8 Non-Deterministic Finite-State Automata (NFAs)

8.1 Basic NFAs

Our DFAs are required to have transition functions that are total (so there is a next state for every current state and input symbol) and to return a single state (so there is a unique state for every current state and input symbol). Thus, the next state is fully determined by the current state and input symbol. As we saw in the previous section, this simplifies the proof that the DFA accepts a specific language. There are many circumstances, though, in which it w8ill be simpler to define the automaton in the first place if we allow for there to be any one of a number of next states or even no next state at all. We will still require transitions to be defined by a function, but it will now return a *set of states* rather than a single state.

Definition 33 (NFA without ε-Transitions) *A* Non-deterministic Finite-state Automaton (NFA) (without ε-transitions) *is a 5-tuple:* $\langle Q, \Sigma, \delta, q_0, F\rangle$ *where:*

Q, Σ, q_0 *and* F *are as in a DFA,*
$\delta : Q \times \Sigma \to \boldsymbol{\mathcal{P}}(Q)$ *is the* transition function
(mapping a state and an input symbol to the set of choices of next state)

Here $\boldsymbol{\mathcal{P}}(Q)$ denotes the powerset of Q—the set of all of its subsets. It should be emphasized that the transition function *is* still a function and is still total. We have accommodated the possibilities of there being either more than one or no potential next state by returning the set of next states: if $\mathbf{card}(\delta(q, \sigma)) = 1$, then the transition on $\langle q, \sigma\rangle$ is deterministic; if it is zero then there is no transition licensed for $\langle q, \sigma\rangle$ and the NFA will "crash"—halt prior to scanning the full input.

Instantaneous descriptions of NFAs are identical to those of DFAs.:

Definition 34 (Instantaneous Description (for both DFAs and NFAs)) *An* instantaneous description *of* $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$*, either a DFA or an NFA, is a pair* $\langle q, w\rangle \in Q \times \Sigma^*$*, where* q *the current state and* w *is the portion of the input under and to the right of the read head.*

The directly computes relation is nearly also the same—we need only to account for the fact the transition function returns a set of states rather than a unique next state.

Definition 35 (Directly Computes Relation (for NFAs without ε-transitions))

$$\langle q, w\rangle \vdash_{\mathcal{A}} \langle p, v\rangle \Leftrightarrow w = \sigma v \text{ and } p \in \delta(q, \sigma).$$

Note that while this is defined essentially identically for both DFAs and NFAs, in the case of NFAs it is no longer even partial functional; an ID may well have many successors. Moreover, it is no longer true that then only IDs without successors are those in which $w = \varepsilon$.

The definition of computation is, again, identical for both NFAs and DFAs. We do, however, need to amend our notion of *closure under* $\vdash_{\mathcal{A}}$ to account for the fact that, while there may now be more than one successor of an ID, only one of those can actually follow it in a single computation.

Definition 36 (Computation (for both DFA and NFAs)) *A computation of a DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ from state q_1 on input w_1 is a sequence of IDs $\langle \langle q_1, w_1\rangle, \ldots\rangle$ in which, for all $i > 0$, $\langle q_{i-1}, w_{i-1}\rangle \vdash_{\mathcal{A}} \langle q_i, w_i\rangle$ and which is closed under $\vdash_{\mathcal{A}}$.*

Here *closed under* $\vdash_{\mathcal{A}}$ means: for all i, if $\langle q_i, w_i\rangle$ has *any* successor, than *one* of those successors will be included in the sequence as $\langle q_{i+1}, w_{i+1}\rangle$. The fact that $\vdash_{\mathcal{A}}$ is no longer partial functional implies that we can no longer speak of *the* computation of $\mathcal{A}$ on $\langle q_1, w_1\rangle$. What's more, while every computation is finite, it is no longer true that they all take exactly $|w_1|$ steps. Computations end when they reach an ID with no successor; this can now be either because the entire input was scanned or because an ID $\langle q, \sigma \cdot w\rangle$ was reached for which $\delta(q, \sigma) = \emptyset$. Only the first case represents successful processing of w_1 by $\mathcal{A}$; we need to be careful to distinguish "halting" computations from those that "crash".

Nevertheless, except for weakening "the computation" to "a computation" there is no need to modify the definition of the computes relation:

Definition 37 (Computes Relation (for both DFAs and NFAs)) $\langle q, w\rangle \vdash^*_{\mathcal{A}} \langle p, v\rangle$ ("$\langle q, w\rangle$ computes $\langle p, v\rangle$ in $\mathcal{A}$") *iff $\langle p, v\rangle$ occurs in a computation of $\mathcal{A}$ on $\langle q, w\rangle$.*

$\langle q, w\rangle \vdash^n_{\mathcal{A}} \langle p, v\rangle$ ("$\langle q, w\rangle$ computes $\langle p, v\rangle$ in n steps in $\mathcal{A}$") *iff $\langle p, v\rangle$ is the $(n+1)^{\text{st}}$ element of a computation of $\mathcal{A}$ on $\langle q, w\rangle$. (I.e., iff $\langle p, v\rangle$ is an n^{th} successor of $\langle q, w\rangle$.)*

The definition of the language accepted by the automaton does not need be modified at all:

Definition 38 (Language Accepted by a DFA or NFA) *The* language accepted by a DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ *is*

$$L(\mathcal{A}) = \{w \mid \langle q_0, w \rangle \stackrel{*}{\underset{\mathcal{A}}{\vdash}} \langle q, \varepsilon \rangle, q \in F\}$$

The fact that we require the final ID of the computation accepting $w \in L(\mathcal{A})$ to be of the form $\langle q, \varepsilon \rangle$ means that computations that crash before scanning all of the input are ruled out.

