Lecture 5: Equivalence of DFAs & NFAs

Michael Engling

Department of Computer Science
Stevens Institute of Technology

CS 334 Automata and Computation
Fall 2015

OUTLINE: Nondeterministic Finite Automata

Constructing a DFA from an NFA

Outline of Proof

Example

NFAs Recognize Regular Languages

Which Form To Prefer

Closure Under Kleene Star

Constructing a DFA from an NFA

In order to show that DFAs and NFAs recognize *precisely* the same languages, we need to show two things:

1. Given any DFA, we can find a corresponding NFA which recognizes the same language as the DFA, and

2. Given any NFA, we can find a corresponding DFA which recognizes the same language as the NFA.

The first of these tasks is trivial. All DFAs are, by default, also NFAs.

The second is more difficult and more taxing, but it can be done methodically with a straightforward construction. That is the task at hand....

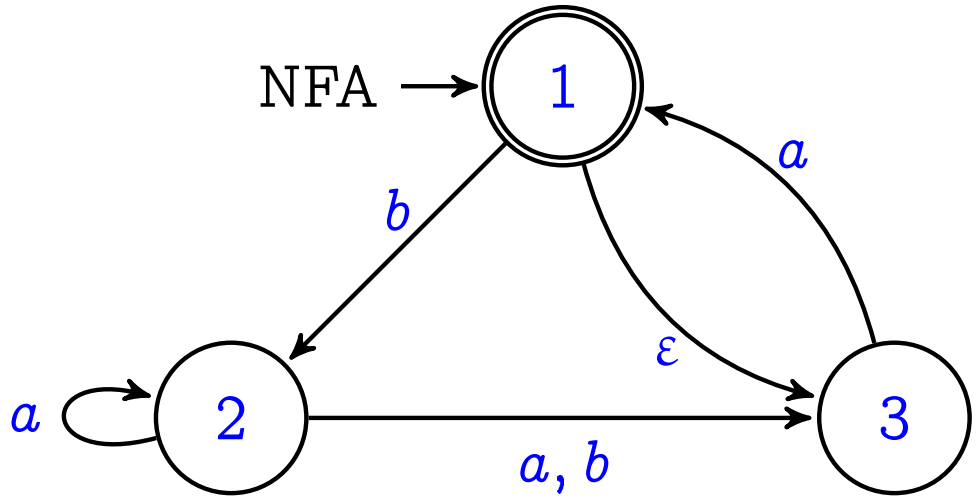
Outline of (Constructive) Proof

We are given an NFA (and only a formal NFA), so we must begin there.

$$N = (Q, \Sigma, \delta, q_0, F)$$

We need to construct D with:
$Q_D = \mathcal{P}(Q)$ A string in the NFA may lead to multiple states.
$\Sigma_D = \Sigma$ We'll need the same alphabet.
$\delta_D(R, a) = \{q \in Q | q \in E(\delta(r, a)) \text{ for some } r \in R\}$. This says "Use the transition function of N, and don't forget the ε-edges."
$q_{0D} = \{E(q_0)\}$. This is the start state of the NFA plus any states reachable via ε-edges for the start state.
$F_D = \{R \in \mathcal{P}(Q) | R \cap F \neq \emptyset\}$. So R contains a final state of the original NFA.

Example



What language does our NFA recognize?

