Non-deterministic Finite Automata
Theory of Computation (CSCI 3500)

Prof. Harley Eades (heades@gru.edu).

Read chapter 1.2.

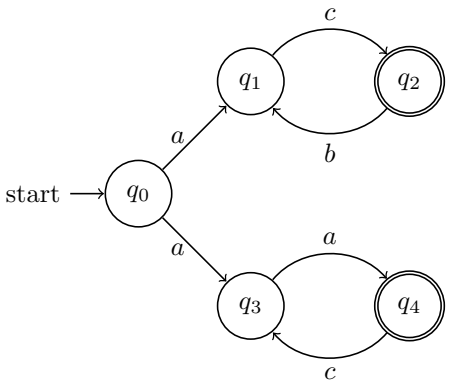
The most important part of DFAs is that they are deterministic, but how do we generalize this, and be able to define an automata that is non-deterministic. Recall from above that the one thing that makes a DFA deterministic is the fact that the transition function is a function. So one might think that if we relax this property and make the transition function, a relation instead, then non-determinism might hold as a result, but this is not what we want. We want to maintain that the transition function is a function, but still be able to have non-determinism.

A finite automata becomes non-deterministic when we allow a state to have multiple transitions with the same label to multiple other states. So how do we modify the definition of DFA to allow for this? The following is a preliminary definition of an NFA:

Definition 1. *A **non-deterministic finite automata (NFA)** is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$ where:*

Q *is the set of states of the NFA,*
Σ *is the alphabet,*
$\delta : Q \times \Sigma \to \mathcal{P}(Q)$ *is the transition function,*
q_0 *is the start state, and*
F *is the set of final states.*

This definition allows for examples such as the following:



In the above example the state q_0 has two transitions to different states both when reading an a. This is accounted for in the definition of the transition from $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. This new transition function takes in as input a pair of a state and an alphabet symbol, and then returns a set of states. For example, the

definition of δ in the previous example is the following:

$$\delta(q_0, a) = \{q_1, q_2\}$$
$$\delta(q_1, b) = \{q_2\}$$
$$\delta(q_2, c) = \{q_1\}$$
$$\delta(q_3, a) = \{q_4\}$$
$$\delta(q_4, c) = \{q_3\}$$

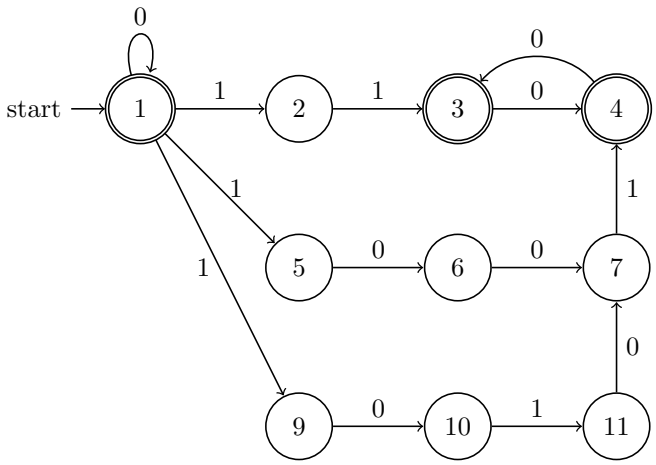
There is one additional form of non-determinism that should be accounted for in the above formal definition; which is the ability to change state without consuming a character. The non-determinism is that at any point we do not know if we should consume a character when moving to another state or not. To account for this we have to add an additional symbol to the set of labels on edges that stands for transitioning to another state without consuming the input. We denote this special symbol ϵ, and finally arrive at the complete definition of an NFA:

Definition 2. *A **non-deterministic finite automata (NFA)** is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$ where:*