Example: For example:

		Q			
	δ	0	1	2	3
Σ	a	$\{1,3\}$	$\{2,3\}$	$\{3\}$	$\{3\}$
	b	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\{0,2\}$

$F = \{0, 2\}$

The language accepted includes the string 'abb' as witnessed by the computation

$$\langle \langle 0, abb \rangle, \langle 1, bb \rangle, \langle 3, b \rangle, \langle 0, \varepsilon \rangle \rangle.$$

There are also other accepting computations on 'abb':

$$\langle \langle 0, abb \rangle, \langle 1, bb \rangle, \langle 1, b \rangle, \langle 0, \varepsilon \rangle \rangle$$
$$\langle \langle 0, abb \rangle, \langle 1, bb \rangle, \langle 1, b \rangle, \langle 2, \varepsilon \rangle \rangle$$

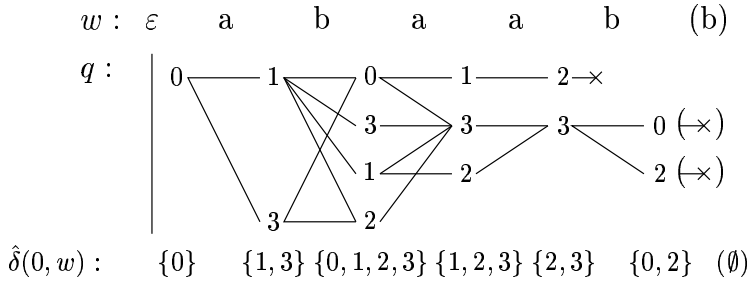
as well as computations that fail to accept 'abb':

$$\langle \langle 0, abb \rangle, \langle 1, bb \rangle, \langle 1, b \rangle, \langle 3, \varepsilon \rangle \rangle$$
$$\langle \langle 0, abb \rangle, \langle 1, bb \rangle, \langle 2, b \rangle \rangle$$

26. Give two distinct accepting computations of the NFA of the previous example on the string '$abab$'.

27. Give two distinct non-accepting computations of the NFA of the previous example on the string '$abab$'.

NFAs correspond to a kind of parallelism in the automata. We can think of the same basic model of automaton: an input tape, a single read head and an internal state, but when the transition function allows more than one next state for a given state and input we keep an independent internal state for each of the alternatives. In a sense we have a constantly growing and shrinking set of automata all processing the same input synchronously. For example, a computation of the NFA given above on '*abaab*' could be interpreted as:

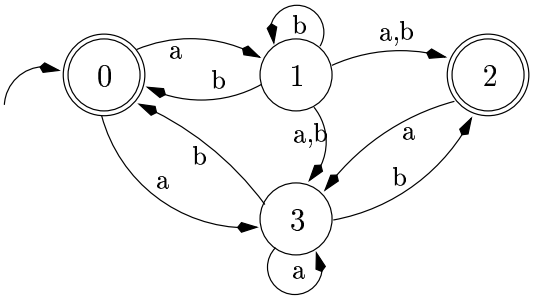


This string is accepted, since there is at least one computation from 0 to 0 or 2 on '*abaab*'. Similarly, each of 'ε', '*ab*', '*aba*' and '*abaa*' are accepted, but '*a*' alone is not. Note that if the input continues with '*b*' as shown there will be no states left; the automaton will crash. Clearly, it can accept no string starting with '*abaabb*' since the computations from 0 or '*abaabb*' end either in $\langle 0, b\rangle$ or in $\langle 2, b\rangle$ and, consequentially, so will all computations from 0 on any string extending it. The fact that in this model there is not necessarily a (non-crashing) computation from q_0 for each string complicates the proof of the language accepted by the automaton—we can no longer assume that if there is no (non-crashing) computation from q_0 to a final state on w then there must be a (non-crashing) computation from q_0 to a non-final state on w. As we shall see, however, we will never need to do such proofs for NFAs directly.

8.2 Transition Graphs of NFAs (without ε-transitions)

In terms of the transition graph of the automaton, if $\delta(q, \sigma) = \{q_1, q_2, \ldots, q_k\}$ then there will be an edge labeled σ from state q to each of the $q_1, q_2, \ldots, q_k$. This means that if there is a path labeled w leading from some q_0 to q, then there are paths labeled $w\sigma$ from q_0 to each of the $q_1, q_2, \ldots, q_k$. In the case that $\delta(q, \sigma) = \emptyset$ there is no edge labeled σ from state q in the transition graph and no path labeled $w\sigma$ that visits q in its next-to-last state.

Example: The transition graph of the NFA of the example of the last section is given in Figure 3.



		Q			
	δ	0	1	2	3
Σ	a	$\{1,3\}$	$\{2,3\}$	$\{3\}$	$\{3\}$
	b	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\{0,2\}$

$F = \{0,2\}$

Figure 3: An NFA and its transition graph.

