NFAs and $\varepsilon$-NFAs

Winter 2020

1 NFAs and $\varepsilon$-NFAs

**NFAs**: A nondeterministic finite automaton (NFA) is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$ where:

- $\Sigma$ is a non-empty finite set whose elements are called symbols (or sometimes called letters). The set $\Sigma$ is called the alphabet of the NFA.
- $Q$ is a non-empty finite set whose elements are called states.
- $q_0 \in Q$ is the initial state (sometimes called the starting state).
- $A \subseteq Q$ is the set of accepting states (sometimes called final states).
- $\delta: Q \times \Sigma \rightarrow 2^Q$ is the transition function, where $2^Q$ denotes the power set of $Q$ (the set of all subsets of $Q$).

The only difference between NFAs and DFAs is the definition of the transition function. For a DFA, the transition function is $\delta: Q \times \Sigma \rightarrow Q$, so the result of $\delta(q, a)$ for a state $q$ and a symbol $a$ is always a single state. For an NFA, the result of $\delta(q, a)$ can be a set of states.

What happens when we transition from a single state to a set of multiple states? There are two common interpretations of what this means in an intuitive sense.

- The NFA is now “in multiple states at once”. The NFA accepts a word if at least one of the states it is in after reading the whole word is an accepting state.
- The NFA is “nondeterministically guessing” which state it should go to next, and the set of states represents all the possible options for its guesses. The NFA accepts a word if there is some sequence of guesses that leads to an accepting state.

Use whichever interpretation you find most intuitive. Either way, it is important to remember that in the NFA recognition algorithm, we keep track of a *set of states* instead of a single state, and we apply the transition function to *every state in the set* whenever we consume a letter.

**$\varepsilon$-NFAs**: An $\varepsilon$-NFA allows for the use of $\varepsilon$-transitions, which are transitions that can be taken without consuming any input. These are particularly useful for constructing automata that consist of multiple smaller automata connected together.

The only difference in the definition is that the transition function is changed to allow for transitions on $\varepsilon$:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q.$$  

The $\varepsilon$-NFA recognition algorithm is also a bit different to account for the fact that it is possible to follow any number of $\varepsilon$-transitions in between consuming each symbol of the input. In between each symbol, we compute the $\varepsilon$-closure of the current state set, which is the set of all states we can reach from the current state set by following $\varepsilon$-transitions.
1.1 Exercises

1. Show that for every DFA, there is an NFA recognizing the same language.

2. Draw NFAs for the following languages over $\Sigma = \{0, 1\}$.
   
   (a) Let $A$ be the language of strings ending in 1011.
   
   (b) Let $B$ be the language of strings ending in either 10 or 01.
   
   (c) Let $C$ be the language of strings beginning with 1000.

3. Draw an $\varepsilon$-NFA for the language $C (A \cup B)$, where $A$, $B$ and $C$ are the languages from the previous exercise.

2 Converting an NFA to a DFA

Recall that the only difference between an NFA and a DFA is the transition function. The transition function for an NFA maps state-symbol pairs to sets of states:

$$\delta: Q \times \Sigma \to 2^Q.$$

Whereas the transition function for an DFA maps state-symbol pairs to single states:

$$\delta: Q \times \Sigma \to Q.$$

This leads to a simple trick to turn NFAs into DFAs. The rough idea is to replace the state set $Q$ of the NFA with $2^Q$. Now each “state” is a set, and the transition function maps “state-symbol pairs” (really set-symbol pairs) to “single states” (single sets). Now the transition function is deterministic, because for each state-symbol pair it outputs one state! This is known as the subset construction.

That’s the intuitive idea, but we have to be careful to make it formal. Suppose we have an NFA $N = (\Sigma, Q, q_0, A, \delta)$ and we want to turn it into a DFA $D = (\Sigma, Q', q'_0, A', \delta')$. Here is how each component of $D$ is defined:

- $\Sigma$ stays the same – we don’t change the alphabet when converting an NFA to a DFA.
- $Q'$ is $2^Q$, the set of all subsets of $Q$.
- $q'_0$ is $\{q_0\}$, the set containing the initial state of $Q$. Remember, the states of $D$ are sets, so we have to add the curly braces!
- $A'$ consists of all sets that contain a state from the NFA’s accepting state set $A$. Formally, we define

$$A' = \{S \in 2^Q : S \cap A \neq \emptyset\}.$$

- Finally, the transition function $\delta': 2^Q \times \Sigma \to 2^Q$, is defined as follows. Given a set $S \in 2^Q$ and a symbol $a \in \Sigma$, the resulting set $\delta'(S, a)$ is given by applying the NFA transition function $\delta$ to each state in $S$, and taking the union of the resulting sets. This is exactly what we do in the NFA recognition algorithm. Formally, we define:

$$\delta'(S, a) = \bigcup_{q \in S} \delta(q, a).$$

For example, if $S = \{1, 2, 3\}$, $\delta(1, a) = \{1, 4\}$, $\delta(2, a) = \emptyset$, and $\delta(3, a) = \{3, 5\}$, then we have

$$\delta'(\{1, 2, 3\}, a) = \delta(1, a) \cup \delta(2, a) \cup \delta(3, a) = \{1, 4\} \cup \emptyset \cup \{3, 5\} = \{1, 3, 4, 5\}.$$
In this worst case, the blow-up in states when converting from an NFA to a DFA is exponential. For example, if we have an NFA with 8 states, the equivalent DFA could have as many as $2^8 = 256$ states. One problem with the formal construction above is that it assumes we always need all the states in $2^Q$; sometimes we do, but this is just the worst-case scenario, and often we can make do with much fewer states.