Certainly it recognizes a^* from states 1 & 3,

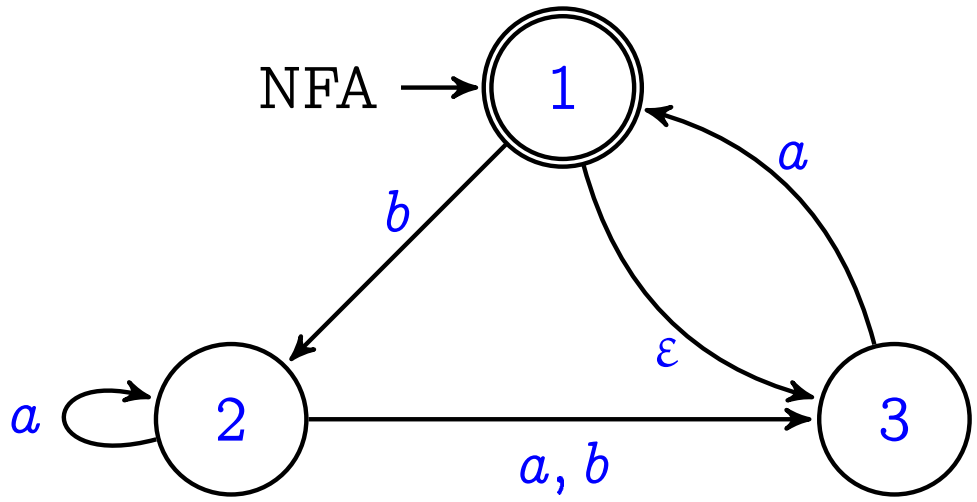
Plus, $bba = b^2a$, 1 to 2 to 3,

Also, $baa = ba^2$, 1 to 2 to 3,

And, ba^*ba, 1 to 2 (to 2, etc.) to 3.

Furthermore, we can concatenate these in any order we like and star them: $(a^*(bba)^*(ba^2)^*(ba^*ba)^*)^*$ Did we miss anything?

Example



Notice that in our NFA:

From state 1, with an a, we hit a dead end (despite ε).

From state 1, with a b, we can only go to 2.

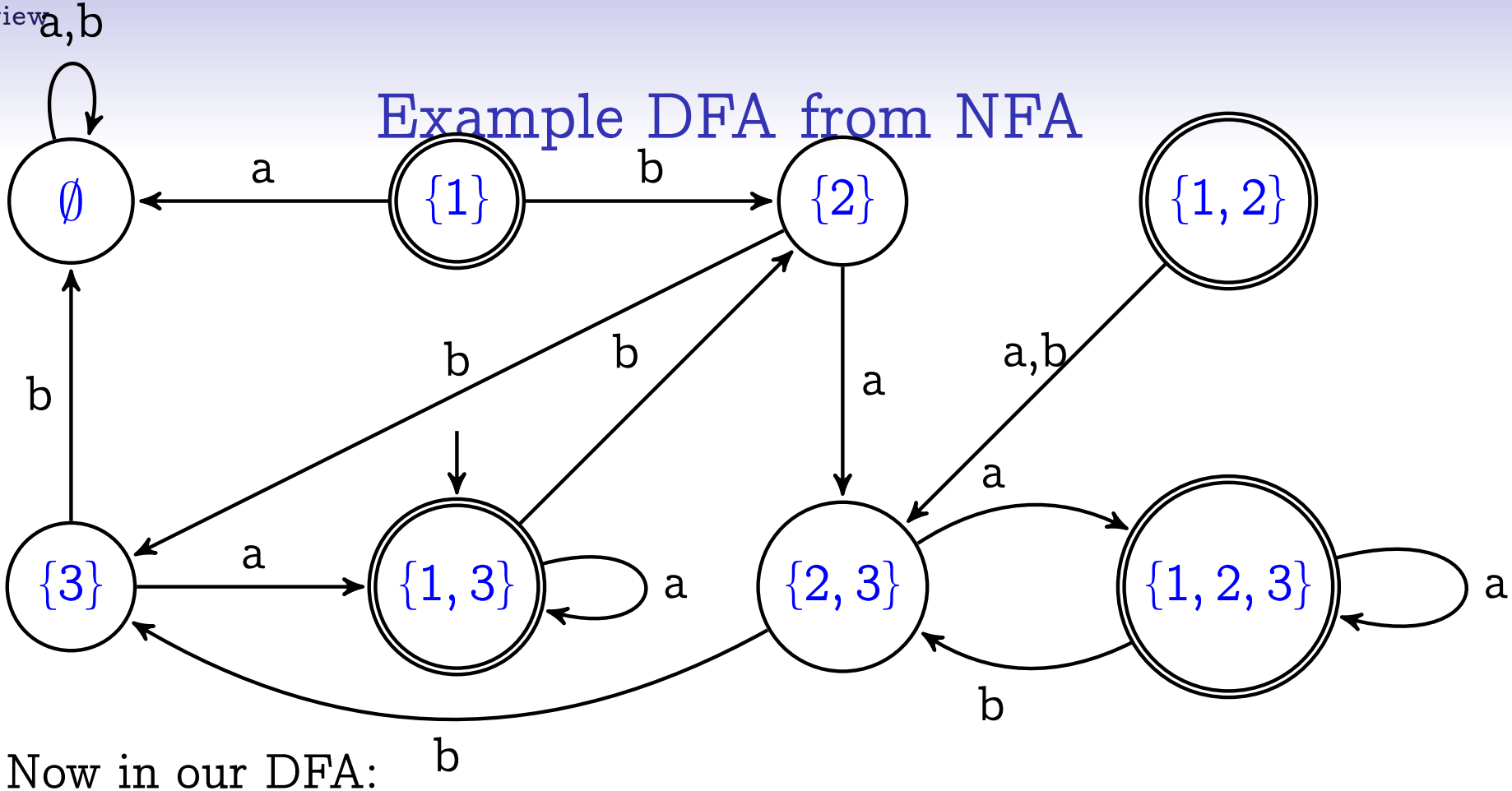
From state 2, with an a, we can stay in 2 OR go to 3,