Clearly, the path function also needs to return sets of states rather than states:

Definition 39 (Path Function of a NFA (preliminary version)) *The* path function $\hat{\delta} : Q \times \Sigma^* \to \boldsymbol{\mathcal{P}}(Q)$ *is the extension of* δ *to strings:*

- *Basis:* $\hat{\delta}(q, \varepsilon) = \{q\}$*, for all* $q \in Q$*.*
- *Induction: If* $q \in Q$*,* $w \in \Sigma^*$ *and* $\sigma \in \Sigma$ *then* $\hat{\delta}(q, w\sigma) = \bigcup_{q' \in \hat{\delta}(q,w)} [\delta(q', \sigma)]$*.*
- *Nothing else.*

This just says that the path labeled ε from any given state q goes only to q itself (or rather never leaves q) and that to find the set of states reached by paths labeled $w\sigma$ from q one first finds all the states q' reached by paths labeled w from q and then takes the union of all the states reached by an edge labeled σ from any of those q'. Another way of expressing it is

$$\hat{\delta}(q, w\sigma) = \{q'' \mid (\exists q' \in \hat{\delta}(q, w))[q'' \in \delta(q', \sigma)]\}.$$

We will still accept a string w iff there is a path labeled w leading from the initial state to a final state, but now there may be many paths labeled w

from the initial state, some of which reach final states and some of which do not. In the context of transition graphs, we need to modify the definition of the language accepted by $\mathcal{A}$ so it includes every string for which at least one path ends at a final state.

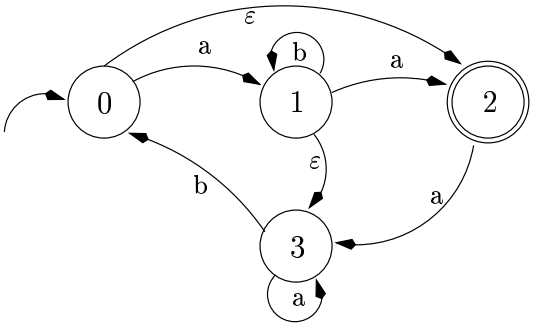
Definition 40 (Language Accepted by a NFA) *The language* accepted *by a NFA* $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ *is*

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

8.3 NFAs with ε-Transitions

We now add an additional degree of non-determinism and allow transitions that can be taken independent of the input—*ε-transitions.*

Example:



Here whenever the automaton is in state 1 it may make a transition to state 3 *without consuming any input.* Similarly, if it is in state 0 it may make such a transition to state 2. The advantage of such transitions is that they allow one to build NFAs in pieces, with each piece handling some portion of the language, and then splice the pieces together to form an automaton handling the entire language. To accommodate these transitions we need to modify the type of the transition function to map pairs drawn from Q and $\Sigma \cup \{\varepsilon\}$ to subsets of Q.

Definition 41 (NFA with ε-Transitions) *A* Non-deterministic Finite-state Automaton (NFA) (with ε-transitions) *is a 5-tuple:* $\langle Q, \Sigma, \delta, q_0, F \rangle$ *where* Q, Σ, q_0 *and* F *are as in a DFA and* δ *is of the type* $Q \times (\Sigma \cup \{\varepsilon\}) \to \boldsymbol{\mathcal{P}}(Q)$, *where* $\boldsymbol{\mathcal{P}}(Q)$ *denotes the* power set *of* Q, *i.e., the set of all subsets of* Q.

We must also modify the definitions of the directly computes relation and the path function to allow for the possibility that ε-transitions may occur anywhere in a computation or path. The ε-transition from state 1 to state 3 in the example, for instance, allows the automaton on input 'a' to go from state 0 not only to state 1 but also to immediately go to state 3. Similarly, it allows the automaton, when in state 1 with input 'b', to move first to state 3 and then take the 'b' edge to state 0 or, when in state 0 with input 'a', to move first to state 2 and then take the 'a' edge to state 3. Thus, on a given input 'σ', the automaton can take any sequence of ε-transitions followed by exactly one σ-transition and then any sequence of ε-transitions. To capture this in the definition of $\hat{\delta}$ we start by defining the function ε-Closure which, given a state, returns the set of all states reachable from it by any sequence of ε-transitions.

Definition 42 (ε-Closure of a State) *The* ε-Closure *of a state q of an automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ is defined inductively as follows:*

- *Basis:* $q \in \varepsilon\text{-Closure}(q)$.
- *Ind: If* $q' \in \varepsilon\text{-Closure}(q)$ *then* $\delta(q', \varepsilon) \subseteq \varepsilon\text{-Closure}(q)$
- *Nothing else.*

The ε-Closure *of a set of states $S \subseteq Q$ is*

$$\varepsilon\text{-Closure}(S) \stackrel{\text{def}}{=} \bigcup_{q \in S} [\varepsilon\text{-Closure}(q)].$$

With this we can modify the definitions of directly computes and the path function.

Definition 43 (Directly Computes Relation (for NFAs in general))

$$\langle q, w\rangle \vdash_{\mathcal{A}} \langle p, v\rangle \Leftrightarrow w = \sigma v \text{ and } p \in \varepsilon\text{-Closure}(\delta(\varepsilon\text{-Closure}(q), \sigma)).$$

Definition 44 (Path Function of a NFA (final version)) *The* path function $\hat{\delta} : Q \times \Sigma^* \to \boldsymbol{\mathcal{P}}(Q)$ *is the extension of δ to strings:*

- *Basis:* $\hat{\delta}(q, \varepsilon) = \varepsilon\text{-Closure}(q)$, *for all* $q \in Q$.
- *Ind: If* $q \in Q$, $w \in \Sigma^*$ *and* $\sigma \in \Sigma$
  then $\hat{\delta}(q, w\sigma) = \bigcup_{q' \in \hat{\delta}(q,w)} [\varepsilon\text{-Closure}(\delta(q', \sigma))]$.