For this reason, when converting NFAs to DFAs in practice, we use the following algorithm, which only finds the sets in $2^Q$ that are reachable from the initial set $\{q_0\}$. This algorithm is not guaranteed to produce a DFA with a minimal number of states, but it is better than assuming we need to include every possible set in $2^Q$.

This algorithm works by constructing a transition table. The rows of the table are states of the DFA, and the columns are letters of the alphabet.

1. Begin with one row, corresponding to the initial state $\{q_0\}$ of the DFA, and all columns empty.

2. For each empty row, let $S$ be the state corresponding to the row, and fill the row as follows. For each symbol $a \in \Sigma$, in column $a$ write the set $\delta'(S, a)$, where $\delta'(S, a)$ is as defined earlier.

   (What's going on here? Recall that $\delta'(S, a)$ consists of all states which, in the original NFA, are reachable from a state in $S$ via the symbol $a$. We are looking at the states we have so far in our DFA, and figuring out which new sets are reachable from these states.)

3. After all the rows are filled, check if there any sets in the table which do not have a corresponding row. For each such set, add a new empty row.

   (Why do we add new empty rows? Because each set which has no row corresponds to a newly reached state. We need to see what states are reachable from these newly reached states, so we add rows for them and repeat the last step with these new rows.)

4. If there are empty rows, return to step 2. If there are no empty rows, the algorithm terminates.

   (Why do we terminate if there are no empty rows? No empty rows means we did not reach any new states in the previous step.)

After constructing the table, you can use it to draw a DFA diagram. You could also skip the table and just draw a DFA directly, adding new states and transitions when the algorithm says to “fill in columns”. However, the result might be more messy than constructing a table first and then drawing the diagram.

### 2.1 Exercise

Convert the following NFA to a DFA using the subset construction.

![NFA Diagram]

Convert the following NFA to a DFA using the subset construction.
3 Converting an \( \varepsilon \)-NFA to a DFA

Converting a \( \varepsilon \)-NFA to a DFA is similar to converting an NFA without \( \varepsilon \)-transitions to a DFA. You just need to account for the \( \varepsilon \)-transitions when figuring out what states are reachable from a given set of states.

- Instead of always starting with initial state \( \{ q_0 \} \), the initial state should be:
  \[ \{ q_0 \} \cup \{ q : q \text{ is reachable from } q_0 \text{ by taking one or more } \varepsilon \text{-transitions} \} \]

- When filling out row \( S \) of the table, compute \( \delta'(S, a) \) for each symbol \( a \) as usual. But before adding this set to the table, figure out which states are reachable from states in \( \delta'(S, a) \) by one or more \( \varepsilon \)-transitions, and throw in these states as well.

In other words, before adding a row or filling out a column in a row, we always want to take the \( \varepsilon \)-closure.

3.1 Exercise

Convert the following \( \varepsilon \)-NFA to a DFA.

![Diagram of \( \varepsilon \)-NFA]

Appendix: Converting an \( \varepsilon \)-NFA to an NFA

Sometimes it is useful to be able to convert an \( \varepsilon \)-NFA to an NFA without \( \varepsilon \)-transitions, rather than going all the way to a DFA. We can use the following algorithm to convert the \( \varepsilon \)-NFA \( M = (Q, \Sigma, q_0, A, \delta) \) to the NFA \( N = (Q', \Sigma, q'_0, A', \delta') \). Since this algorithm was not taught in lectures, you are not required to know it for the exams.

1. The start state \( q'_0 \) of the NFA is the same as the start state \( q_0 \) of the \( \varepsilon \)-NFA.
2. We take the state set \( Q' \) of the NFA \( N \) to be the start state \( q_0 \), as well as any other state \( q \in Q \) which has at least one non-\( \varepsilon \) transition going into it. In other words, states that can only be reached via \( \varepsilon \) transitions get removed.
3. For each state in \( q \in Q' \) and each letter \( a \in \Sigma \), add a transition from \( q \) to all states which can be reached from \( q \) in the \( \varepsilon \)-NFA \( M \) by consuming exactly one \( a \) (possibly following multiple \( \varepsilon \)-transitions along the way).
4. Mark a state \( q \in Q' \) as accepting if it was accepting in the original \( \varepsilon \)-NFA, or if it can reach an accepting state in the original \( \varepsilon \)-NFA by following only \( \varepsilon \)-transitions.

Exercise

Convert the \( \varepsilon \)-NFA from the previous section to an NFA using the above algorithm.