From state 2, with a b, we can only go to 3,

From state 3, with an a, we can go to 1 or return to 3, and

From state 3, with a b, we hit a dead end.

Example DFA from NFA



Now in our DFA:

From state $\{1\}$, with an a, we go to h$\emptyset$ll.

From state $\{1\}$, with a b, we go to $\{2\}$.

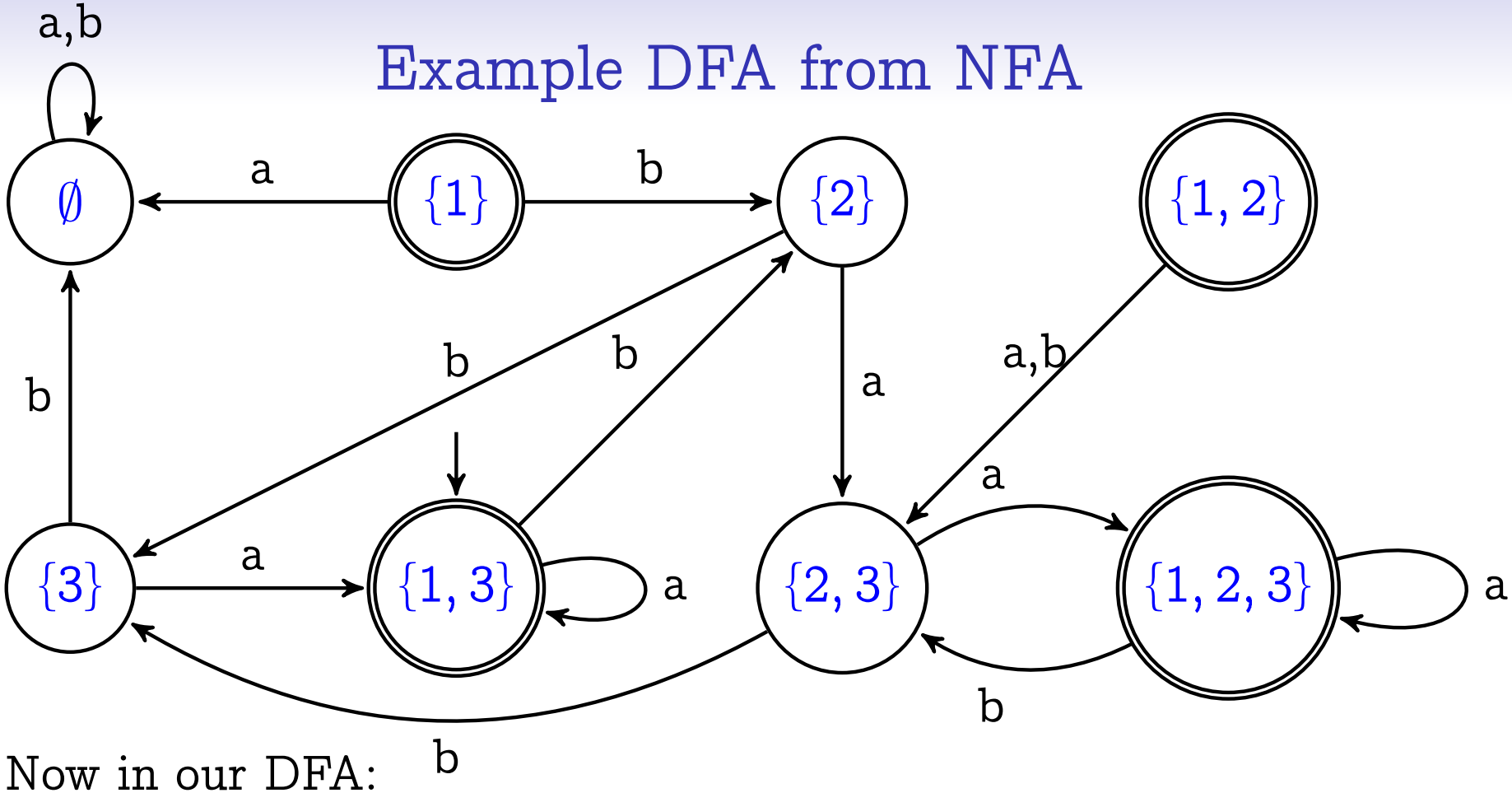
From state $\{2\}$, with an a, go to $\{2, 3\}$,