- *Nothing else.*

In essence, these say that the set of states that may reach from ID $\langle q, \sigma \cdot w \rangle$ are those that can be reached by making any number of ε-transitions and exactly one σ transition, and that to find the set of states reachable by a path labeled w from a state q in an NFA with ε-transitions start by finding the set of states reachable from q using only ε-transitions and then, for each symbol σ of w (in order) find the set of states reachable from those by an edge labeled σ and then the set of states reachable from *those* by any sequence of ε-transitions, etc.

Nothing else in the definitions need change. The automaton still accepts w if there is any computation on $\langle q_0, w \rangle$ that terminates in a final state after scanning the entire input. Equivalently, it accepts w if there is a path labeled w from the initial state to a final state, which is to say, if $\hat{\delta}(q_0, w)$ includes any member of F. Note that the automaton of the example above will accept 'ε' since state 2 is in ε-Closure(0) and, therefore in $\hat{\delta}(0, \varepsilon)$.

8.4 Equivalence of NFAs with and without ε-transitions

It is not hard to see that ε-transitions do not add to the accepting power of the model. The effect of the ε-transition from state 1 to state 3 in the example, for instance, can be obtained by adding 'a' edges from 0 and 1 to 3 and a 'b' edge from 1 to 3. Similarly, most of the effect of the ε-transition from 0 to 2 can be obtained by adding an 'a' transition from 0 to 3 and 'b' transitions from 1 and 3 to 2. Note that in both these cases this corresponds to extending $\delta(q, \sigma)$ to include all states in $\hat{\delta}(q, \sigma)$. The remaining effect of the ε-transition from 0 to 2 is the fact that the automaton accepts 'ε'. This can be obtained, of course, by simply adding 0 to F. Formalizing this we get a lemma.

Lemma 8 (Equivalence of NFAs with and without ε-transitions) *A language $L \subseteq \Sigma^*$ is $L(\mathcal{A})$ for an NFA with ε-transitions $\mathcal{A}$ iff it is $L(\mathcal{A}')$ for an NFA without ε-transitions $\mathcal{A}'$.*

Proof: Note, first, that every NFA without ε-transitions is trivially an NFA with (no) ε-transitions as well. For the other direction we will show how, given an NFA with ε-transitions, to construct an NFA without ε-transitions

that accepts the same language.
For $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ let $\mathcal{A}' = \langle Q, \Sigma, \delta', q_0, F'\rangle$ where

$$\begin{aligned} \delta'(q,\sigma) &= \hat{\delta}(q,\sigma), \quad \text{for all } q \in Q \text{ and } \sigma \in \Sigma \\ \text{and} \quad F' &= \begin{cases} F \cup \{q_0\} & \text{if } \hat{\delta}(q_0,\varepsilon) \cap F \neq \emptyset, \\ F & \text{otherwise .} \end{cases} \end{aligned}$$

Note that δ' includes no ε-transitions and, therefore, $\mathcal{A}'$ is the appropriate type of NFA. To show that $L(\mathcal{A}') = L(\mathcal{A})$ we need to establish that

$$\hat{\delta}'(q_0, w) \cap F' \neq \emptyset \Leftrightarrow \hat{\delta}(q_0, w) \cap F \neq \emptyset.$$

This would, of course, follow if we could show that $\hat{\delta}'(q, w) = \hat{\delta}(q, w)$ for all q and w, and it is not hard to see that for non-empty w this will be the case. The problem, of course, is that it will not be the case for $w = \varepsilon$ since, by definition,

$$\hat{\delta}'(q, \varepsilon) = \{q\}$$

while

$$\hat{\delta}(q, \varepsilon) = \varepsilon\text{-Closure}(q).$$

But the 'ε' case is handled by the fact that q_0 has been added to F' if $\hat{\delta}(q_0, \varepsilon) \cap F \neq \emptyset$. Thus,

$$\begin{aligned} \varepsilon \in L(\mathcal{A}') &\Leftrightarrow \hat{\delta}'(q_0,\varepsilon) \cap F' \neq \emptyset \\ &\Leftrightarrow q_0 \in F' \\ &\Leftrightarrow q_0 \in F \text{ or } \hat{\delta}(q_0,\varepsilon) \cap F \neq \emptyset \\ &\Leftrightarrow \hat{\delta}(q_0,\varepsilon) \cap F \neq \emptyset \qquad \text{since } q_0 \in \hat{\delta}(q_0,\varepsilon) \\ &\Leftrightarrow \varepsilon \in L(\mathcal{A}). \end{aligned}$$

What remains, then, is to establish the $\hat{\delta}$ and $\hat{\delta}'$ coincide on all strings of length at least one. This is left to you as an exercise.

28. Prove the claim: for all $w \in \Sigma^+$ and $q \in Q$, $\hat{\delta}'(q, w) = \hat{\delta}(q, w)$.

⊣

8.5 Equivalence of NFAs and DFAs

In general non-determinism, by introducing a degree of parallelism, may increase the accepting power of a model of computation. But if we subject NFAs to the same sort of analysis as we have used in defining DFAs we shall see that to simulate an NFA one needs only track finitely much information about each string. Consider, again, the example in which we modeled the computation of the NFA as a set of automata processing the input synchronously. In order to determine if a string w is accepted by the NFA all we need to do is to track, at each stage of the computation (i.e., at each prefix of the input), the states of those automata. Since there is never any reason to include more than one automaton for each state, this will just be some subset of Q—in fact, it is easy to see that the set of states after processing w will be just $\hat{\delta}(q_0, w)$. Since Q is finite, it has finitely many subsets. Thus we can simulate an NFA with state set Q with a DFA that has a state for each subset of Q. The process of constructing a deterministic analog of a non-deterministic machine is known as *determinization.*