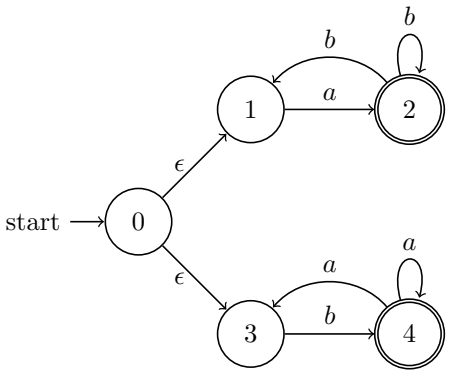
Q *is the set of states of the NFA,*
Σ *is the alphabet,*
$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ *is the transition function,*
q_0 *is the start state, and*
F *is the set of final states.*

1 Examples

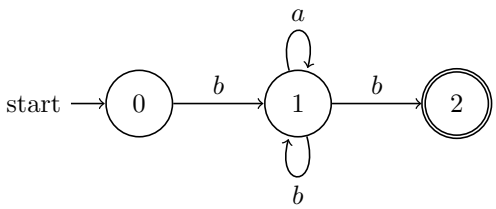
An NFA recognizing a select number of multiples of three in binary:



An NFA recognizing the language over the alphabet $\Sigma = \{a, b\}$ where a word is just a or every a is followed by at least one b, or is a b or every b is followed by at least one a:



The following NFA recognizes the language over alphabet $\Sigma = \{a, b\}$ where every word begins and ends with a b:



2 Running a NFA

First, notice that the following two runs are valid using the final NFA in the previous section:

$0bbb$	$0bbb$
$1bb$	$1bb$
$1b$	$2b$
1 (Reject)	2 (Accept)

So testing for acceptance when running a NFA is non-trivial. It is not enough to simply exhaust the input and test to see what state the NFA stops at. In this section we will define the run function formally for NFAs[1].

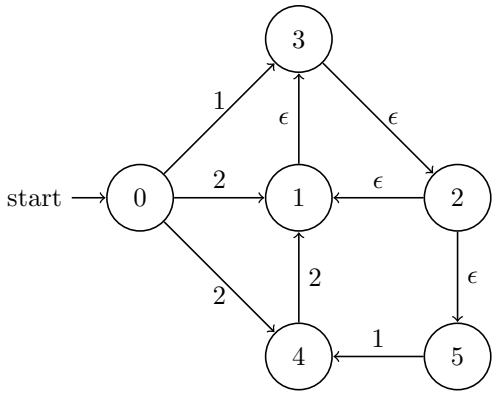
One hurdle that the run function must get over is how to handle the empty transition. That is, the transitions labeled with ϵ. To handle this we first need to a helper function that computes the set of all states reachable from a particular state via empty transitions.

Definition 3. *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA. Then we define the* ***ϵ-closure****, $\mathsf{eclose(X)}$, where $X \subseteq Q$ inductively as follows:*

If $q \in X$, then $q \in \mathsf{eclose}(X)$.
If $p \in \mathsf{eclose}(X)$ and $r \in \delta(p, \epsilon)$, then $r \in \mathsf{eclose}(X)$.

The set $\mathsf{eclose}(X)$ simply contains all the states reachable from any state in X via an empty transition. Now X is ϵ-closed if and only if $X = \mathsf{eclose}(X)$ which implies that if $q \in X$ and $\delta(q, \epsilon) = q'$, then $q' \in X$. Lets consider an example:

[1]This section is based on the notes of Thierry Coquand `http://www.cse.chalmers.se/~coquand/AUTOMATA/o4.pdf`.

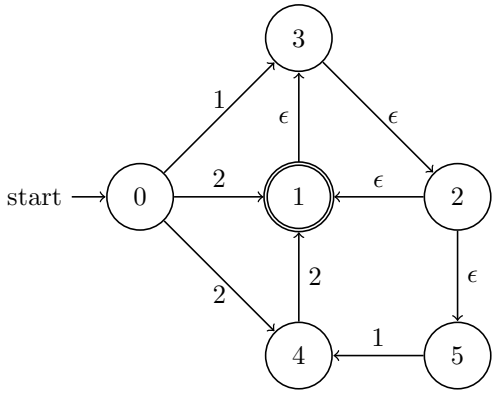


We can see that $\mathsf{eclose}(\{1\}) = \{1, 3, 2, 5\}$. Note that $\mathsf{eclose}(X)$ for any $X \subseteq Q$ is the smallest subset of Q that is ϵ-closed. Now the set $\mathsf{eclose}(X)$ will be used to keep track of all the reachable states do not consume input.