From state $\{2\}$, with a b, we can only go to $\{3\}$,

From state $\{3\}$, with an a, we can go to $\{1, 3\}$, and

From state $\{3\}$, with a b, we go to h$\emptyset$ll.

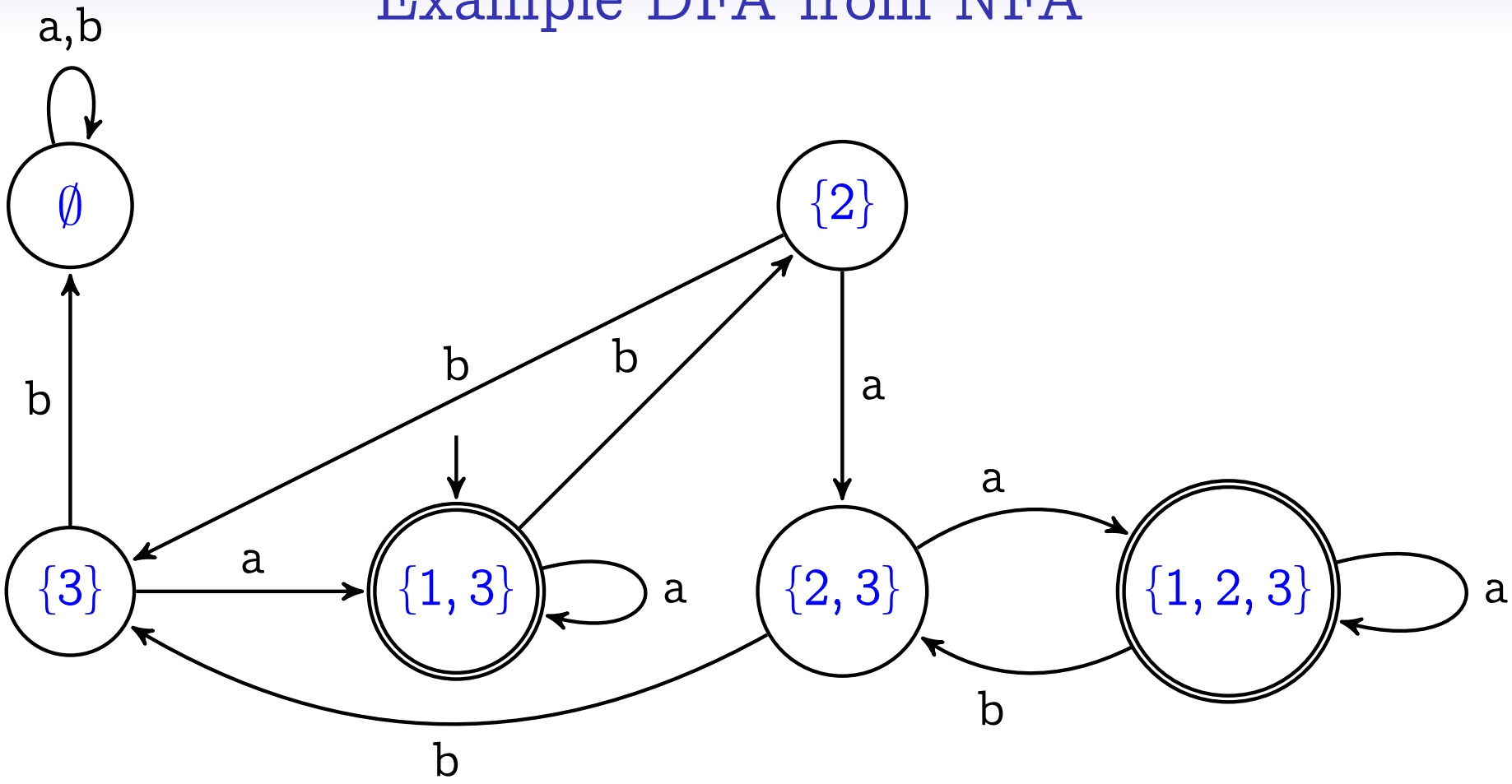
Example DFA from NFA



Now in our DFA:

Note that in our DFA, states $\{1\}$ and $\{1,2\}$ are *unreachable*. That is, there are no edges (not even start edge) entering them. Thus we can delete them wihtout affecting the performance of the machine.

Example DFA from NFA



Does our DFA recognize $(a^*(bba)^*(ba^2)^*(ba^*ba)^*)^*$?

NFAs Recognize Regular Languages

Thus, for every DFA there is an NFA (itself), and for every NFA we can construct a corresponding DFA which recognizes the same language.

Since we defined regular languages as those recognized by DFAs, this means that a language is regular if and only if it is recognizes by an NFA.

Which Form To Prefer

Deterministic Finite Automata seem more rigid and structured than Nondeterministic ones. We always know, for example, that each string fed to a DFA will follow a unique path and terminate in a particular state. Neither notion is true for NFAs.

For this reason, we will sometimes use DFAs to prove negative aspects of our languages. Indeed, on Friday we will use a DFA to show that a particular language is *not* regular. This will give us an insight on how to show MANY languages are not regular.

In contrast, Nondeterministic Finite Automata seem more flexible and easier to build. Indeed, sometimes we can assemble them almost immediately from a regular expression.

For this reason, we will often use NFAs to prove positive aspects of our languages by constructing NFAs (rather than their more demanding DFA counterparts.) Indeed, we'll do so straightaway:

Closure Under Kleene Star

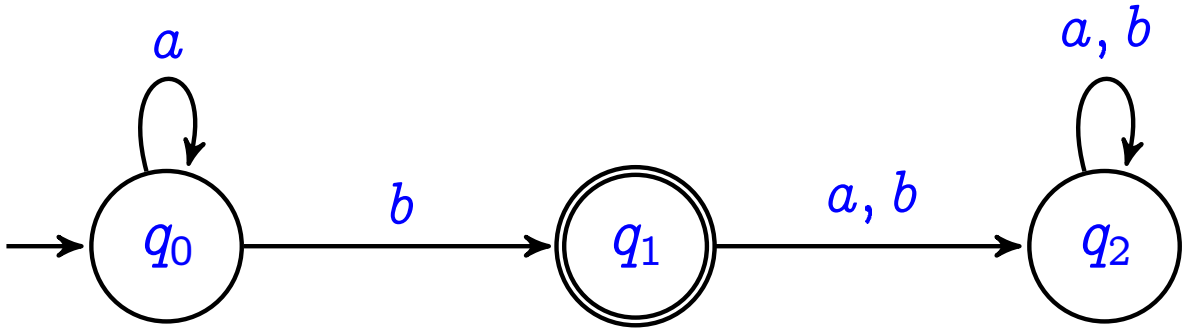
We used an elaborate construction with DFAs to show that the regular languages are closed under the union operation, $L_1 \cup L_2$.

Then we saw how easy it was to demonstrate the same idea with NFAs by using a new start statement and connecting to the start statements of our DFAs via ε edges.

We also used NFAs to tie strings together for the concatenation operation, $L_1 \circ L_2$, this time using ε edges to connect the final states of the first automaton to the start state of the second.

We'll now present one more variation on this theme to show the regular languages are closed under the Kleene Star operation, L^*. That is, if L is a regular language, then L^* is also regular.

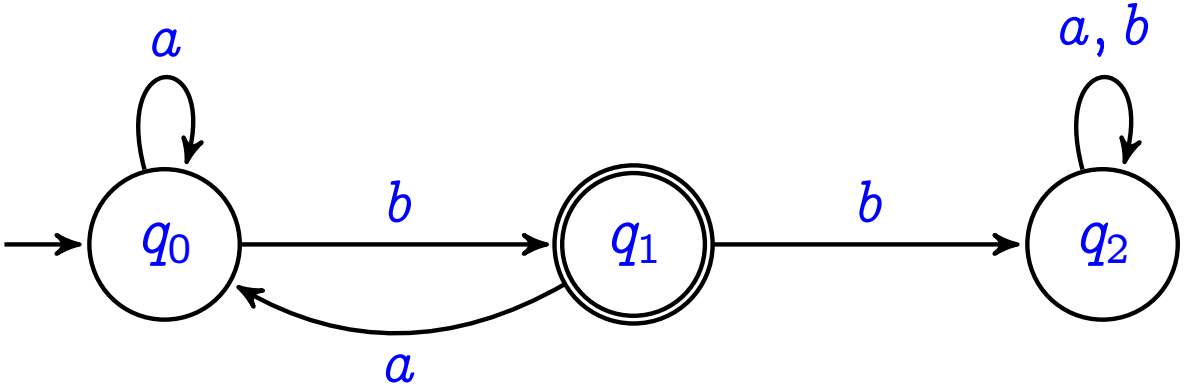
Closure Under Kleene Star



This is a DFA recognizing a^*b.

Suppose we wish to recognize $(a^*b)^*$?

Closure Under Kleene Star



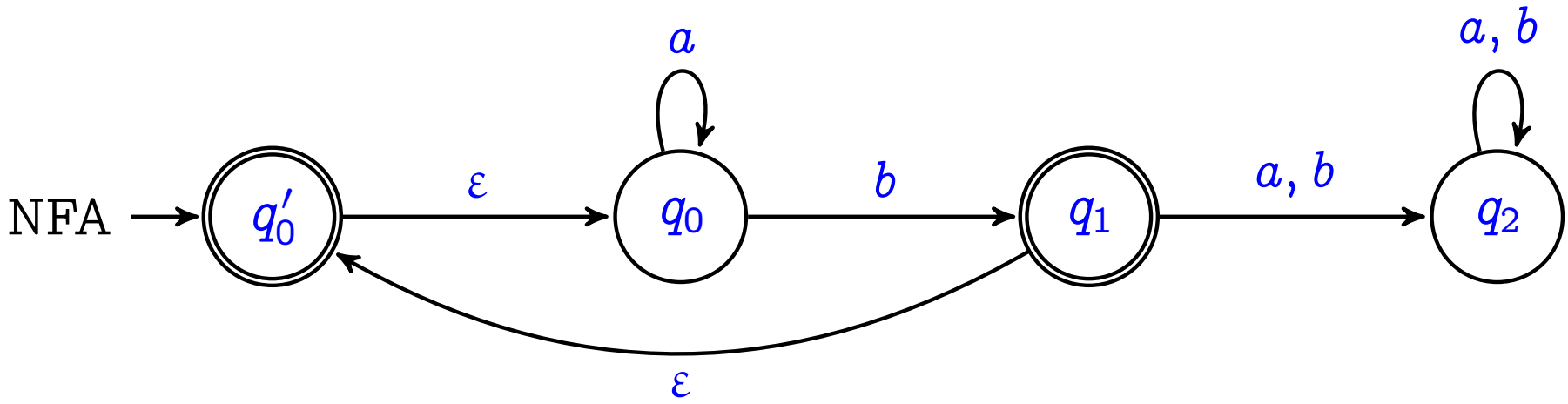
This is a DFA recognizing a^*b.

Suppose we wish to recognize $(a^*b)^*$?

Tinkering with the DFA is harder than it seems.

Why doesn't the above work?

Closure Under Kleene Star



This is an NFA recognizing $(a^*b)^*$.

This suggests a constructive proof showing that the regular languages are closed under the Kleene star operation:

Create a new (accepting) start state, connect it to the DFA start state via ε edge, and connect all final states (why all?) to the start state via ε edges.