Lemma 9 (Equivalence of NFAs and DFAs) *A language $L \subseteq \Sigma^*$ is $L(\mathcal{A})$ for an NFA (with or without ε-transitions) $\mathcal{A}$ iff it is $L(\mathcal{A}')$ for some DFA $\mathcal{A}'$.*

Proof (Subset Construction)**:** Again, if $L = L(\mathcal{A}')$ for a DFA it is easy to see that it is also $L(\mathcal{A})$ for an NFA—one in which the transition function always returns a singleton set of states. For the other direction we, again, will give a construction, this time one that, given a NFA, builds a DFA accepting the same language. Because the state set of the DFA will be the set of all subsets of the state set of the NFA this construction is known as the *Subset (or Powerset) Construction.*
For $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ an NFA (for ease of proof we will assume that it does not have ε-transitions) let $\mathcal{A}' = \langle Q', \Sigma, Q'_0, \delta', F'\rangle$, where:

$$\begin{aligned} Q' &= \boldsymbol{\mathcal{P}}(Q) \\ Q'_0 &= \{q_0\} \\ \delta(S,\sigma) &= \bigcup_{q\in S}[\delta(q,\sigma)] \qquad \text{for each } \sigma \in \Sigma \text{ and } S \in Q' (\text{i.e., } S \subseteq Q) \\ F' &= \{S \subseteq Q \mid S \cap F \neq \emptyset\}. \end{aligned}$$

We must prove for all $w \in \Sigma^*$ that

$$\hat{\delta}'(Q_0', w) \cap F' \neq \emptyset \Leftrightarrow \hat{\delta}(q_0, w) \cap F \neq \emptyset.$$

We will do this by proving the slightly strengthened claim:

$$\text{for all } w \in \Sigma^*, \qquad \hat{\delta}'(Q_0', w) = \hat{\delta}(q_0, w),$$

by induction on $|w|$.

(Basis)

Suppose $|w| = 0$. Then $w = \varepsilon$ and

$$\hat{\delta}'(Q_0', \varepsilon) = \{q_0\} = \delta(q_0, \varepsilon).$$

(Induction)

Suppose $|w| = n + 1$ and that the claim is true for all strings of length n. Then $w = v\sigma$ for some $\sigma \in \Sigma$ and $v \in \Sigma^*$ of length n. To show that the claim is true for w as well:

$$\begin{aligned}
\hat{\delta}'(Q_0', v\sigma) &= \delta'(\hat{\delta}'(Q_0', v), \sigma) && \text{by definition of } \hat{\delta}' \\
&= \delta'(\hat{\delta}(q_0, v), \sigma) && \text{by IH} \\
&= \bigcup_{q' \in \hat{\delta}(q_0, v)} [\delta(q', \sigma)] && \text{by definition of } \delta' \\
&= \hat{\delta}(q_0, v\sigma) && \text{by definition of } \hat{\delta}.
\end{aligned}$$

Then

$$\begin{aligned}
w \in L(\mathcal{A}') &\Leftrightarrow \hat{\delta}'(Q_0', w) \in F' \\
&\Leftrightarrow \hat{\delta}'(Q_0', w) \cap F \neq \emptyset \\
&\Leftrightarrow \hat{\delta}(q_0, w) \cap F \neq \emptyset \\
&\Leftrightarrow w \in L(\mathcal{A}).
\end{aligned}$$

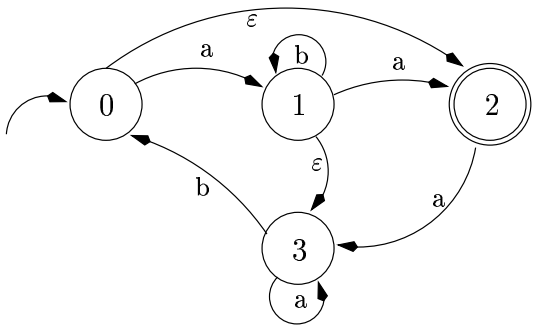
⊣

8.5.1 Applying Subset Construction

As defined the subset construction builds a DFA with many states that can never be reached from Q_0'. Since they cannot be reached from Q_0' there is

no path from Q'_0 to a state in F' which passes through them and they can be deleted from the automaton without changing the language it accepts. In practice it is much easier to build Q' as needed, only including those state sets that actually are needed.

To see how this works, lets carry out an example. For maximum generality, let's start with the NFA *with ε-transitions* given above, repeated here:



When given a transition graph of an NFA with ε-transitions like this there are 6 steps required to reduce it to a DFA:

1. Write out the transition function and set of final states of the NFA.
2. Convert it to an NFA without ε-transitions.
   (a) Compute the ε-Closure of each state in the NFA.
   (b) Compute the transition function of the equivalent NFA without ε-transitions.
   (c) Compute the set of final states of the equivalent NFA without ε-transitions.
3. Convert the equivalent NFA to a DFA.
   (a) Starting with $\{q_0\}$ and repeating for each state set encountered in the construction compute the transition function for each input symbol.
   (b) Compute the set of final states of the equivalent DFA.

While it is tempting to work directly from the transition graph and to combine steps there are two reasons to not do so. First, both shortcuts are prone to error. It is easy to miss edges when referring repeatedly to the graph

and it is, in particular, easy to miss relevant ε-transitions when trying to simultaneously remove them and determinize the result. The second reason is that we have only proved the correctness of the constructions eliminating ε-transitions and determinizing the result separately. While it is not difficult to define a construction combining them, if you use such a construction you must prove its correctness.

