# CSCI 341–Fall 2024: Lecture Notes
# Set 5: DFA-NFA Equivalence, Closures

Edward Talmage

September 20, 2024

## 1 Equivalence

**Claim 1.** *Let $N$ be an NFA, with $L = L(N)$. Then there exists a DFA $M$ s.t. $L(M) = L$.*

- That is, every NFA can be simulated by a DFA.
- Thus, the set of languages recognized by NFAs is the set of regular languages.
  - Technically, that claim requires a reverse proof as well, but that is easy: every DFA *is* an NFA.

**Exercise:** How might you try to prove this claim? What will our equivalent DFA look like?

**Intuition:** We need a state for every element of the power set of $N$'s states, to track the set of possible current states of the NFA. Thus, the DFA's state set will be the power set of the NFA's state set. Now, the DFA's current state encodes all of the states the NFA could be in after consuming this much of the input. We will have to be careful to include every state reachable on $\varepsilon$-transitions, as well.

*Proof.* Let $N = (Q, \Sigma, \delta, q_0, F)$. Define a DFA $M = (Q', \Sigma, \delta', q_0', F')$ with

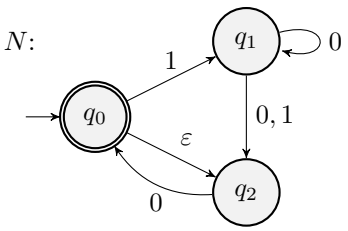
- $Q' = \mathcal{P}(Q)$
- $\delta' : Q' \times \Sigma \to Q', \delta'(S, a) = \cup_{r \in S} E(\delta(r, a)) = \hat{\delta}(S, a)$
- $q_0' = E(\{q_0\})$
- $F' = \{S \mid S \cap F \neq \emptyset\}$

By construction, there is a DFA equivalent to every NFA. □

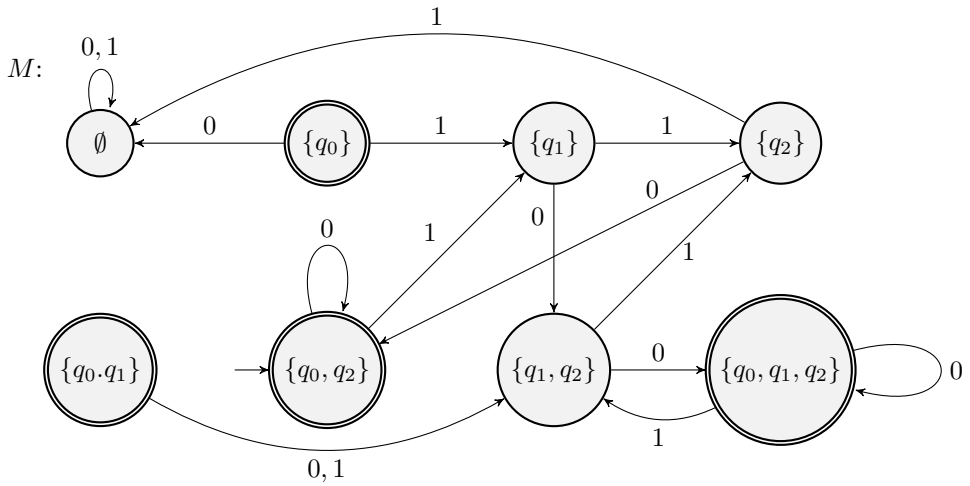
**Example:** Consider the following NFA, $N$.

(a) Draw an equivalent DFA by the above construction.

(b) Is there a more concise equivalent DFA?




We can construct an equivalent DFA as follows:




**Exercise:** Can you give a more concise DFA for this language?

- At a minimum, can delete the two unreachable states on the left.

**Exercise:** Use this construction to build a DFA equivalent to that from the last recitation which accepted all strings with a 1 in the third-last position.

## 2 Closure Properties

Now that we know that NFAs and DFAs are equivalent, we can prove closure properties of regular languages using NFAs, not just DFAs. In the following, let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be DFAs recognizing arbitrary regular languages $L_1$ and $L_2$, respectively.

**Exercise:** How could you (easily) build an NFA recognizing $L_1 \cup L_2$?

- Add a new start state with $\varepsilon$-transitions to the start states of the two machines.
- $Q = Q_1 \cup Q_2 \cup \{q_0\}$, where $q_0$ is a new state
- $F = F_1 \cup F_2$
- $q_0$ is start state
- $\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \\ \emptyset & \text{else} \end{cases}$

We can also complete the proof that regular languages are closed under concatenation and Kleene star.

*Proof.* 2. Given regular languages $L_1$ and $L_2$, we will show that $L_1 \circ L_2 = \{ab \mid a \in L_1, b \in L_2\}$ is regular.

**Exercise:** Give the intuition for how to construct an NFA that will recognize this concatenation.

To determine whether a string is in the concatenation of $L_1$ and $L_2$, we want to run $M_1$, then $M_2$, but we have to guess when to switch between them, since we don't know how long $a$ and $b$ are (or there could be multiple valid splits). We can let the non-determinism do this guessing by adding $\varepsilon$-transitions from every accept state of $L_1$ to the start state of $L_2$. Thus, anytime a string drives $M_1$ to an accept state, we will have processes which both continue running $M_1$ on the rest of the string and try to switch to running $M_2$ on the rest of the string.

- $Q = Q_1 \cup Q_2$

- $F = F_2$
- $q_0 = q_1$
- $\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \setminus F_1 \\ \delta_2(q,a) & q \in Q_2 \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1, a = \varepsilon \\ \delta_1(q,a) & q \in F_1, a \neq \varepsilon \\ \emptyset & else \end{cases}$

Since we have constructed an NFA recognizing $L_1 \circ L_2$, there is also an equivalent DFA recognizing the same language, so it is regular.

3. Given a regular language $L_1$, we now show that $L_1^*$ is also regular.

**Exercise:** How can we build an NFA to recognize $L_1^*$?

We need to run $M_1$, but every time we get to an accept state, we want to branch, guessing either that we have finished one string from $L_1$ and are starting another, or that we have not finished the current substring yet. We also need to be sure to accept $\varepsilon$, even if it is not in $L_1$.

- $Q = Q_1 \cup \{q_0\}$ for some $q_0 \notin Q$
- $q_0$ is the start state
- $F = F_1 \cup \{q_0\}$
- $\delta(q,a) = \begin{cases} \{q_1\} & q = q_0, a = \varepsilon \\ \delta_1(q,a) & q \in Q_1 \setminus F_1 \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1, a = \varepsilon \\ \delta_1(q,a) & q \in F_1, a \neq \varepsilon \\ \emptyset & else \end{cases}$

□

It is worth noting that when given a regular language, we now know that there is a DFA that recognizes that language, and that there is an NFA that recognizes it. We can use either in a construction. Most often, you will use DFAs for the base languages, since they are simpler and have fewer weird cases to remember. If we are constructing a machine to show that a language is regular, we can construct either a DFA or an NFA. Most often, we will build an NFA, since you can more freely specify the behavior you want (using non-determinism and $\varepsilon$-transitions).

**Exercise:** Given the two following NFAs recognizing $L_1$ and $L_2$ respectively, give machines for $L_1 \cup L_2$, $L_1 \circ L_2$, $L_1^*$, and $L_2^*$.