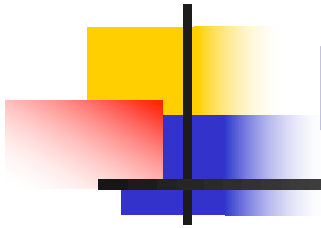
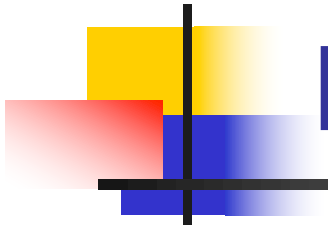
Equivalence of NFAs and DFAs + Closure under the regular operations



Equivalence of NFAs and DFAs

- DFAs and NFAs recognize the same class of languages
- The equivalence is both *surprising* and *useful*
  - **Surprising**: NFAs *appear* to have more power than DFAs
  - **Useful**: describing an NFA for a given language is sometimes much easier than describing a DFA for that language

- We say that two machines are ***equivalent*** if they recognize the same language



Equivalence of NFAs and DFAs

Theorem: Every NFA has an equivalent DFA.

Proof Idea:

- If a language is recognized by an NFA, then we must show the existence of a DFA that also recognizes it
- The idea is to convert the NFA into an equivalent DFA that *simulates* the NFA
- Key point of the simulation is keeping track of the set of states in the "tree of possibilities" view of nondeterministic computation
- If k is the number of states of the NFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA will have 2^k states
- To finalize, need to figure out which will be the *start state* and *accept states* of the DFA, and what will be the *transition function*

Formal proof
(case where there are no ε-arrows)

PROOF Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. Before doing the full construction, let's first consider the easier case wherein N has no ε arrows. Later we take the ε arrows into account.

Formal proof
(case where there are no ε-arrows)

1. $Q' = \mathcal{P}(Q)$.
   Every state of M is a set of states of N. Recall that $\mathcal{P}(Q)$ is the set of subsets of Q.
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q | \ q \in \delta(r, a) \text{ for some } r \in R\}$.
   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it shows where a takes each state in R. Because each state may go to a set of states, we take the union of all these sets. Another way to write this expression is
   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$[4]
3. $q_0' = \{q_0\}$.
   M starts in the state corresponding to the collection containing just the start state of N.
4. $F' = \{R \in Q' | \ R \text{ contains an accept state of } N\}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

[4]The notation $\bigcup_{r \in R} \delta(r, a)$ means: the union of the sets $\delta(r, a)$ for each possible r in R.

Formal proof
(case where there are ε-arrows)

Now we need to consider the ε arrows. To do so, we set up an extra bit of notation. For any state R of M, we define $E(R)$ to be the collection of states that can be reached from members of R by going only along ε arrows, including the members of R themselves. Formally, for $R \subseteq Q$ let

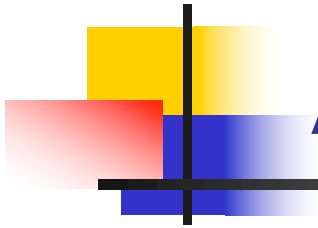
$$E(R) = \{q |\ q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}.$$

Then we modify the transition function of M to place additional fingers on all states that can be reached by going along ε arrows after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this effect. Thus

$$\delta'(R, a) = \{q \in Q |\ q \in E(\delta(r, a)) \text{ for some } r \in R\}.$$

Formal proof
(case where there are ε-arrows)

Additionally, we need to modify the start state of M to move the fingers initially to all possible states that can be reached from the start state of N along the ε arrows. Changing q_0' to be $E(\{q_0\})$ achieves this effect. We have now completed the construction of the DFA M that simulates the NFA N.



Alternative characterization of regular languages

Corollary: A language is regular if and only if some NFA recognizes it.

- ==> (a language is regular if some NFA recognizes it)
  - The theorem we just proved shows that any NFA can be converted into an equivalent DFA. Consequently, if an NFA recognizes some language, so does some DFA, and hence the language is regular.
- <== (a language is regular only if some NFA recognizes it)
  - IOW: if a language is regular, some NFA must be recognizing it. Obviously, this condition is true because a regular language has a DFA recognizing it and any DFA is also an NFA.

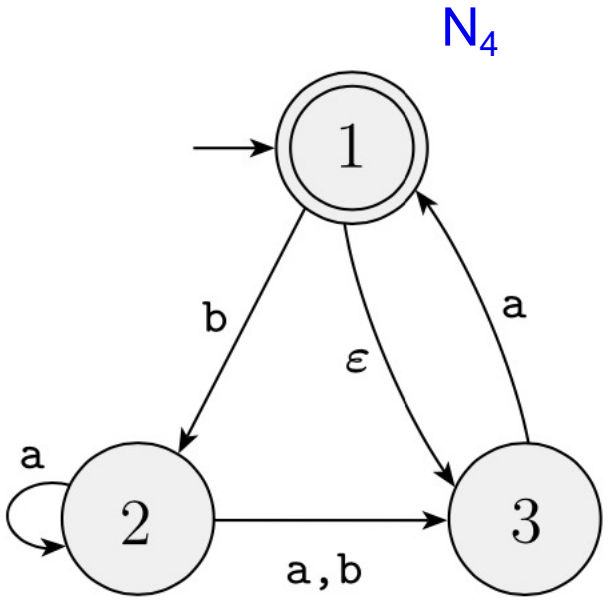
Example: converting NFA to DFA

Consider the NFA $N_4 = (Q, \{a,b\}, \delta, 1, \{1\})$
with $Q = \{1, 2, 3\}$

To construct a DFA D that is equivalent to N_4 , we first determine D's states.

Since N_4 has three states, {1,2,3}, we construct D with eight states, one for each subset of N_4's states.

D's sates are:
{∅, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}



Example: converting NFA to DFA

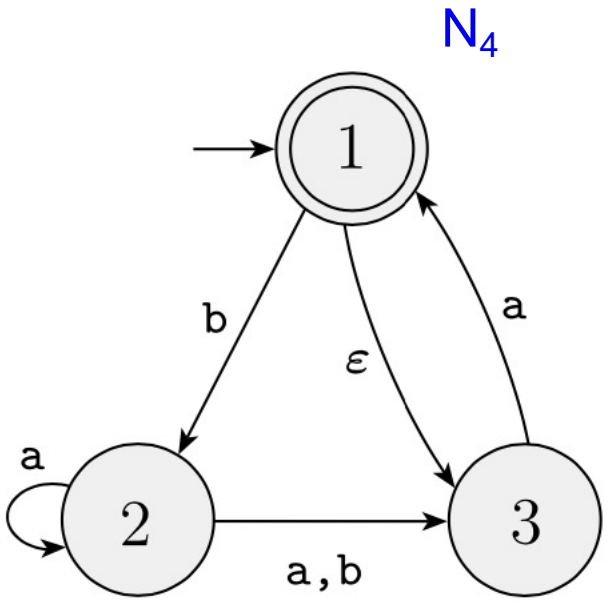
Consider the NFA $N_4 = (Q, \{a,b\}, \delta, 1, \{1\})$
with $Q = \{1, 2, 3\}$

Next, we determine the start and accept states of D.

The start state is E({1}), the set of states reachable from 1 by traveling along ε arrows, plus 1 itself.

An ε arrow goes from 1 to 3, so E({1}) = {1,3}.

The new accept states are those containing N_4's accept states; thus {{1}, {1,2}, {1,3}, {1,2,3}}



Example: converting NFA to DFA

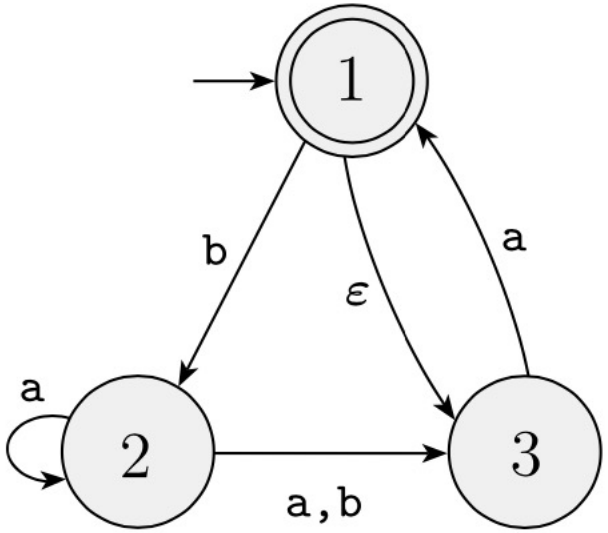
Consider the NFA $N_4 = (Q, \{a,b\}, \delta, 1, \{1\})$
with $Q = \{1, 2, 3\}$

Finally, we determine D's transition function.

Each of D's states goes to one place on input a and one place on input b.

Examples:
In D, state {2} goes to {2,3} on input a because in N_4, state 2 goes to both 2 and 3 on input a and we can't go farther from 2 or 3 along ε arrows.
State {2} goes to state {3} on input b because in N_4, state 2 goes only to state 3 on input b and we can't go farther from 3 along ε arrows.

N_4



Example: converting NFA to DFA

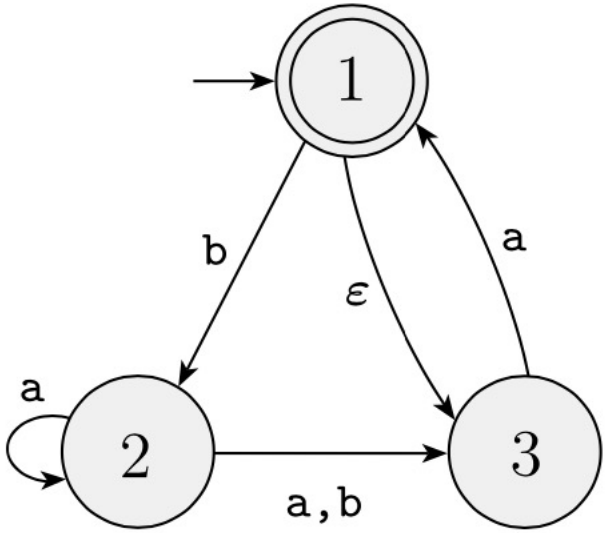
More examples:

State {1} goes to ∅ on a because no arrows exit it.
It goes to {2} on b.

State {3} goes to {1,3} on a because in N_4, state 3 goes to 1 on a and 1 in turn goes to 3 with an ε arrow.
State {3} on b goes to ∅.

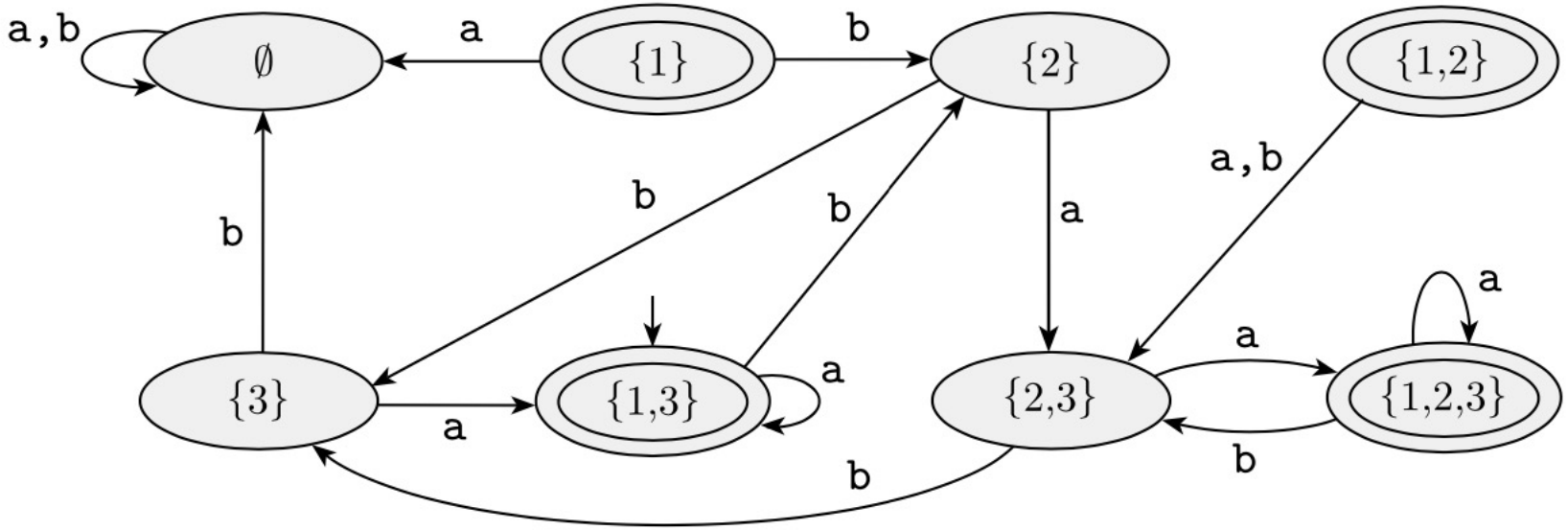
State {1,2} on a goes to {2,3} because 1 points at no states with a arrows, 2 points at both 2 and 3 with a arrows, and neither points anywhere with ε arrows.
State {1,2} on b goes to {2,3}.

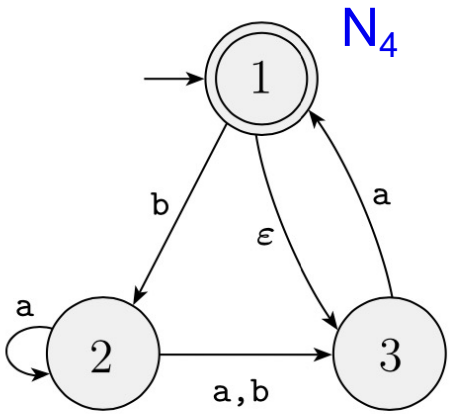
N_4



Example: converting NFA to DFA

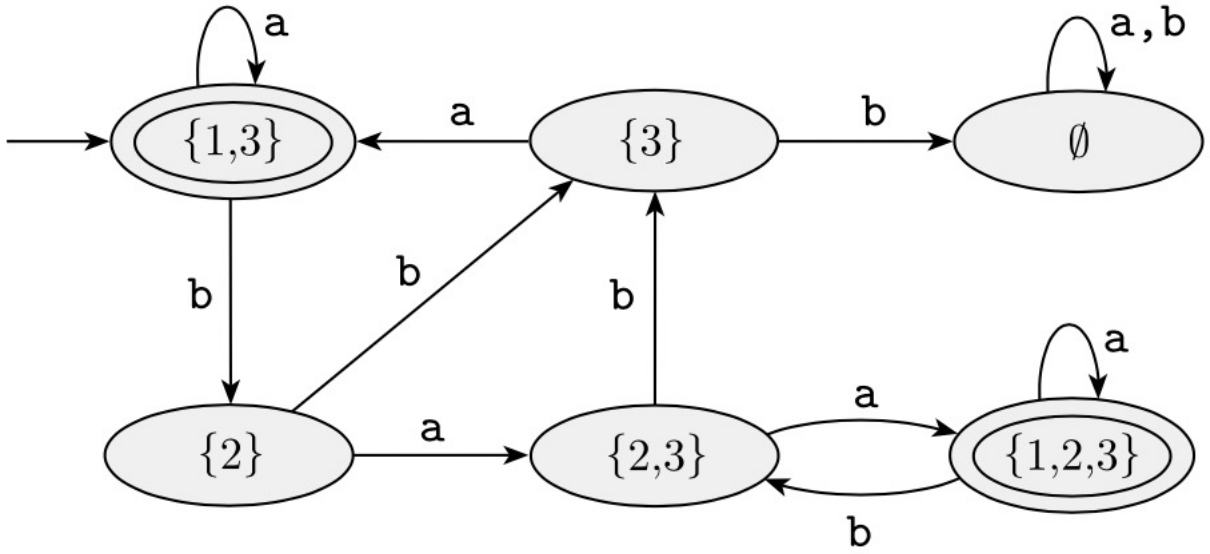
Continuing in this way, we obtain the diagram for D shown below:

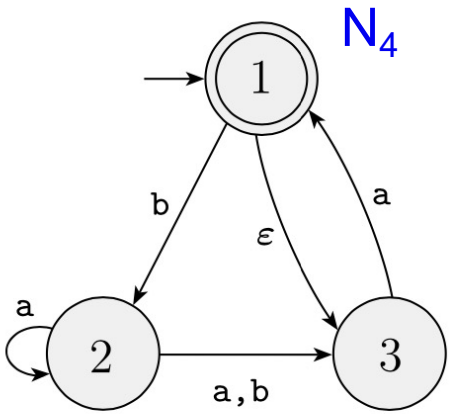


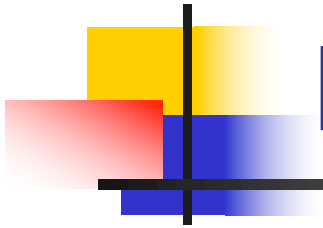


Example: converting NFA to DFA

We can simplify the previous machine by observing that no arrows point at states {1} and {1,2}, so they may be removed without affecting the performance of the machine:

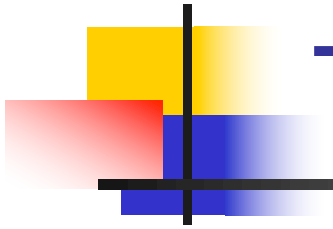






Returning to our closure theorems…

- Our aim is to prove that the *union*, *concatenation*, and *star* of regular languages are still regular.
- We abandoned the original attempt to do so when dealing with concatenation operation was too complicated. The use of nondeterminism makes the proof much easier.
- First, let us re-consider the closure under union theorem again to see how its proof can be simplified using nondeterminism.

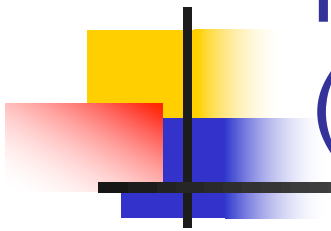


Theorem

The class of regular languages is closed under the union operation

IOW: if A_1 and A_2 are regular languages, so is $A \cup B$.

Proof Idea
(that we saw in earlier lecture and uses DFAs)



- Have regular languages A_1 and A_2 and want to show that $A_1 \cup A_2$ also is regular
- Because A_1 and A_2 are regular, we know that some FA M_1 recognizes A_1 and some FA M_2 recognizes A_2.
- To prove that $A_1 \cup A_2$ is regular, we demonstrate a FA M that recognizes $A_1 \cup A_2$.

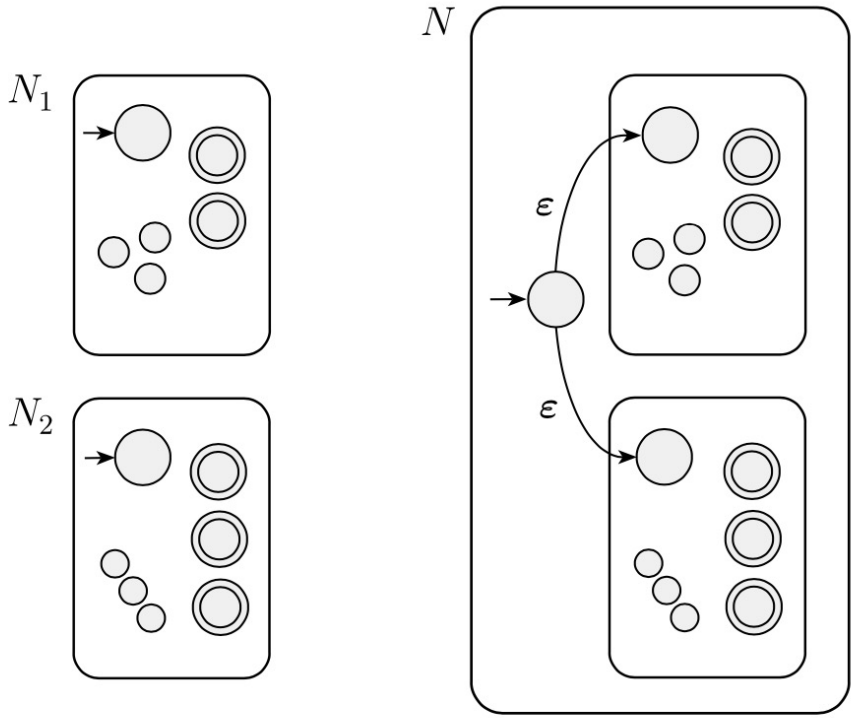
- The proof is by *construction*. We construct M from M_1 and M_2.
- M must accept its input exactly when either M_1 or M_2 would accept it in order to recognize the union language.
- It works by **simulating** both M_1 and M_2 and accepting if either of the simulations accept.
- An approach that works: remember ***pairs*** of states (to do the simulation simultaneously)

Proof using NFAs (much simpler)

PROOF IDEA We have regular languages A_1 and A_2 and want to prove that $A_1 \cup A_2$ is regular. The idea is to take two NFAs, N_1 and N_2 for A_1 and A_2, and combine them into one new NFA, N.

Machine N must accept its input if either N_1 or N_2 accepts this input. The new machine has a new start state that branches to the start states of the old machines with ε arrows. In this way, the new machine nondeterministically guesses which of the two machines accepts the input. If one of them accepts the input, N will accept it, too.

Proof using NFAs



Proof using NFAs (formal)

PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.
2. The state q_0 is the start state of N.
3. The set of accept states $F = F_1 \cup F_2$.
   The accept states of N are all the accept states of N_1 and N_2. That way, N accepts if either N_1 accepts or N_2 accepts.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$