The transition function:

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\varepsilon)$
0	$\{1\}$	$\emptyset$	$\{2\}$
1	$\{2\}$	$\{1\}$	$\{3\}$
2	$\{3\}$	$\emptyset$	$\emptyset$
3	$\{3\}$	$\{0\}$	$\emptyset$

$F = \{2\}$.

ε-Closure:

q	ε-Closure(q)
0	$\{0,2\}$
1	$\{1,3\}$
2	$\{2\}$
3	$\{3\}$

The transition function of the equivalent NFA without ε-transitions:

$$\delta'(q,\sigma) \stackrel{\text{def}}{=} \hat{\delta}(q,\sigma) \stackrel{\text{def}}{=} \bigcup_{q' \in \varepsilon\text{-Closure}(q)} [\varepsilon\text{-Closure}(\delta(q'\sigma))].$$

q	ε-Closure(q)	$\delta'(q,a)$	$\delta'(q,b)$
0	$\{0,2\}$	$\{1,3\}$	$\emptyset$
1	$\{1,3\}$	$\{2,3\}$	$\{0,1,2,3\}$
2	$\{2\}$	$\{3\}$	$\emptyset$
3	$\{3\}$	$\{3\}$	$\{0,2\}$

$F' = \{0,2\}$ since ε-Closure$(0) = \{0,2\} \cap F \neq \emptyset$.

To construct the transition function of the equivalent DFA start with $S = Q'_0 = \{q_0\} = \{0\}$ and compute

$$\delta''(S,\sigma) = \bigcup_{q \in S} [\delta'(q,\sigma)].$$

for each $\sigma \in \Sigma$.

S	$\delta''(S,a)$	$\delta''(S,b)$
$\{0\}$	$\{1,3\}$	$\emptyset$

Next introduce lines for each of these state sets

S	$\delta''(S,a)$	$\delta''(S,b)$
$\{0\}$	$\{1,3\}$	$\emptyset$
$\{1,3\}$	$\{2,3\}$	$\{0,1,2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$

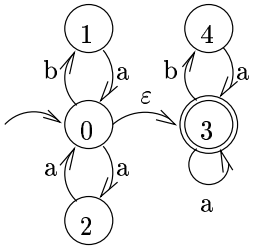
and repeat until no new lines are introduced

S	$\delta''(S,a)$	$\delta''(S,b)$
$\{0\}$	$\{1,3\}$	$\emptyset$
$\{1,3\}$	$\{2,3\}$	$\{0,1,2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{2,3\}$	$\{3\}$	$\{0,2\}$
$\{0,1,2,3\}$	$\{1,2,3\}$	$\{0,1,2,3\}$
$\{3\}$	$\{3\}$	$\{0,2\}$
$\{0,2\}$	$\{1,3\}$	$\emptyset$
$\{1,2,3\}$	$\{2,3\}$	$\{0,1,2,3\}$

The set of final state F'' is then the set of state sets that include states of F':

$$F'' = \{\{0\},\{2,3\},\{0,1,2,3\},\{0,2\},\{1,2,3\}\}.$$

29. Convert the following NFA to a DFA.



9 The Equivalence of DFAs and Regular Expressions

We have now looked at two ways of defining languages—as the denotation of a regular expression and as the set accepted by a DFA or, equivalently, an NFA. Surprisingly, the class of languages that can be defined in the one way turns out to be exactly the class of languages definable in the other. This theorem, originally due to Kleene, was one of the first dramatic theorems of Formal Language Theory. We will establish it in two parts: first we will show how to convert any regular expression to an equivalent NFA and then we will show how to convert any DFA into an equivalent regular expression. Since we have already shown the equivalence of NFAs and DFAs, this will suffice.

9.1 Constructing NFAs from Regular Expressions

Lemma 10 *If $L \subseteq \Sigma^*$ is $L(R)$ for a regular expression R then it is also $L(\mathcal{A})$ for an NFA.*

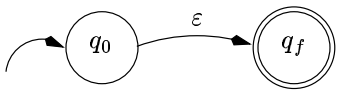
Proof: Not surprisingly, we will prove this by induction on the structure of the regular expression. To simplify the proof, we will strengthen the hypothesis slightly: IF $L = L(R)$ for a regular expression R then $L = L(\mathcal{A})$ for an NFA $\mathcal{A}$ which has a single final state.

(Basis:)

- Suppose $R = \emptyset$. Let $\mathcal{A}$ be

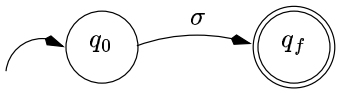
  Since there are no paths from the start state to the final state $L(\mathcal{A}) = \emptyset = L(\emptyset)$. Furthermore, $\mathcal{A}$ has a single final state.

- Suppose $R = \varepsilon$. Let $\mathcal{A}$ be



Here the empty string is in $L(\mathcal{A})$ since there is an ε-transition from the start state to the final state, but no other strings are in $L(\mathcal{A})$ since there are no other transitions out of either state. Again, $\mathcal{A}$ has a single final state.

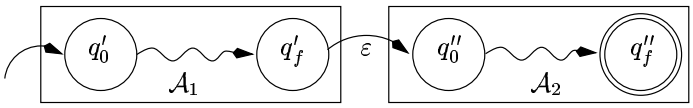
- Suppose $R = \sigma$ for some $\sigma \in \Sigma$. Let $\mathcal{A}$ be



Essentially the same argument applies in this case.