Definition 4. *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is an NFA. Then we define the run function, $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ as follows:*

$$\hat{\delta}(q, \epsilon) = \mathsf{eclose}(\{q\})$$
$$\hat{\delta}(q, aw) = \bigcup_{p \in \Delta(\mathsf{eclose}(q), a)} \hat{\delta}(p, w)$$
$$\text{where } \Delta(X, a) = \bigcup_{q \in X} \delta(q, a)$$

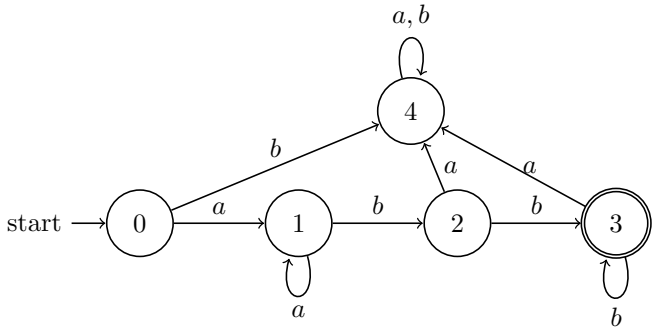
Lets consider an example:



Now lets use the run function to determine if the word 212 is accepted by the previous NFA:

$$
\begin{array}{rcl}
\hat{\delta}(0, 212) & = & \bigcup_{p \in \Delta(\mathsf{eclose}(0),2)} \hat{\delta}(p, 12) \\
& = & \bigcup_{p \in \Delta(\{0\},2)} \hat{\delta}(p, 12) \\
& = & \bigcup_{p \in \{1,4\}} \hat{\delta}(p, 12) \\
& = & \bigcup_{p \in \{1,4\}} (\bigcup_{p' \in \Delta(\mathsf{eclose}(p),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in \Delta(\mathsf{eclose}(1),1)} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in \Delta(\{1,2,3,5\},1)} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in \bigcup_{p \in \{1,2,3,5\}} \delta(p,1)} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in (\delta(1,1) \cup \delta(2,1) \cup \delta(3,1) \cup \delta(5,1))} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in (\{\} \cup \delta(2,1) \cup \delta(3,1) \cup \delta(5,1))} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in (\{\} \cup \{\} \cup \delta(3,1) \cup \delta(5,1))} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in (\{\} \cup \{\} \cup \{\} \cup \delta(5,1))} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in (\{\} \cup \{\} \cup \{\} \cup \{4\})} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\bigcup_{p' \in \{4\}} \hat{\delta}(p', 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup (\bigcup_{p' \in \Delta(\mathsf{eclose}(4),1)} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup (\bigcup_{p' \in \Delta(\{4\},1)} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup (\bigcup_{p' \in (\bigcup_{p'' \in \{4\}} \delta(p'',1))} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup (\bigcup_{p' \in \delta(4,1)} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup (\bigcup_{p' \in \{\}} \hat{\delta}(p', 2)) \\
& = & (\hat{\delta}(4, 2)) \cup \{\} \\
& = & \hat{\delta}(4, 2) \\
& = & \bigcup_{p \in \Delta(\mathsf{eclose}(4),2)} \hat{\delta}(p, \epsilon) \\
& = & \bigcup_{p \in \Delta(\{4\},2)} \hat{\delta}(p, \epsilon) \\
& = & \bigcup_{p \in (\bigcup_{p' \in \delta(4,1)})} \hat{\delta}(p, \epsilon) \\
& = & \bigcup_{p \in \{1\}} \hat{\delta}(p, \epsilon) \\
& = & \hat{\delta}(1, \epsilon) \\
& = & \mathsf{eclose}(\{1\}) \\
& = & \{1, 3, 2, 5\}
\end{array}
$$

