
* Any ε transition from a? Yes, the one that connects a to v.
* Any ε transition from v? Yes, the one that connects v to itself. Because this transition forms a self loop, we won’t test v again.

The new active states are w ,a and v.



Running a non-deterministic automaton.

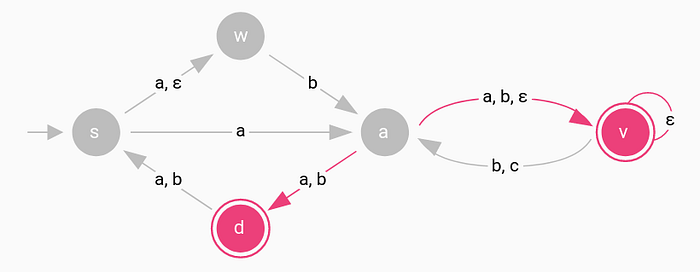
The next symbol of the input **aabc** is a. The current active states are w ,a and v.

* Any exiting transition from w that contains the symbol a? No.
* Any exiting transition from a that contains the symbol a? Yes, the ones that connect a to d and v.
* Any exiting transition from v that contains the symbol a? No.

For the new activated states d and v, we’re going to follow all the *epsilon transitions* and repeat until we run out of them:

* Any ε transition from d? No.
* Any ε transition from v? Yes, the one that connects v to itself. Because this transition forms a self loop, we won’t test v again.

The new active states are d and v.



Running a non-deterministic automaton.

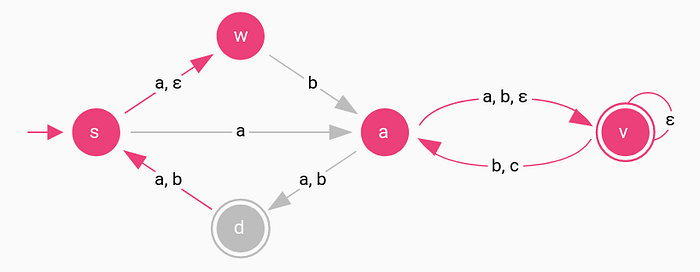
The next symbol of the input **aabc** is b. The current active states are d and v.

* Any exiting transition from d that contains the symbol b? Yes, the one that connect d to s.
* Any exiting transition from v that contains the symbol b? Yes, the one that connect v to a.

For the new activated states s and a, we’re going to follow all the *epsilon transitions* and repeat until we run out of them:

* Any ε transition from s? Yes, the one that connects to w .
* Any ε transition from w? No.
* Any ε transition from v? Yes, the ones that connects v to itself and a.
* Any ε transition from a? Yes, the one that connects to v . Because we already tested v we won’t test it again.

The new active states are s, w, a and v.



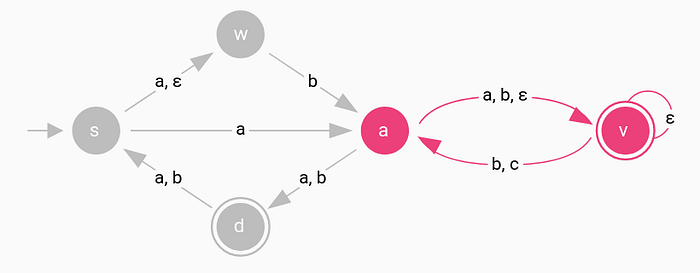
Running a non-deterministic automaton.

The last symbol of the input **aabc** is c. The current active states are s, w, a and v.

* Any exiting transition from s that contains the symbol c? No.
* Any exiting transition from w that contains the symbol c? No.
* Any exiting transition from a that contains the symbol c? No.
* Any exiting transition from v that contains the symbol c? Yes, the one that connects to a .

For the new activated state a, we’re going to follow all the *epsilon transitions* and repeat until we run out of them:

* Any ε transition from a? Yes, the one that connects to v .
* Any ε transition from v? Yes, the one that connects v to itself.



Running a non-deterministic automaton.

The final active states are a and v. Is any of them an accepting state? Yes, state v is an accepting state. That means the output is *accept*. In case that after traversing the input the final state was not an accepting state, the output would be *reject*.

**Formal definition**

The formal definition of a non-deterministic automaton is similar to the definition of a deterministic one. It consists of states, transitions, a transition function, an alphabet, and some accepting states. The only difference is the kind of transition function.

A non-deterministic automaton is a 5-tuple ⟨**Q**, **Σ**, **δ**, **q₀**, **F**⟩, where

1. **Q** is a finite set of all states
2. **Σ** is a finite set of all symbols, the alphabet
3. **δ**: **Q** × **Σε** → **P(Q)** is the transition function
4. **q₀** ∈ **Q** is the starting state
5. **F** ⊆ **Q** is the set of accepting states

In deterministic automatons, the transition function takes as input a state, a symbol and returns the next state, as output. In non-deterministic automatons, the transition function takes as input a state, a symbol or ε and returns the set of all possible next states, as output.