(Induction:)

- Suppose $R = S_1 \cdot S_2$ where S_1 and S_2 are regular expressions and that $L(S_1) = L(\mathcal{A}_1)$ and $(S_2) = L(\mathcal{A}_2)$ for NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ with single final states. Let $\mathcal{A}$ be



Formally, if $\mathcal{A}_1 = \langle Q_1, \Sigma, q'_0, \delta_1, \{q'_f\}\rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma, q''_0, \delta_2, \{q''_f\}\rangle$ we let

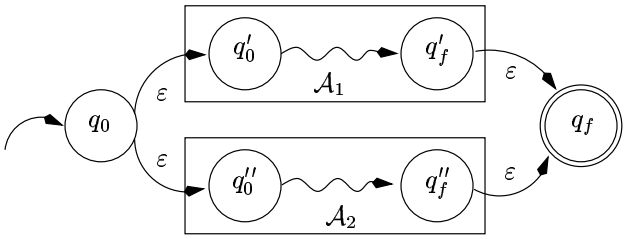
$$\mathcal{A} = \langle Q_1 \cup Q_2, \Sigma, q'_0, \delta, \{q''_f\}\rangle$$

where for all $q \in Q_1 \cup Q_2$ and $x \in \Sigma \cup \{\varepsilon\}$:

$$\delta(q,x) = \begin{cases} \delta_1(q,x) & \text{if } q \in Q_1 \setminus q'_f, \\ \delta_1(q'_f,x) & \text{if } q = q'_f \text{ and } x \neq \varepsilon, \\ \delta_1(q'_f,\varepsilon) \cup \{q''_0\} & \text{if } q = q'_f \text{ and } x = \varepsilon, \\ \delta_2(q,x) & \text{if } q \in Q_2. \end{cases}$$

Clearly there is a path from q'_0 to q''_f labeled w iff $w = u \cdot v$ where there is a path from q'_0 to q'_f labeled u and one from q''_0 to q''_f labeled v. Thus $L(\mathcal{A}) = L(\mathcal{A}_1) \cdot L(\mathcal{A}_2) = L(S_1 \cdot S_2)$. Furthermore, q'_f is the sole final state of $\mathcal{A}$.

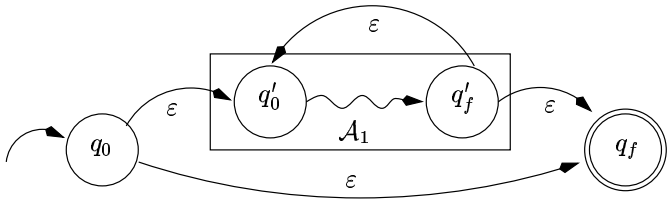
- Suppose $R = S_1 + S_2$ where S_1 and S_2 are regular expressions and that $L(S_1) = L(\mathcal{A}_1)$ and $L(S_2) = L(\mathcal{A}_2)$ for NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ with single final states. Let $\mathcal{A}$ be



30. Give the formal definition of $\mathcal{A}$ if $\mathcal{A}_1$ and $\mathcal{A}_2$ are as in the previous case.

Clearly there is a path from q_0 to q_f labeled w iff there is a path labeled w from q_0' to q_f' or one from q_0'' to q_f''. Thus $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2) = L(S_1 + S_2)$.

- Suppose, finally, that $R = S_1^*$ and that $L(S_1) = L(\mathcal{A}_1)$, etc. Let $\mathcal{A}$ be



A similar formal construction applies. Here any path from q_0 to q_f follows either the ε-transition directly from q_0 to q_f or consists of the concatenation of one or more paths from q_0' to q_f'.

31. Why does this construction not simply use the initial and final states of $\mathcal{A}_1$ as the initial and final states of $\mathcal{A}$, simply adding ε-transitions from q_0' to q_f' and from q_f' to q_0'? Give an example of an automaton $\mathcal{A}_1$ for which the simpler construction fails. This example will *not* be an automaton which would be constructed from a regular expression. The simpler construction *will* work in the context of the proof, but the induction hypothesis needs to be strengthened slightly. How? Explain why this suffices.

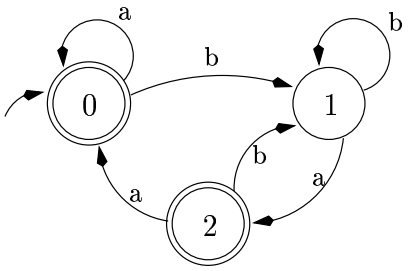
⊣

32. Construct an NFA accepting $L((a + bc)^*a + b)$.

9.2 Constructing Regular Expressions from DFAs

Lemma 11 *If $L \subseteq \Sigma^*$ is $L(\mathcal{A})$ for a DFA $\mathcal{A}$ then it is $L(R)$ for a regular expression R.*

The approach we will use to proving the lemma involves constructing and solving a system of simultaneous equations defining, for each state, the set of strings labeling paths from the initial state to that state. For concreteness, we will work an example, based on the following DFA, while we develop the proof.



For each $q \in Q$ let

$$R_q \stackrel{\text{def}}{=} \{w \mid \hat{\delta}(q_0, w) = q\}.$$

Let's consider which strings are in R_q for a given state q. Clearly R_q includes ε iff $q = q_0$. Otherwise, every string in R_q is of the form $w\sigma$, where w is in R_p and there is a transition from state p to state q on σ (i.e., $\delta(p, \sigma) = q$). Thus

$$R_q = \begin{cases} \{\varepsilon\} \cup \bigcup_{\delta(p,\sigma)=q} [R_p \cdot \{\sigma\}] & \text{if } q = q_0, \\ \bigcup_{\delta(p,\sigma)=q} [R_p \cdot \{\sigma\}] & \text{otherwise .} \end{cases}$$

For the example automaton, we get a system of three equations:

$$\begin{aligned} R_0 &= R_0 a + R_2 a + \varepsilon \\ R_1 &= R_0 b + R_1 b + R_2 b \\ R_2 &= R_1 a \end{aligned}$$

Here we have used, for instance, '$R_0 a$' to denote the concatenation of the languages R_0 and $\{a\}$ and '+' to denote union. (Equations of this form are known as *Regular Equations.*)

Our objective is to obtain a solution to the system of regular equations in the form of regular expressions denoting each of the R_q. Then, since the

language accepted by a DFA is the set of all strings labeling paths from the initial state to some final state, which is to say

$$L(\mathcal{A}) = \bigcup_{q \in F} [R_q],$$

we can obtain a regular expression denoting $L(\mathcal{A})$ by combining (with '+') the regular expressions denoting R_q for each $q \in F$.

The question, then, is how to obtain the solution to the system of regular equations. The crucial step, of course, is eliminating occurrences of R_q from the right hand side of the definition of R_q. For this we employ the following lemma.

Lemma 12 *Let P, Q and R be sets of strings with $\varepsilon \notin P$. Then the equation*

$$R = Q + RP$$

has a unique solution

$$R = QP^*.$$

Proof: It is useful to consider what happens when we substitute $Q + RP$ into itself for R:

$$\begin{aligned} R &= Q + RP \\ &= Q + (Q + RP)P = Q + QP + RP^2 \\ &= Q + QP + (Q + RP)P^2 = Q + QP + QP^2 + RP^3 \\ &\vdots \\ &= Q + QP + QP^2 + \cdots + QP^i + RP^{i+1} \\ &\vdots \end{aligned}$$

Thus, $QP^i \subseteq R$ for all $i \geq 0$. Consequently, $\bigcup_{i \geq 0} [QP^i] \subseteq R$, which is to say, $QP^* \subseteq R$.

In fact, $R = QP^*$ is *a* solution to $R = Q + RP$, which we can verify by substituting QP^* for R in $R = Q + RP$:

$$QP^* = Q + QP^*P = Q(\varepsilon + PP^*) = QP^*.$$

We now need to show that this solution is unique. We will do this by showing that whenever $R = S$ is a solution then $S \subseteq QP^*$. Then, since $QP^* \subseteq R$ and $R = S$ we will have $S = QP^*$.
Suppose $R = S$ and $w \in S$. Then, from above, we have, for all $i \geq 0$,

$$w \in Q + QP + QP^2 + \cdots + QP^i + RP^{i+1},$$

and, in particular,

$$Q + QP + QP^2 + \cdots + QP^{|w|} + RP^{|w|+1}.$$

Now, since $\varepsilon \notin P$ every string in $RP^{|w|+1}$ is strictly longer than $|w|$. Thus $w \notin RP^{|w|+1}$. It must be the case, then, that

$$w \in Q + QP + QP^2 + \cdots + QP^i$$

and, therefore, $w \in QP^*$. Since the choice of w was arbitrary, we have that $w \in S \Rightarrow w \in QP^*$. In other words, $S \subseteq QP^*$. ⊣

The proof of the uniqueness of QP^* as a solution depends on the fact that $\varepsilon \notin P$ (otherwise $RP^{|w|+1}$ may include strings that are not strictly longer than $|w|$), but the proof that QP^* is a solution does not. Presumably if $\varepsilon \in P$ then, while QP^* will still be a solution, there may be other solutions.

33. Show that if $P = \{\varepsilon\}$ then every set that contains Q as a subset is a solution to $R = Q + RP$.

34. Give an example of an R, P and Q such that $R = Q + RP$ but $R \neq QP^*$.

To see how to use this result to solve the system of equations, consider the equations we obtained for the example DFA. The idea is to focus on one equation at a time, identifying Q and P such that the equation is of the form $R = Q + RP$:

$$R_0 = R_0a + R_2a + \varepsilon = \underbrace{R_2a + \varepsilon}_{Q} + R_0 \underbrace{a}_{P}.$$

Then

$$\begin{aligned} R_0 &= QP^* \\ &= (R_2a + \varepsilon)a^* \\ &= R_2aa^* + a^*. \end{aligned}$$

Substituting R_1a for R_2

$$R_0 = R_1aaa^* + a^*.$$

Substituting these into the equation for R_1

$$\begin{aligned} R_1 &= R_0b + R_1b + R_2b \\ &= (R_1aaa^* + a^*)b + R_1b + R_1ab \\ &= R_1aaa^*b + a^*b + R_1b + R_1ab \\ &= a^*b + R_1(b + ab + aaa^*b) \\ &= \underbrace{a^*b}_{Q} + R_1 \underbrace{a^*b}_{P}, \qquad \text{since } b + ab + aaa^*b = a^*b. \end{aligned}$$

Then

$$R_1 = a^*b(a^*b)^*.$$

Finally, substituting this into the equations for R_0 and R_2:

$$\begin{aligned} R_0 &= a^*b(a^*b)^*aaa^* + a^* \text{ and} \\ R_2 &= a^*b(a^*b)^*a. \end{aligned}$$

It follows, then, that

$$\begin{aligned} L(\mathcal{A}) &= R_0 + R_2 \\ &= a^*b(a^*b)^*aaa^* + a^* + a^*b(a^*b)^*a \\ &= a^* + a^*b(a^*b)^*aa^*. \end{aligned}$$

35. Construct a regular expression denoting the language accepted by the following DFA.