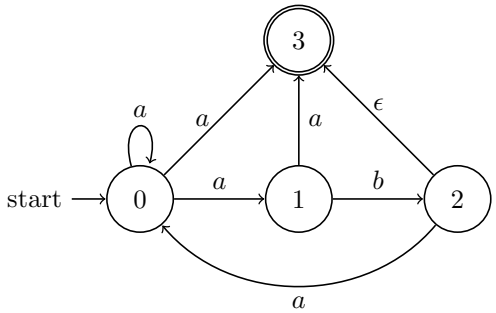
One thing that one should notice is that NFAs subsume DFAs. That is any DFA is automatically an NFA. So we should be able to define a DFA, and use the NFAs run function above to compute with it. Consider the following DFA:



Now apply the NFA run function for the previous DFA to the word *aabb*.

$$
\begin{aligned}
\hat{\delta}(0, aabb) &= \bigcup_{p \in \Delta(\mathsf{eclose}(0), a)} \hat{\delta}(p, abb) \\
&= \bigcup_{p \in \Delta(\{0\}, a)} \hat{\delta}(p, abb) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{0\}} \delta(p', a))} \hat{\delta}(p, abb) \\
&= \bigcup_{p \in \delta(0, a)} \hat{\delta}(p, abb) \\
&= \bigcup_{p \in \{1\}} \hat{\delta}(p, abb) \\
&= \hat{\delta}(1, abb) \\
&= \bigcup_{p \in \Delta(\mathsf{eclose}(1), a)} \hat{\delta}(p, bb) \\
&= \bigcup_{p \in \Delta(\{1\}, a)} \hat{\delta}(p, bb) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{1\}} \delta(p', a))} \hat{\delta}(p, bb) \\
&= \bigcup_{p \in \delta(1, a)} \hat{\delta}(p, bb) \\
&= \bigcup_{p \in \{1\}} \hat{\delta}(p, bb) \\
&= \hat{\delta}(1, bb) \\
&= \bigcup_{p \in \Delta(\mathsf{eclose}(1), b)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \Delta(\{1\}, b)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{1\}} \delta(p', b))} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \delta(1, b)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \{2\}} \hat{\delta}(p, b) \\
&= \hat{\delta}(2, b) \\
&= \bigcup_{p \in \Delta(\mathsf{eclose}(2), b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \Delta(\{2\}, b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{2\}} \delta(p', b))} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \delta(2, b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \{3\}} \hat{\delta}(p, \epsilon) \\
&= \hat{\delta}(3, \epsilon) \\
&= \mathsf{eclose}(\{3\}) \\
&= \{3\}
\end{aligned}
$$

All of our examples of running an NFA on a word that was accepted ended at the final state. However, this may not always happen. In fact, in a NFA we could end in a non-final state, but still accept the word. Here is an example:



The word ab is an accepted word, and so is $abaab$, but notice that the state 2 is not an accepting state. Well lets consider what happens when we run the NFA on the word ab:

$$
\begin{aligned}
\hat{\delta}(0, ab) &= \bigcup_{p \in \Delta(\mathsf{eclose}(0), a)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \Delta(\{0\}, a)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{0\}} \delta(p', a))} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \delta(0, a)} \hat{\delta}(p, b) \\
&= \bigcup_{p \in \{1\}} \hat{\delta}(p, b) \\
&= \hat{\delta}(1, b) \\
&= \bigcup_{p \in \Delta(\mathsf{eclose}(1), b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \Delta(\{1\}, b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in (\bigcup_{p' \in \{1\}} \delta(p', b))} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \delta(1, b)} \hat{\delta}(p, \epsilon) \\
&= \bigcup_{p \in \{2\}} \hat{\delta}(p, \epsilon) \\
&= \hat{\delta}(2, \epsilon) \\
&= \mathsf{eclose}(\{2\}) \\
&= \{2, 3\}
\end{aligned}
$$

We can characterize when a word is accepted by an NFA.

Definition 5. *A word, w, is accepted by an NFA, $M = (Q, \Sigma, \delta, q_0, F)$, if and only if $\hat{\delta}(q_0, w) = E$ and $E \cap F \neq \emptyset$.*

The previous definition states that a word is accepted by an NFA if and only if we can run the NFA on that word, exhaust all input, and obtain a set of states that contains at least one final state.

3 NFA-DFA Equivalence

A language is regular if and only if it is the language accepted by a DFA, but what do we call the language that is accepted by an NFA? Are NFAs more powerful than DFAs? The answer to the latter question might seem positive, because NFAs allow for non-determinism which seems more general than determinism. However, in the case of NFAs and DFAs they are actually equivalent! To prove this we will show the following lemma holds:

Lemma 6. *A language, L, is regular if and only if there exists a NFA, M, such that $L(M) = L$.*

To prove the previous lemma we will show that any NFA can be translated into an equivalent DFA, and vice versa. The vice-versa direction is easy, because any DFA is trivially a NFA. The fun part is the opposite direction.

Definition 7. *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA. Then we construct its equivalent DFA, $M_D = (Q_D, \Sigma_D, \delta_D, q_D, F_D)$, as follows:*

$$
\begin{aligned}
&Q_D = \mathcal{P}(Q) \\
&\Sigma = \Sigma_D \\
&\delta_D(R, a) = \{q \in Q \mid q \in \mathsf{eclose}(\delta(r, a)) \text{ for some } r \in R\} \\
&q_D = \mathsf{eclose}(\{q_0\}) \\
&F_D = \{R \in Q_D \mid R \cap F \neq \emptyset\}
\end{aligned}
$$

Clearly, the previous algorithm is correct, because every word accepted by M will be accepted by M_D, because we use the original transition function, and gather all of the final states reachable with respect to the input word.

We can now prove our original result:

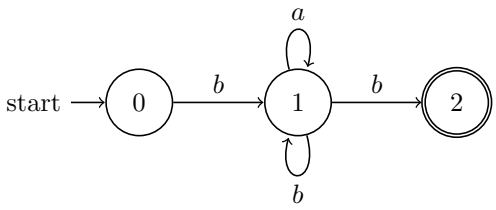
Lemma 8. *A language, L, is regular if and only if there exists a NFA, M, such that $L(M) = L$.*

Proof. ($\Rightarrow$) Suppose L is regular. Then it suffices to show that there exists an NFA, M, such that $L(M) = L$, but we know by the definition of regularity that there must be a DFA, M_D, such that $L(M_D) = L$. Furthermore, every DFA is trivially a NFA. Therefore, take $M = M_D$.

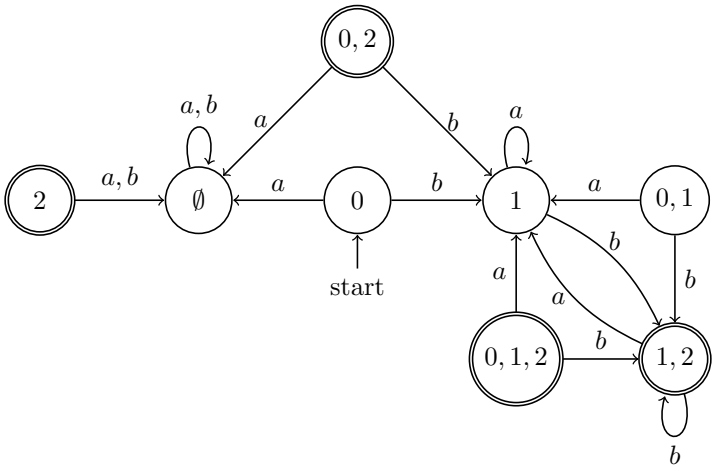
($\Leftarrow$) Suppose M is a NFA, then we must show that $L(M)$ is regular. The definition of regularity states that we must construct a DFA,M_D, that recognizes the language $L(M)$. Take M_D to be the DFA resulting from the NFA-to-DFA conversion algorithm from Def. 7. □

The following is an example conversion:

The NFA



is equivalent to the DFA



An exercise: convert the following to a DFA: