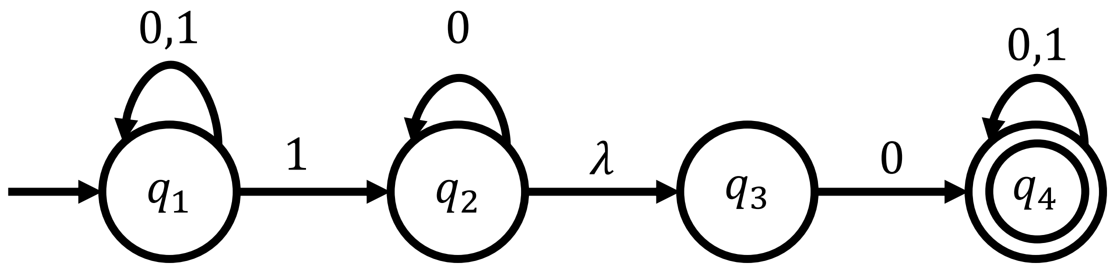
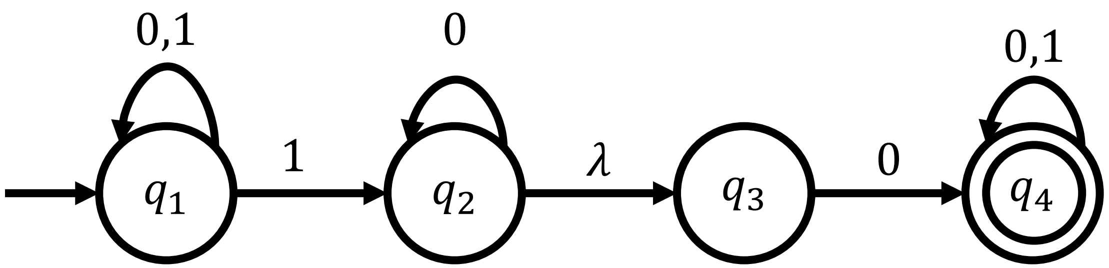
Nondetermism (1.2)

COT 4210 Discrete Structures II
Summer 2025
Department of Computer Science
Dr. Steinberg



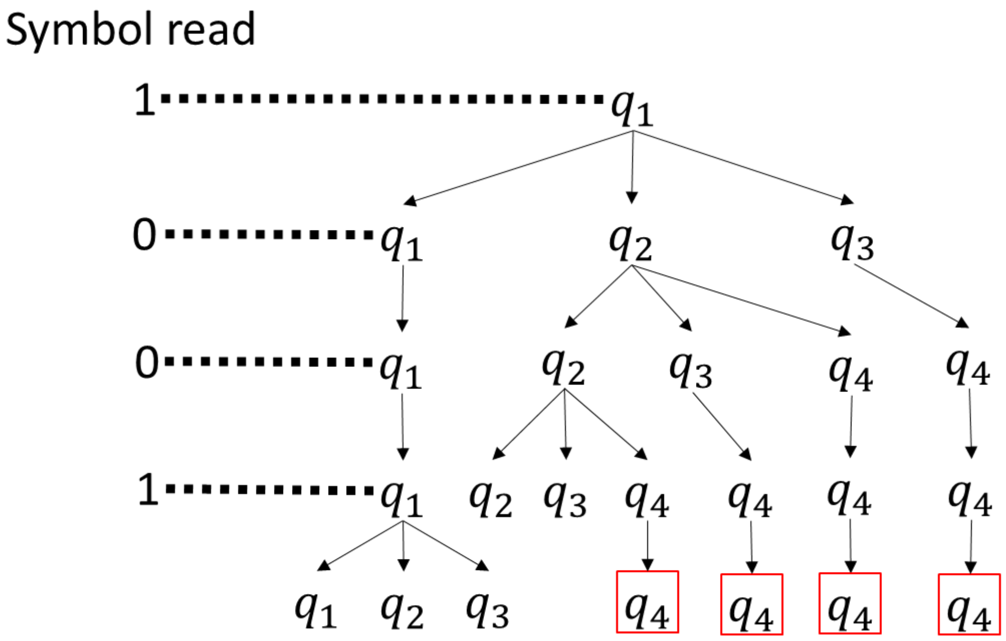
Nondeterminism

- An *Nondeterministic Finite Automata* looks like a DFA, *except* an NFA:
  - Can have more than one possible transition per state per input symbol
  - Doesn't have to have a transition for every state for every input symbol
  - Can transition on the empty string
- It also *works* like a DFA, except acceptance is nondeterministic
  - A DFA accepts if *the* path for the input string ends on an accept state
  - An NFA accepts if *any* path for the input string ends on an accept state

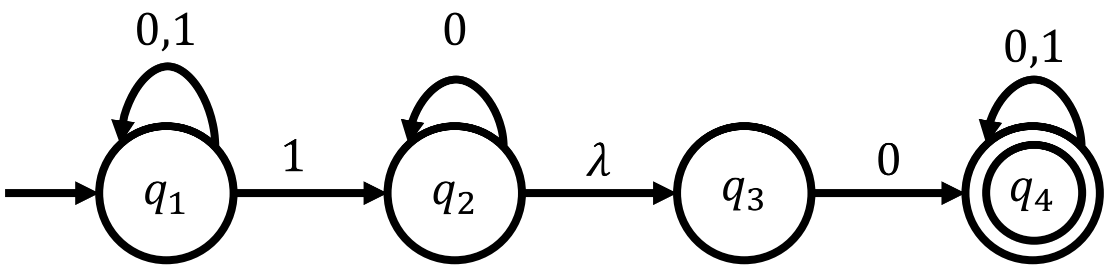


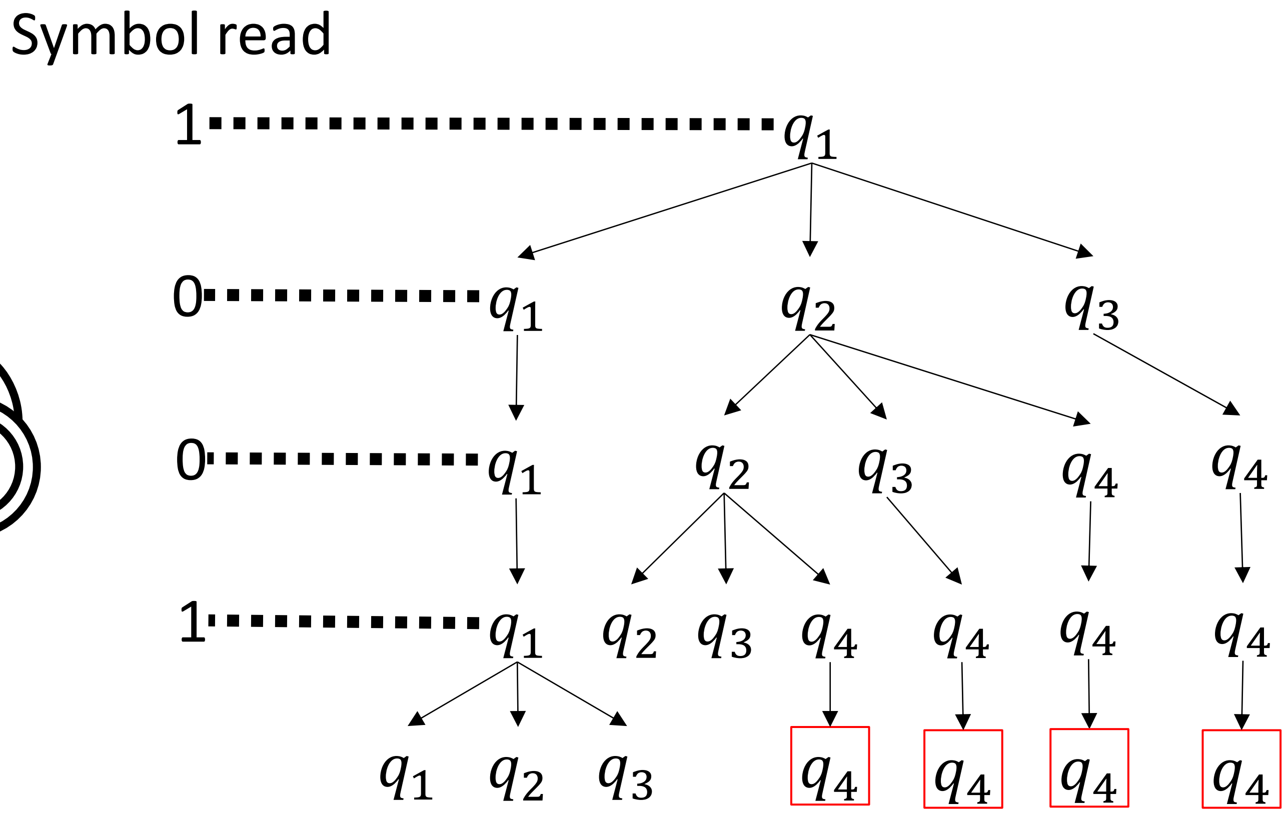
Computation in an NFA

- The easiest way to think of how an NFA works is to think of a *threaded DFA*
  - Every time there is a choice of more than one path, the NFA splits off a copy of itself to follow each path
  - The copies conceptually run in parallel
  - A copy that reaches the end of input either accepts or rejects normally
  - A copy that reaches a symbol it cannot transition on stops and rejects
  - The NFA itself accepts if *any* copy accepts

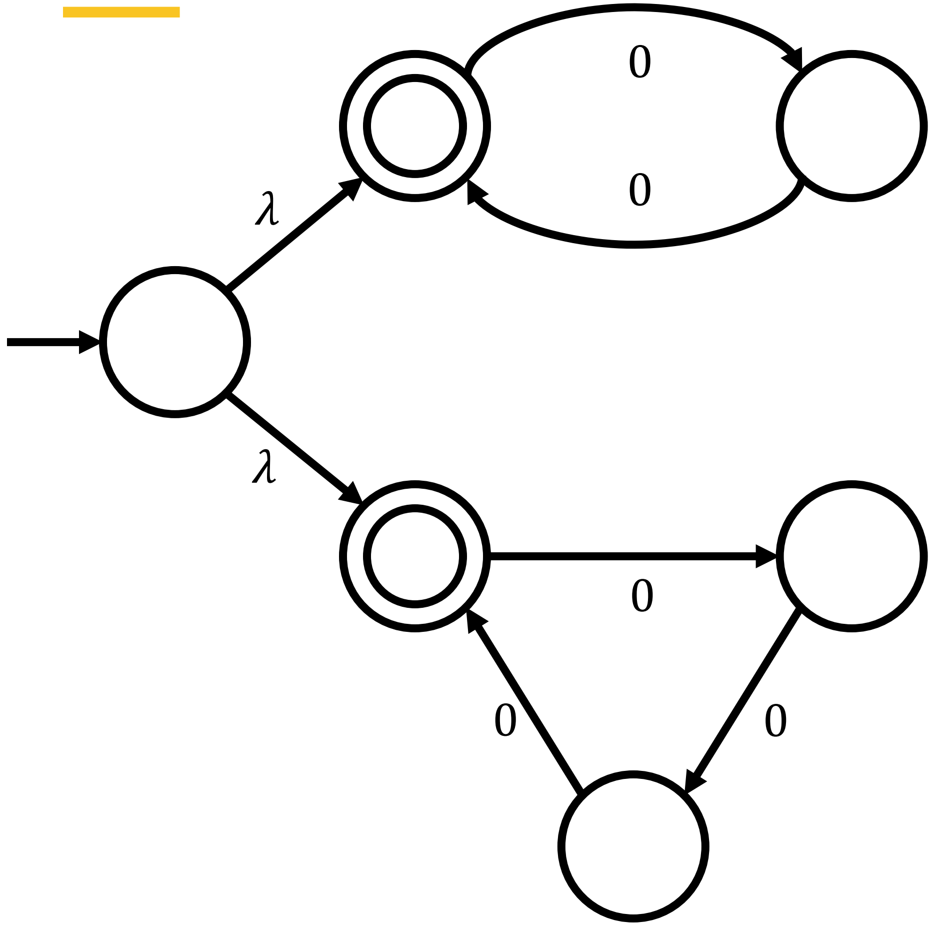


Input 1001 on our NFA



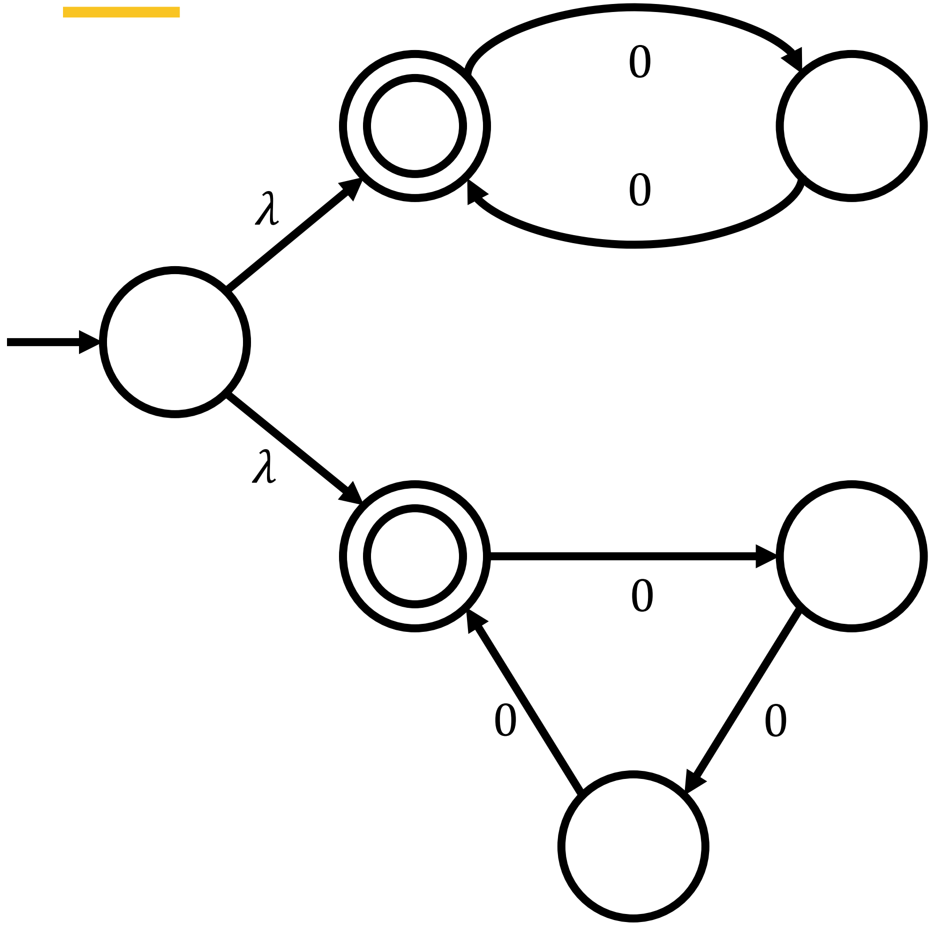


NFA Example



What language does this machine recognize?

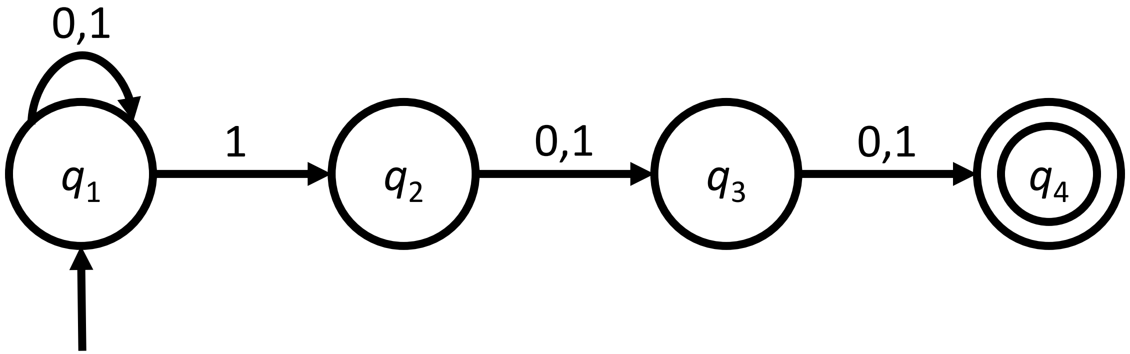
NFA Example

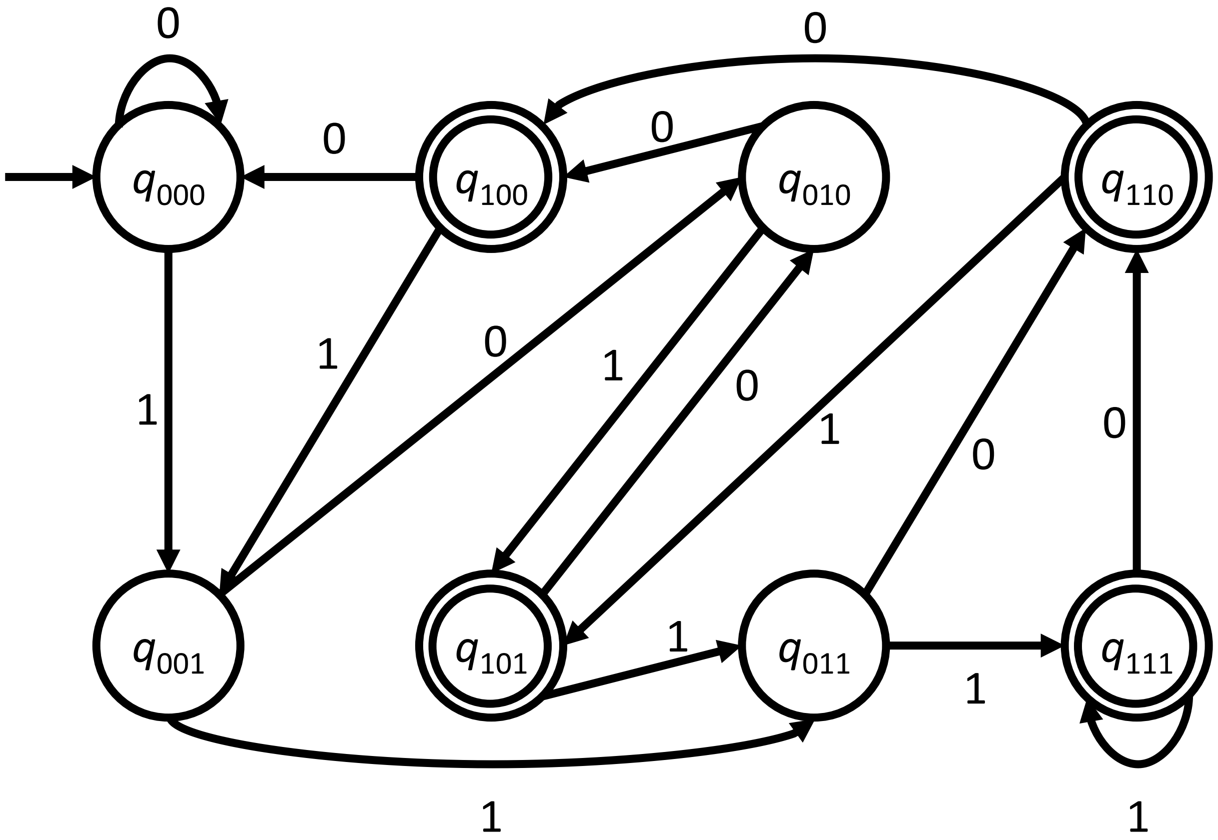


What language does this machine recognize?

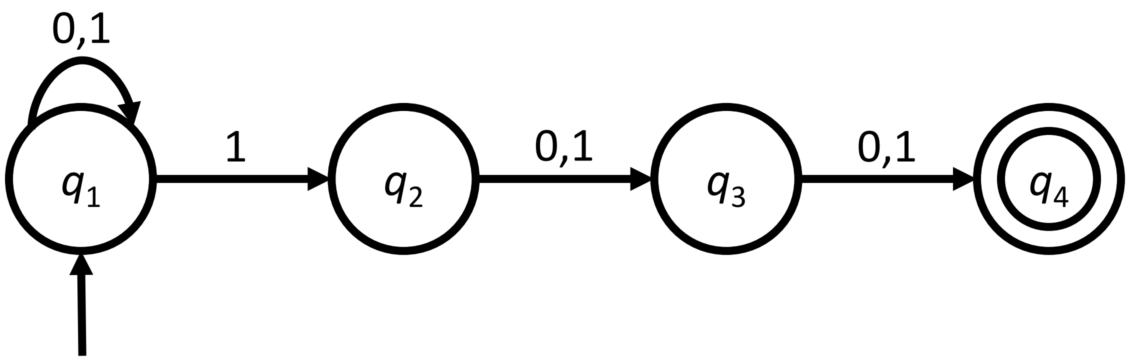
$L = \{0^n : n \bmod 2 = 0 \text{ or } n \bmod 3 = 0\}$

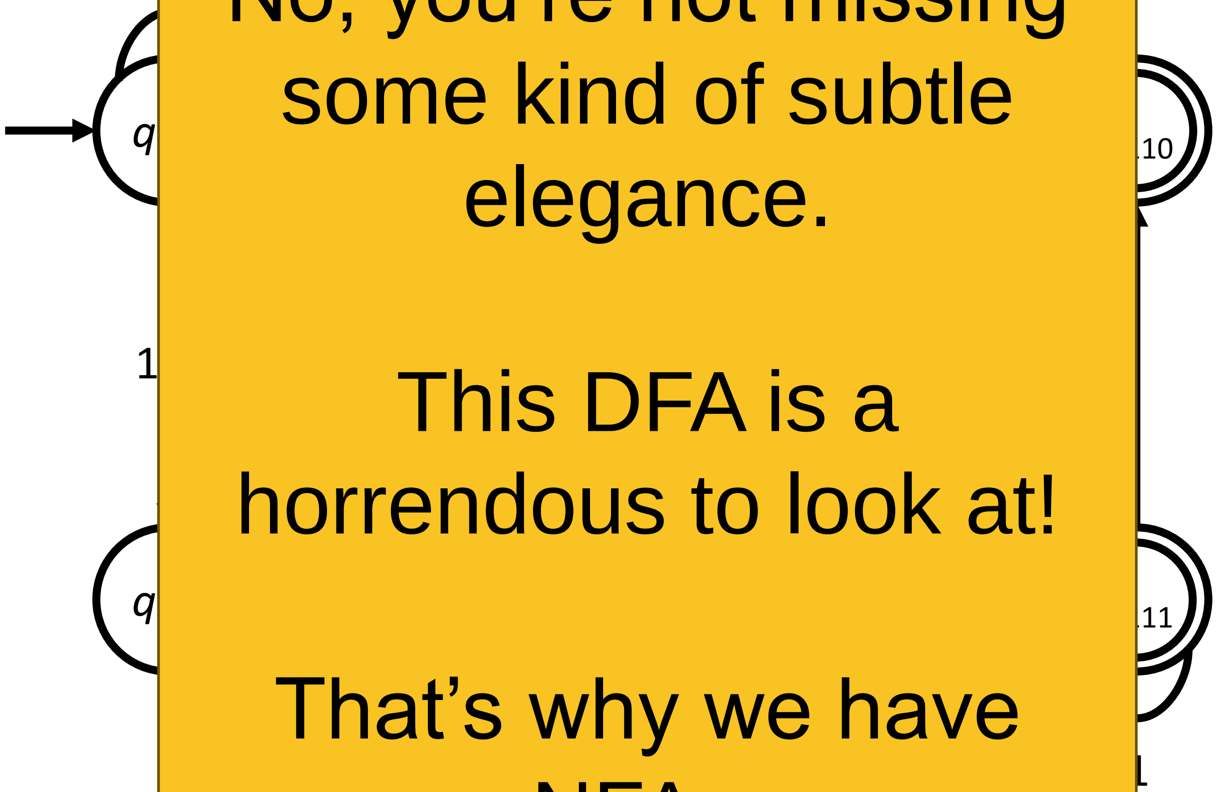
An NFA and its DFA
(Yes, this is, in fact, the best we can do.)





An NFA and its DFA
(Yes, this is, in fact, the best we can do.)





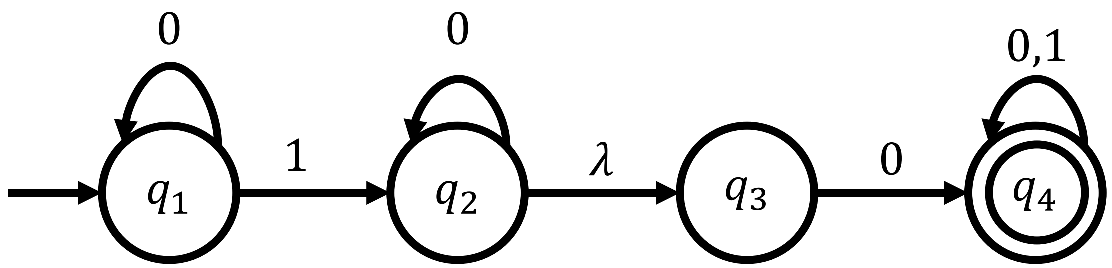
No, you’re not missing some kind of subtle elegance.

This DFA is a horrendous to look at!

That’s why we have NFAs.

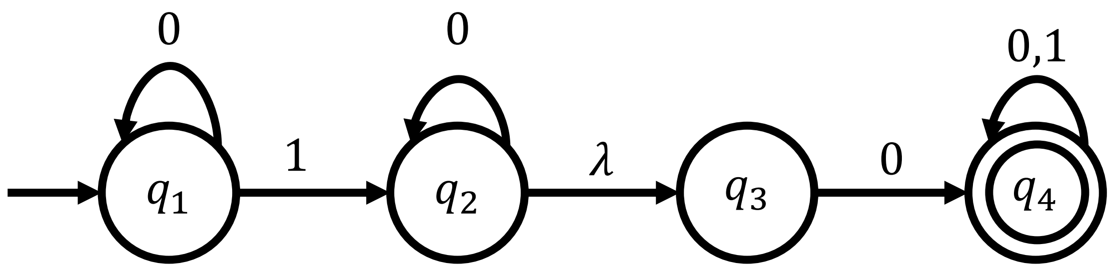
Definition: Nondeterministic Finite Automaton

- A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ that consists of:
  - Q — A finite set of *states*
  - Σ — An *alphabet*
  - $\delta: Q \times \Sigma \rightarrow P(Q)$ — A *transition function*
  - $q_0 \in Q$ — A *start state*
  - $F \subseteq Q$ — A set of *accept* (or *final*) states



Question

You might be wondering what would happen if a string was fed to the NFA and a state does not have a definition for it. For example, what happens if the string “110” was fed to the above NFA?



Question

You might be wondering what would happen if a string was fed to the NFA and a state does not have a definition for it. For example, what happens if the string “110” was fed to the above NFA?

Since $\delta(q_2, 1)$ and $\delta(q_3, 1)$ is undefined, we have a situation called **dead configuration**. That means the NFA will just automatically stop without further action.

Question

What can an NFA do that a DFA can't?

Question

What can an NFA do that a DFA can’t?

Absolutely Nothing!

Let’s do some problems with NFAs!

To the board!

The Equivalence of NFAs and DFAs

Equivalence

- To prove NFAs and DFAs equivalent, it suffices to show that:
  - For every DFA, an NFA can be made that recognizes exactly the same language.
    - **This is easy.**
      - The capabilities of NFAs are a strict superset of those of DFAs, so every DFA is already an NFA. There's no proof to write.
  - For every NFA, a DFA can be made that recognizes exactly the same language.
    - **This isn't.** So let's do it.

NFA-to-DFA Conversion: Simulating the NFA

- Keep in mind our three observations about the computation process of an NFA:
  1. **Multiple transitions imply that at any given time, an NFA is in a *set of states***
  2. **Empty transitions are handled in computation by *including their possibilities* in the set of states**
  3. **Missing transitions simply *don't add anything* to the next set of states**
- Now take this a bit further:
  - The power set P(Q) of an NFA's states is the set of all possible subsets of its states
  - So at any given time, the set of states an NFA is in is an element in P(Q)
  - P(Q) is *itself* a set
- So **on a given transition, an NFA is simply transitioning from one element of P(Q) to another**

NFA-to-DFA Conversion: Simulating the NFA

- We have also observed that:

4. **On a given transition, an NFA is simply transitioning from one element of P(*Q*) to another**

- Now consider empty and missing transitions:
  - The possibilities of empty transitions are included in the set of states by look-ahead
  - Missing transitions are handled by simply not adding anything to the set of states
  - So given the next input symbol, we account for them completely in the next set of states
- This means that **an NFA transitions from one element of P(*Q*) to another element of P(*Q*) based only on the next input symbol**

NFA-to-DFA Conversion: Simulating the NFA

- Keep in mind our three observations about the computation process of an NFA:

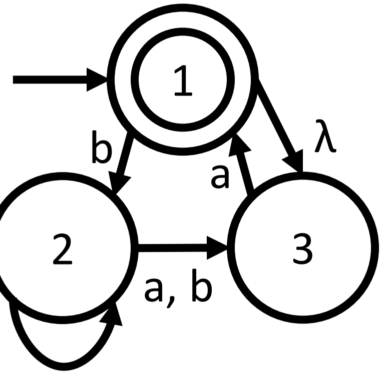
  1. **Multiple transitions imply that at any given time, an NFA is in a *set of states***
  2. **Empty transitions are handled in computation by *including their possibilities* in the set of states**
  3. **Missing transitions simply *don't add anything* to the next set of states**

- We have also observed that:

  4. **On a given transition, an NFA is simply transitioning from one element of P(*Q*) to another**
  5. **An NFA transitions from one element of P(*Q*) to another element of P(*Q*) based only on the next input symbol**

- So while an NFA is computing, we have:
  - A finite set of states it can be in, and
  - A way to know which state it will be in next, given only its current state and the input symbol

Building the DFA

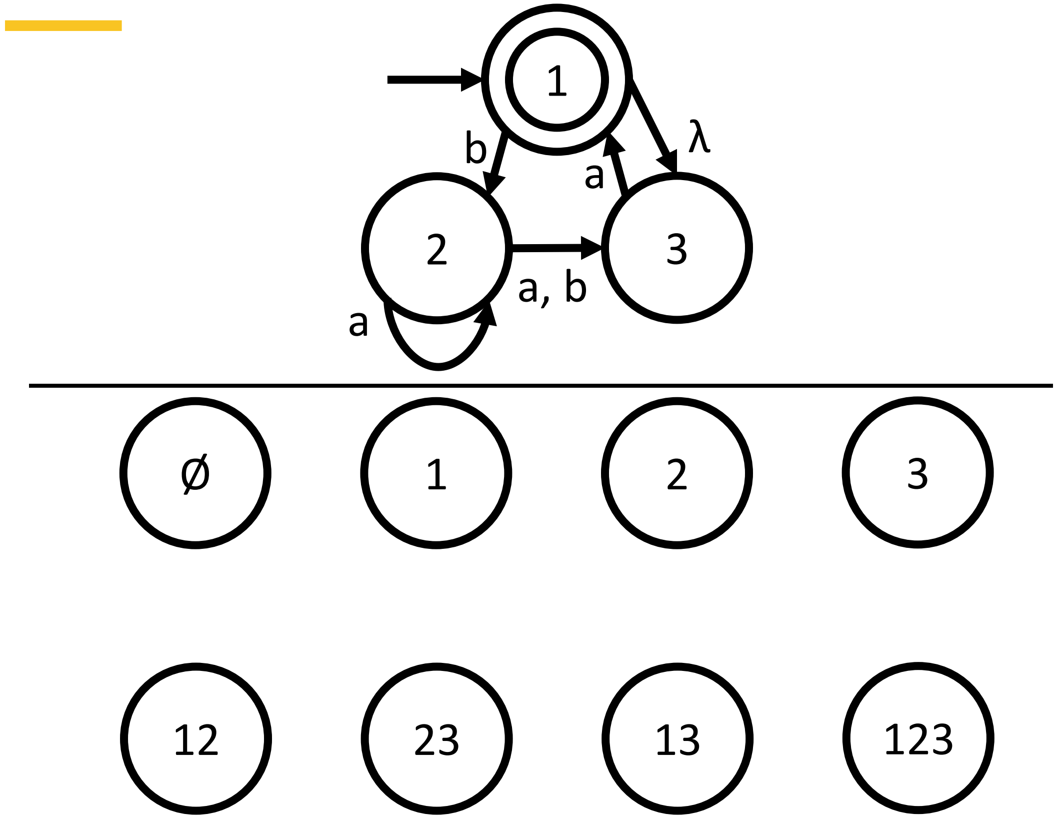


We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

We need to figure out:

- The state set
- The transition function
- The start state
- The final states

Building the DFA: States

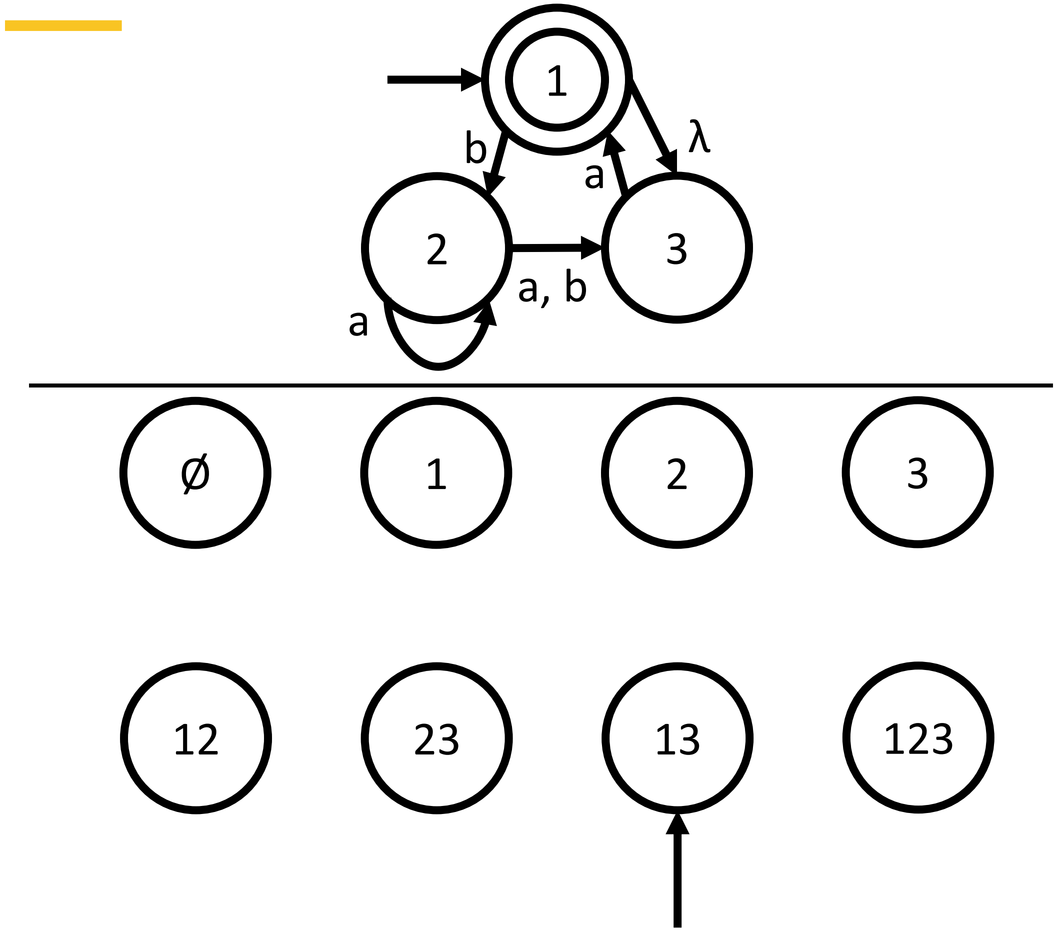


We want to build a DFA $D = \{\, Q_D, \Sigma, \delta_D, q_{0D}, F_D \,\}$ that simulates NFA $N = \{\, Q_N, \Sigma, \delta_N, q_{0N}, F_N \,\}$

The state set is the easiest part: just remember that we need to simulate being in some subset of the states of N, and say:

- **Q_D = P(Q_N), the power set of Q_N**

Building the DFA: Starting



We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**

Next let's look at the start state

- Easy answer: the state corresponding to being in, and only in, the start state of the NFA
- …with one wrinkle: empty-string transitions

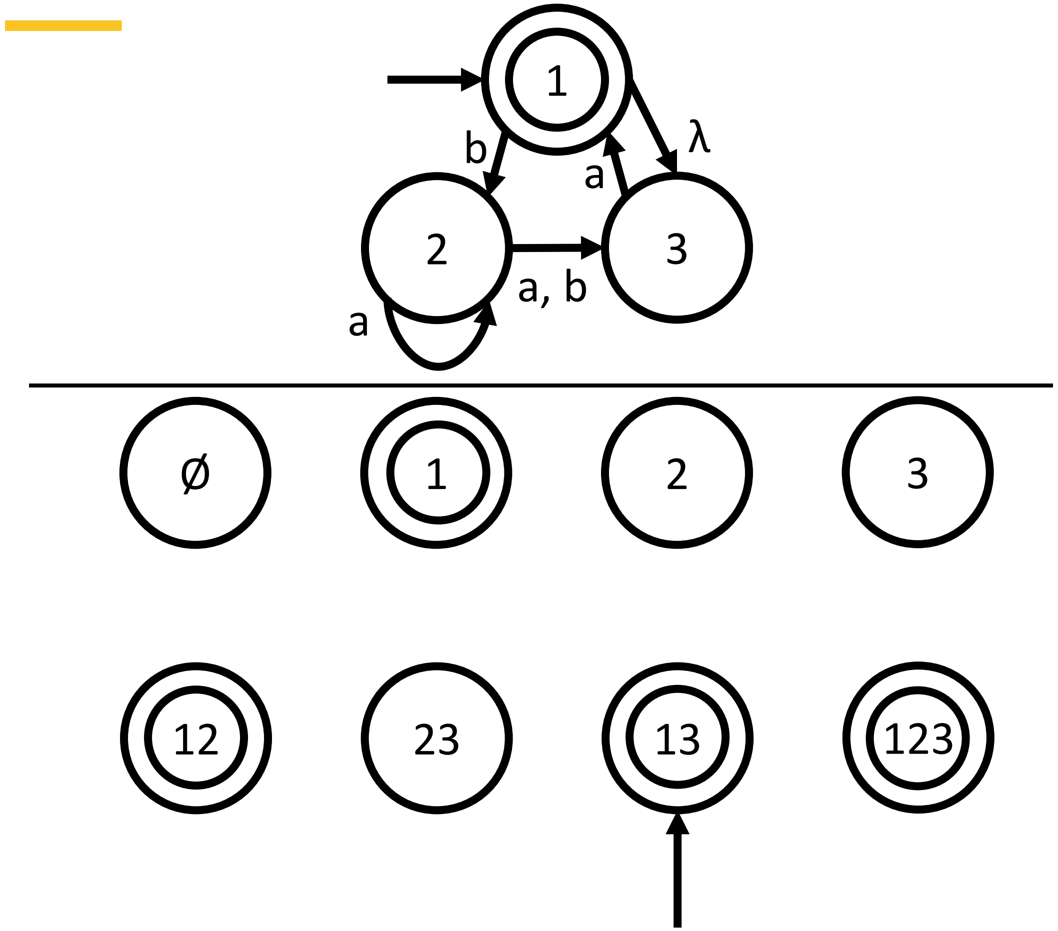
So we need the set-state containing:

- q_{0N}
- The states you can reach from q_{0N} with only empty-string transitions

Let's call that set-state E({q_{0N}}), and say:

- **q_{0D} = E({q_{0N}})**

Building the DFA: Acceptance



We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**
- **q_{0D} = E({q_{0N}})**

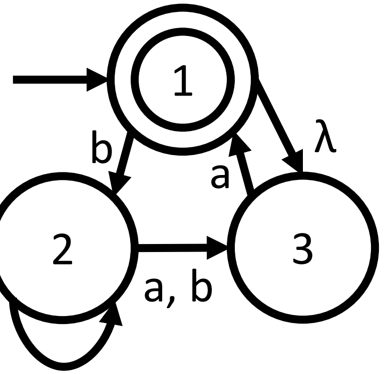
Next the accept states – which actually *are* easy

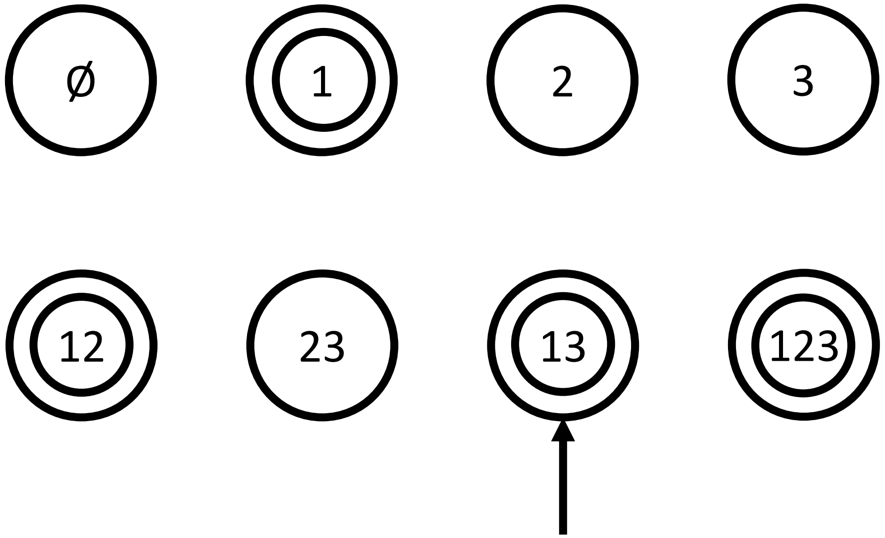
- Recall that the NFA accepts if it has *any* computation path to an accept state
- This means that in our computation, if there is any state we could be in that is an NFA accept state, we accept

So a state-set accepts if it *contains any accept state* from the NFA

- $F_D = \{R \in Q_D \mid R$ and F_N have a common member$\}$, or
- $\boldsymbol{F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}}$

Building the DFA: Transition





We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**
- **q_{0D} = E($\{q_{0N}\}$)**
- **$F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$**

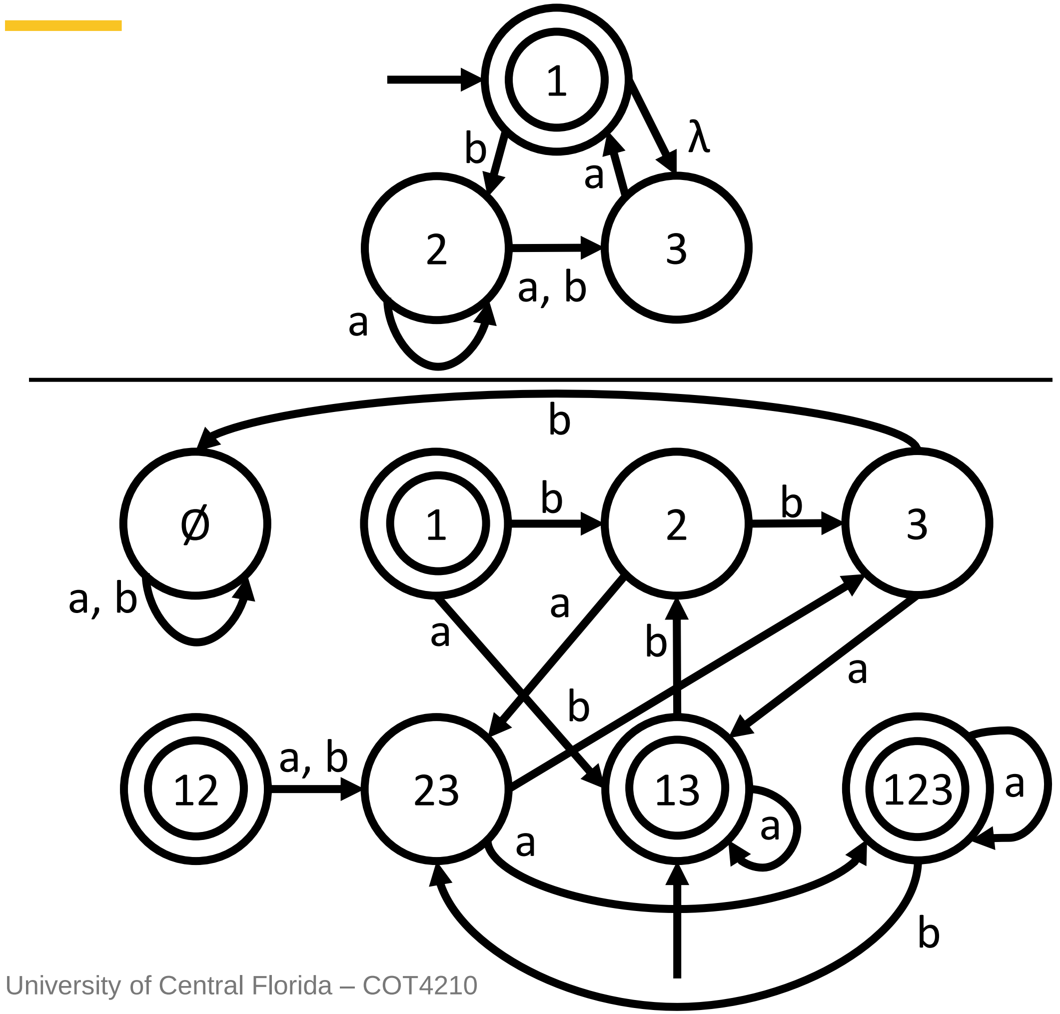
Now the transition function. Remember:

- The NFA transitions between *sets of states*
- We simulate that by having a state for each possible set

So to transition on a given symbol *a*, we:

- Look at *all* the NFA states we are currently simulating
- Look at all the states *they* can possibly transition to on *a*
- Transition to the set of all of those states

Building the DFA: Transition



We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**
- **q_{0D} = E({q_{0N}})**
- **$F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$**

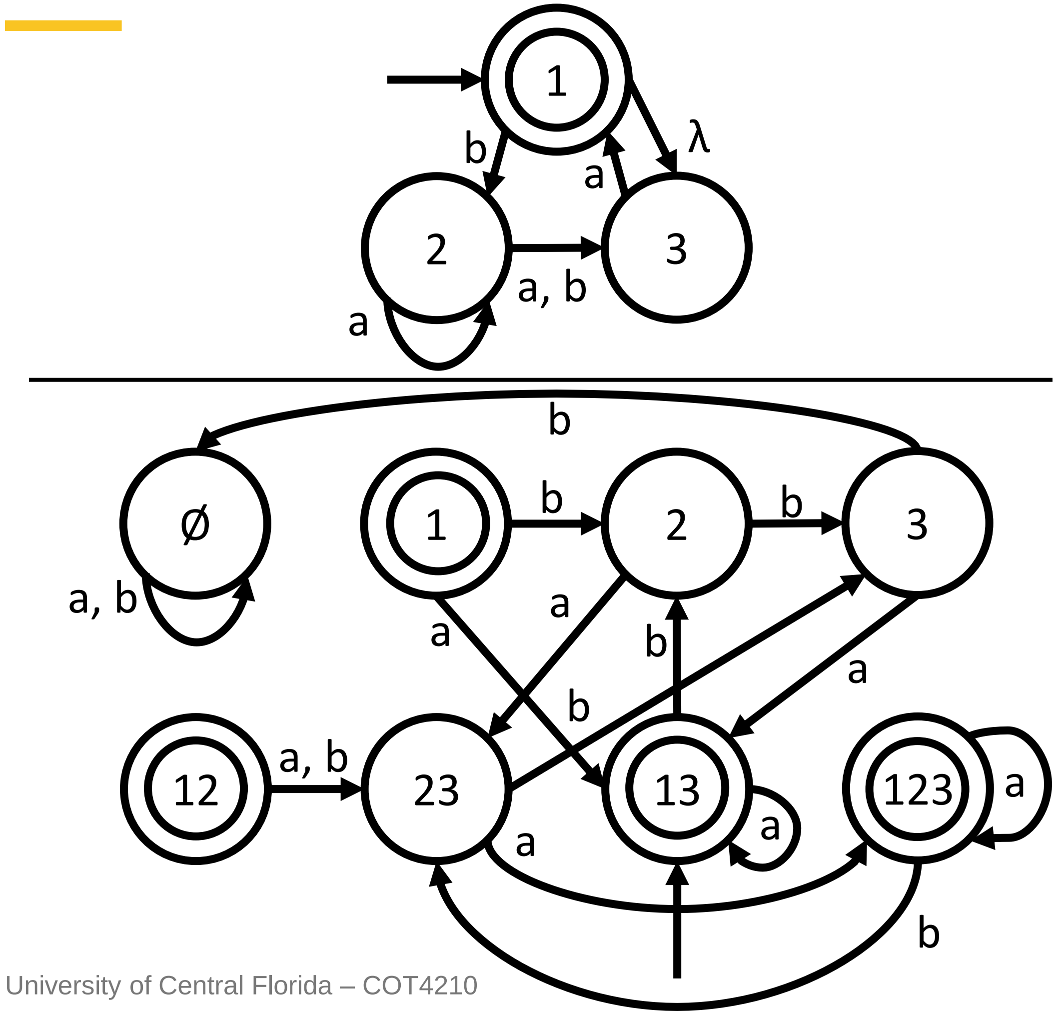
To transition on a given symbol *a*, we:

- Look at *all* the NFA states we are currently simulating
- Look at all the states *they* can possibly transition to on *a*
- Transition to the set of all of those states

So we define $\delta_D\colon Q_D \times \Sigma \to Q_D$ as:

- $\delta_D(R, a) = \{q \in Q_N \mid q \in \delta_N(r, a) \text{ for some } r \in R\}$
- …almost

Building the DFA: Transition



We want to build a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**
- **q_{0D} = E($\{q_{0N}\}$)**
- **$F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$**

We just defined $\delta_D\colon\ Q_D \times \Sigma \rightarrow Q_D$ as:

- $\delta_D(R, a) = \{q \in Q_N \mid q \in \delta_N(r, a) \text{ for some } r \in R\}$

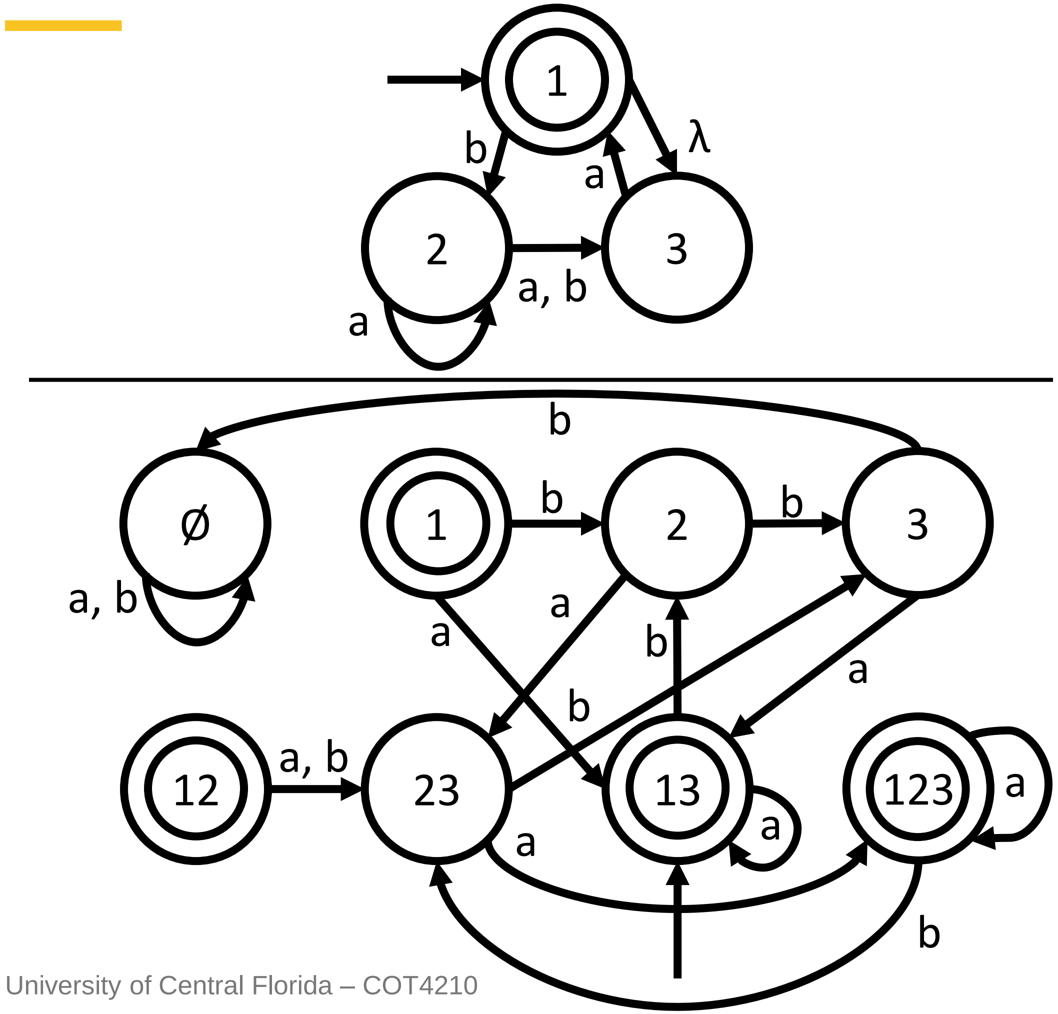
But we need to consider empty string transitions

- We can just do this the same way we did with the start state

So we finally say:

- $\boldsymbol{\delta_D(R, a) = \{q \in Q_N \mid q \in E(\delta_N(r, a)) \text{ for some } r \in R\}}$

Building the DFA: Transition



We **have built** a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

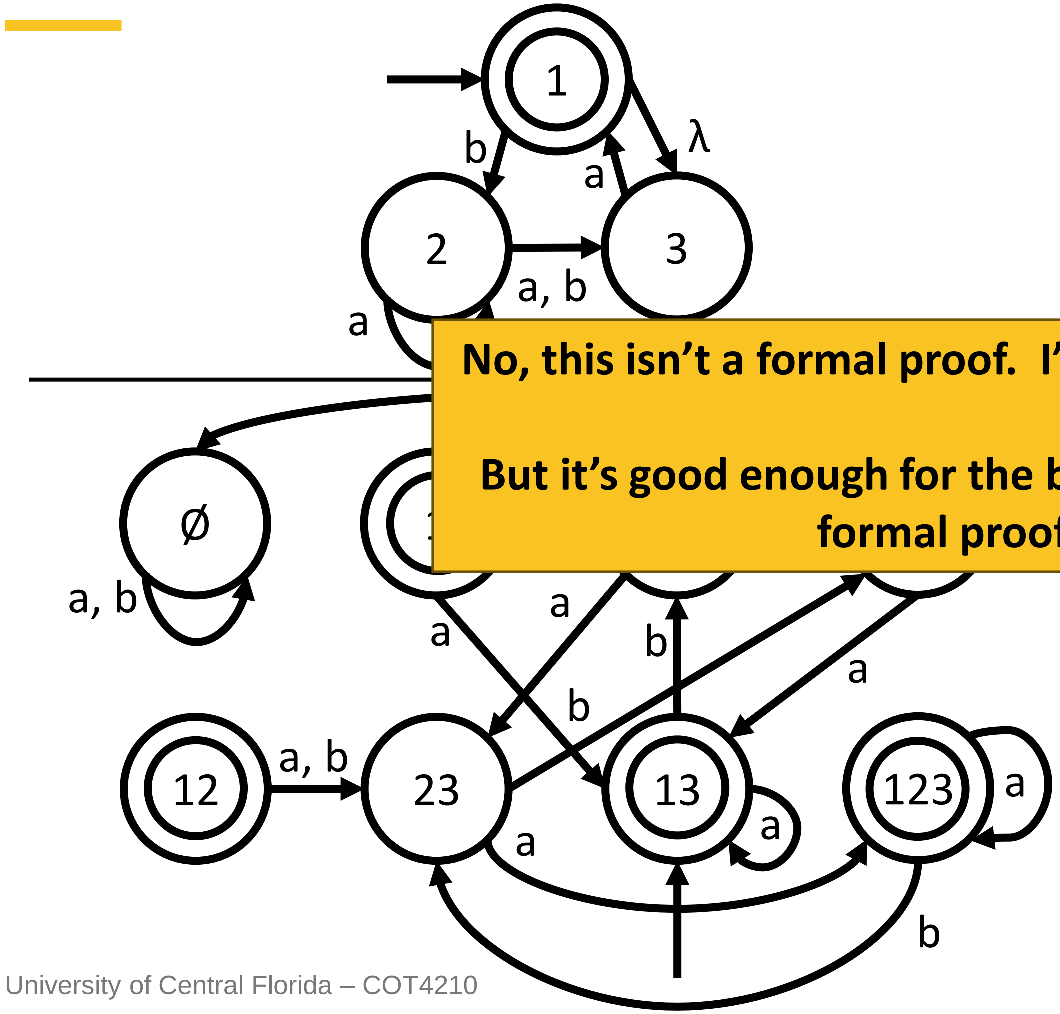
- **Q_D = P(Q_N), the power set of Q_N**
- **$q_{0D} = E(\{q_{0N}\})$**
- **$F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$**
- $\boldsymbol{\delta_D(R, a) = \{q \in Q_N \mid q \in E(\delta_N(r, a)) \text{ for some } r \in R\}}$

From our observations during construction, *D* is always in a state corresponding to the subset of states *N* could be in on the same input

We have observed that for any NFA *N*, a corresponding DFA *D* exists that recognizes the same language as *N*

Hence, by definition of regular languages, any language recognized by an NFA is regular.

Building the DFA: Transition



No, this isn't a formal proof. I'm not going to draw the little square.

But it's good enough for the book, let alone this class, and doing a formal proof would take weeks.

We **have built** a DFA $D = \{ Q_D, \Sigma, \delta_D, q_{0D}, F_D \}$ that simulates NFA $N = \{ Q_N, \Sigma, \delta_N, q_{0N}, F_N \}$

- **Q_D = P(Q_N), the power set of Q_N**
- **q_{0D} = E({q_{0N}})**
- }

$\delta_N(r, a))$ **for some** $r \in R\}$

truction, *D* is always in a state corresponding to the subset of states *N* could be in on the same input

We have observed that for any NFA *N*, a corresponding DFA *D* exists that recognizes the same language as *N*

Hence, by definition of regular languages, any language recognized by an NFA is regular.

The Consequences of Equivalence

Consequences 1

If every NFA has a DFA, then every language that can be recognized with an NFA can be recognized with a DFA.

But that means…

Consequences 1

If every NFA has a DFA, then every language that can be recognized with an NFA can be recognized with a DFA.

But that means…

Corollary to NFA/DFA Equivalence: *A language is regular if and only if it can be recognized by an NFA.*

Consequences 2

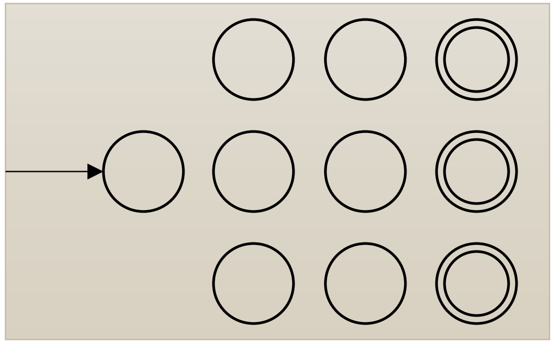
Not too long ago we proved that the class of regular languages was closed under union.

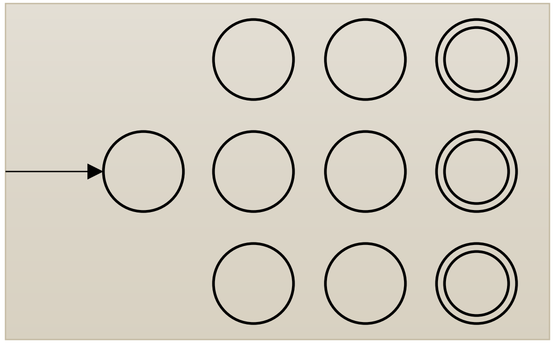
It hurt. A lot.

Let's do that again, a lot faster…

- ..and after that, we'll prove that it's closed under concatenation and star closure

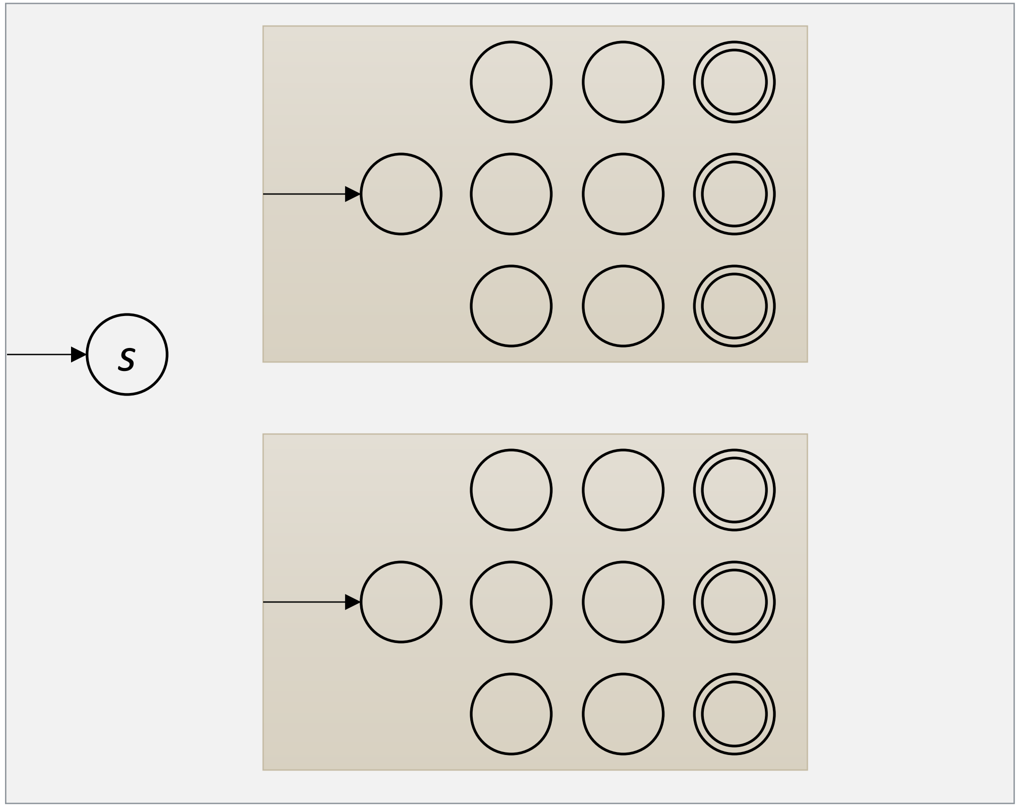
Proof: Closure of Regular Languages – Union





Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages A and B.

Proof: Closure of Regular Languages – Union

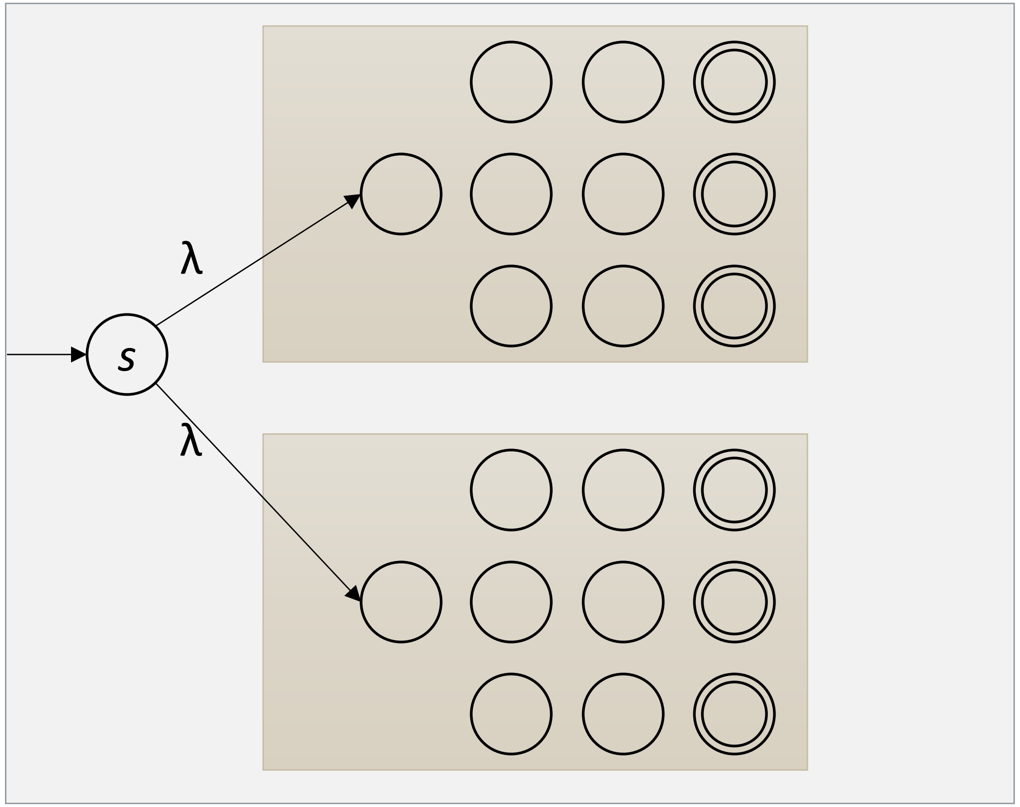


Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages A and B.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B \cup \{ s \}$
- Start state s
- $F = F_A \cup F_B$

Proof: Closure of Regular Languages – Union

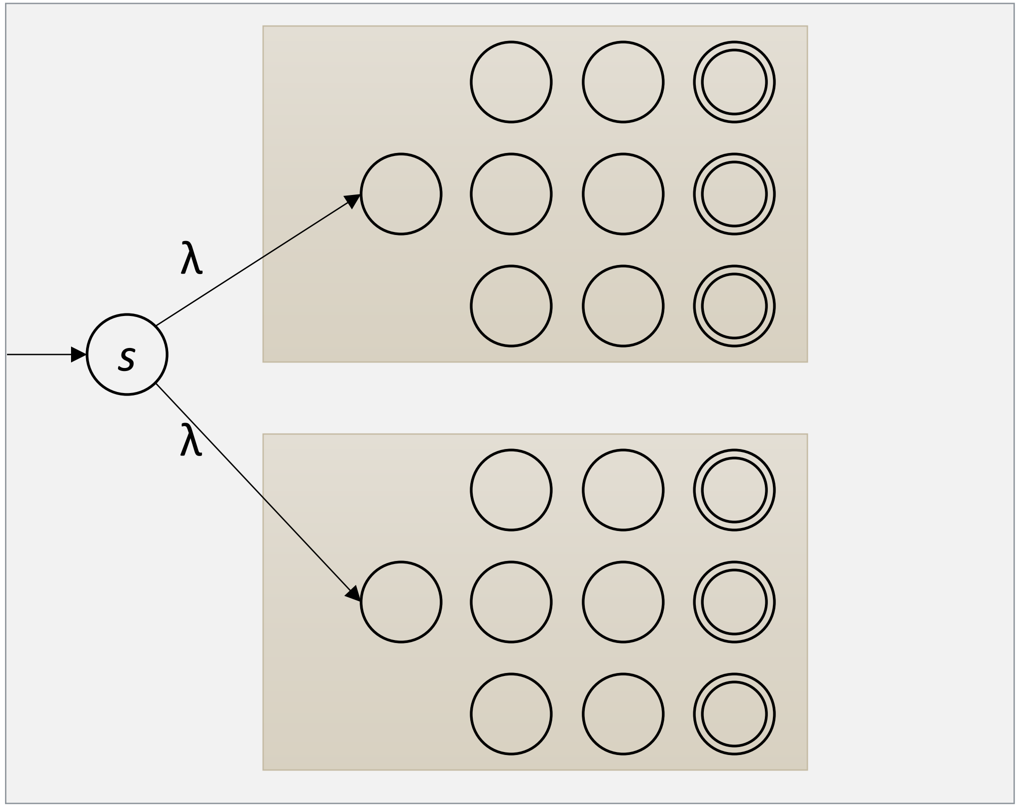


Let $N_A = \{\, Q_A, \Sigma, \delta_A, q_{0A}, F_A \,\}$ and $N_B = \{\, Q_B, \Sigma, \delta_B, q_{0B}, F_B \,\}$ be NFAs recognizing regular languages A and B.

Construct a new NFA $N = \{\, Q, \Sigma, \delta, s, F \,\}$ with:

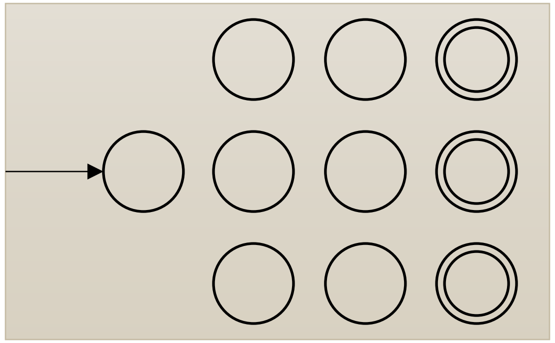
- $Q = Q_A \cup Q_B \cup \{\, s \,\}$
- Start state s
- $F = F_A \cup F_B$
- $\delta(q,a) = \begin{cases} \delta_A(q,a) & q \in Q_A \\ \delta_B(q,a) & q \in Q_B \\ \{q_{0A}, q_{0B}\} & q = s \text{ and } a = \lambda \\ \emptyset & \text{otherwise} \end{cases}$

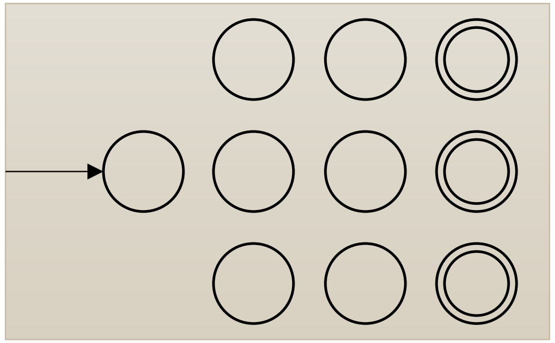
Proof: Closure of Regular Languages – Union



- N clearly accepts everything N_A or N_B accept, and nothing else.
- Therefore by recognition, N accepts everything in A or in B, and nothing else.
- Therefore by union, N accepts everything in $A \cup B$, and nothing else.
- Therefore by recognition, N recognizes $A \cup B$.
- Therefore, there is an *NFA* recognizing $A \cup B$.
- Therefore, $A \cup B$ is regular. □

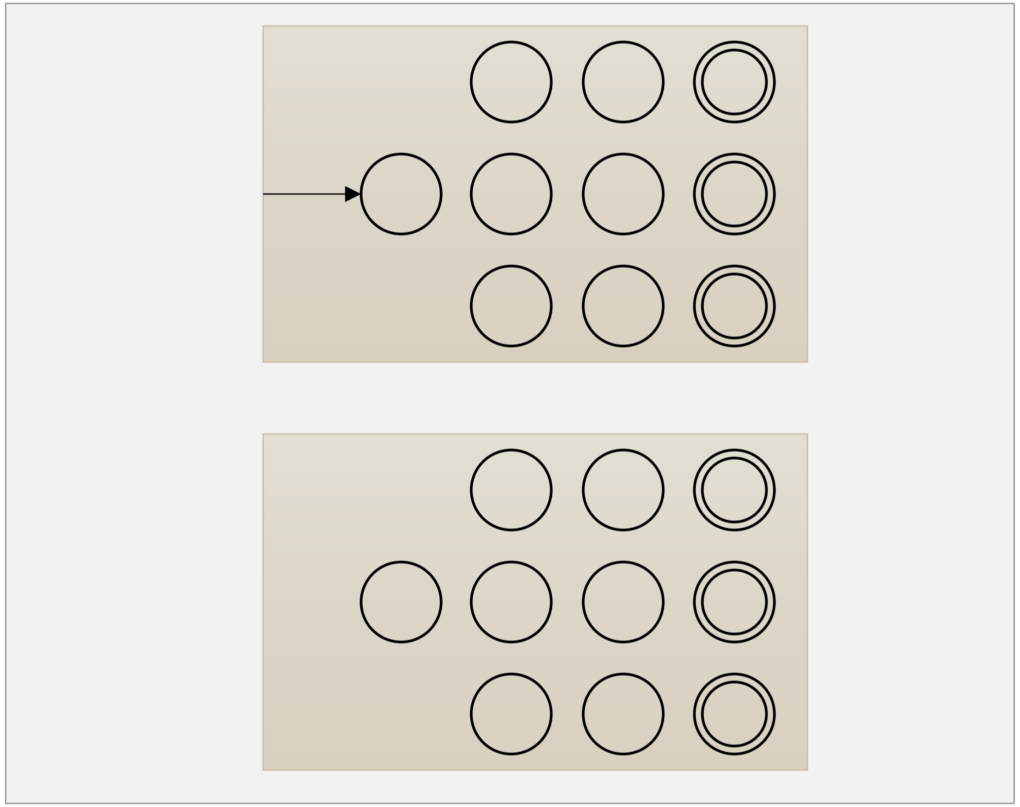
Proof: Closure of Regular Languages – Concatenation





Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages A and B.

Proof: Closure of Regular Languages – Concatenation

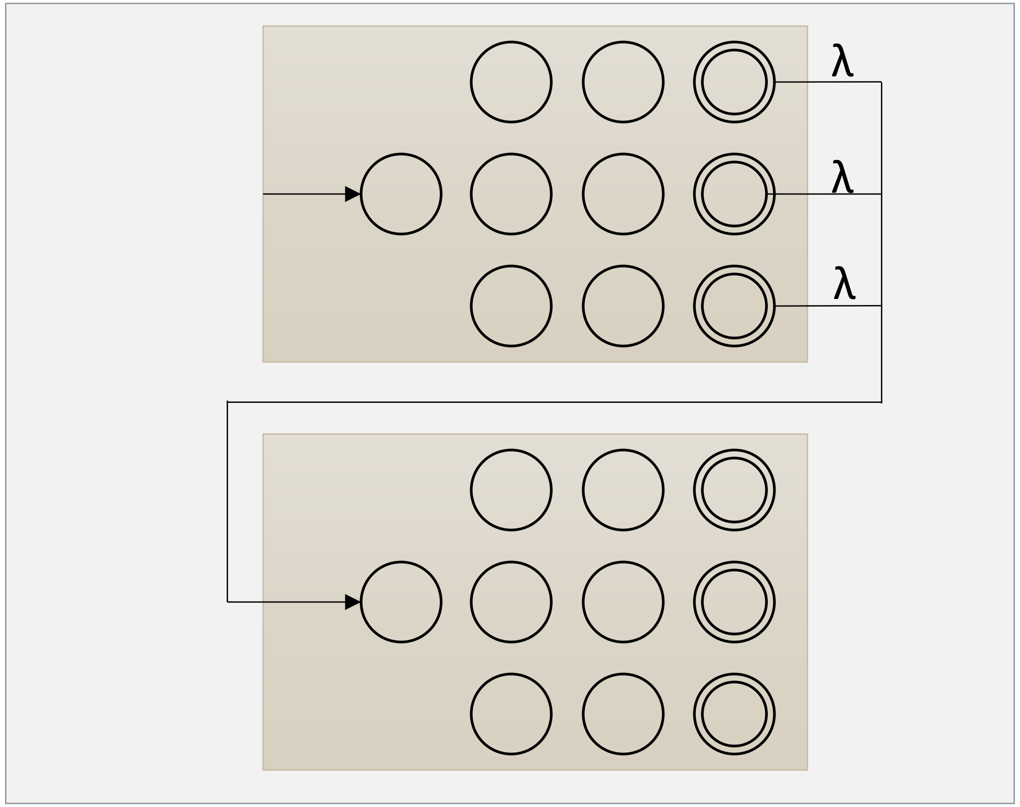


Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages A and B.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B$
- Start state q_{0A}
- $F = F_B$

Proof: Closure of Regular Languages – Concatenation

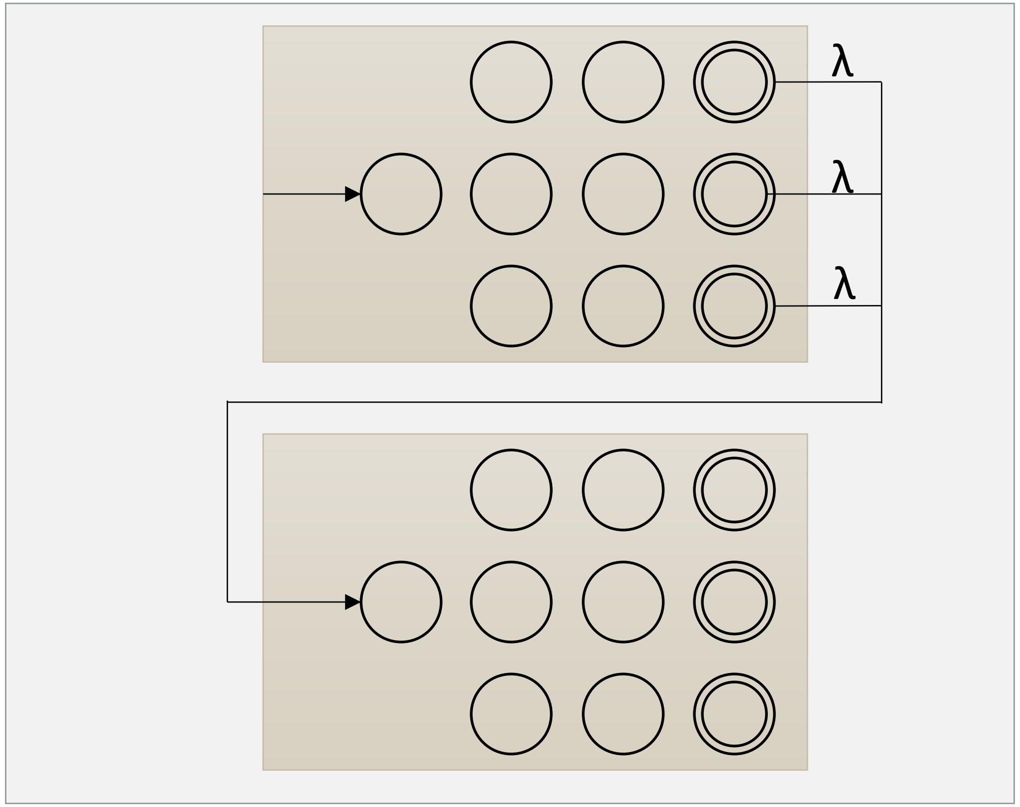


Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages A and B.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B$
- Start state q_{0A}
- $F = F_B$
- $\delta(q,a) = \begin{cases} \delta_B(q,a) & q \in Q_B \\ \delta_A(q,a) & q \in Q_A \text{ and } q \notin F_A \\ \delta_A(q,a) & q \in F_A \text{ and } a \neq \lambda \\ \delta_A(q,\varepsilon) \cup \{q_{0B}\} & q \in F_A \text{ and } a = \lambda \end{cases}$

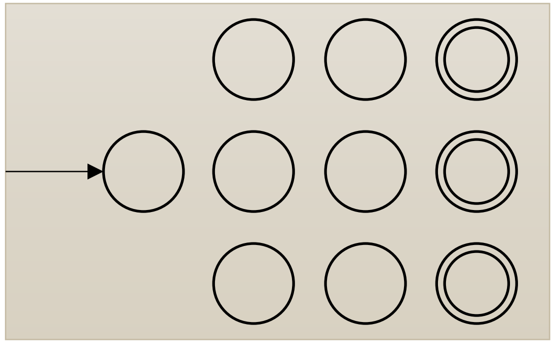
Proof: Closure of Regular Languages – Concatenation



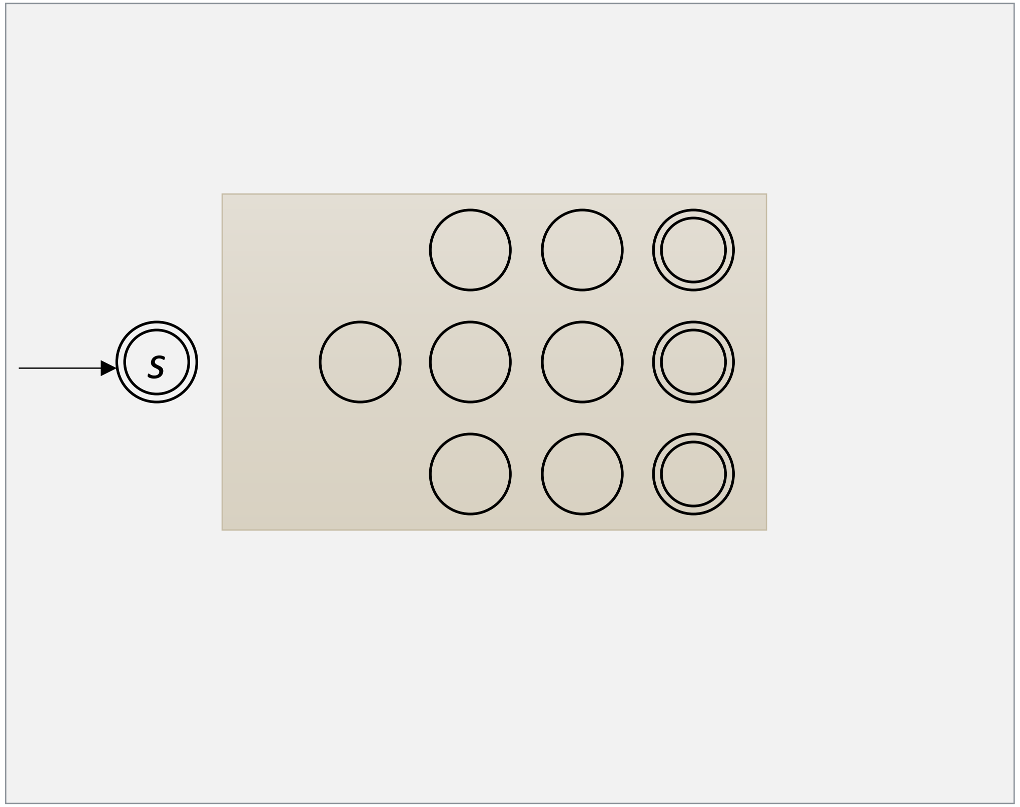
- N clearly accepts every string consisting of a string accepted by N_A followed by a string accepted by N_B, and only those strings.
- Therefore by recognition, N accepts every string that is a string in A followed by a string in B, and nothing else.
- Therefore by concatenation, N accepts every string in AB, and nothing else.
- Therefore by recognition, N recognizes AB.
- Therefore, there is an *NFA* recognizing AB.
- Therefore, AB is regular. □

Proof: Closure of Regular Languages – Star

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ be an NFA recognizing regular language A.



Proof: Closure of Regular Languages – Star

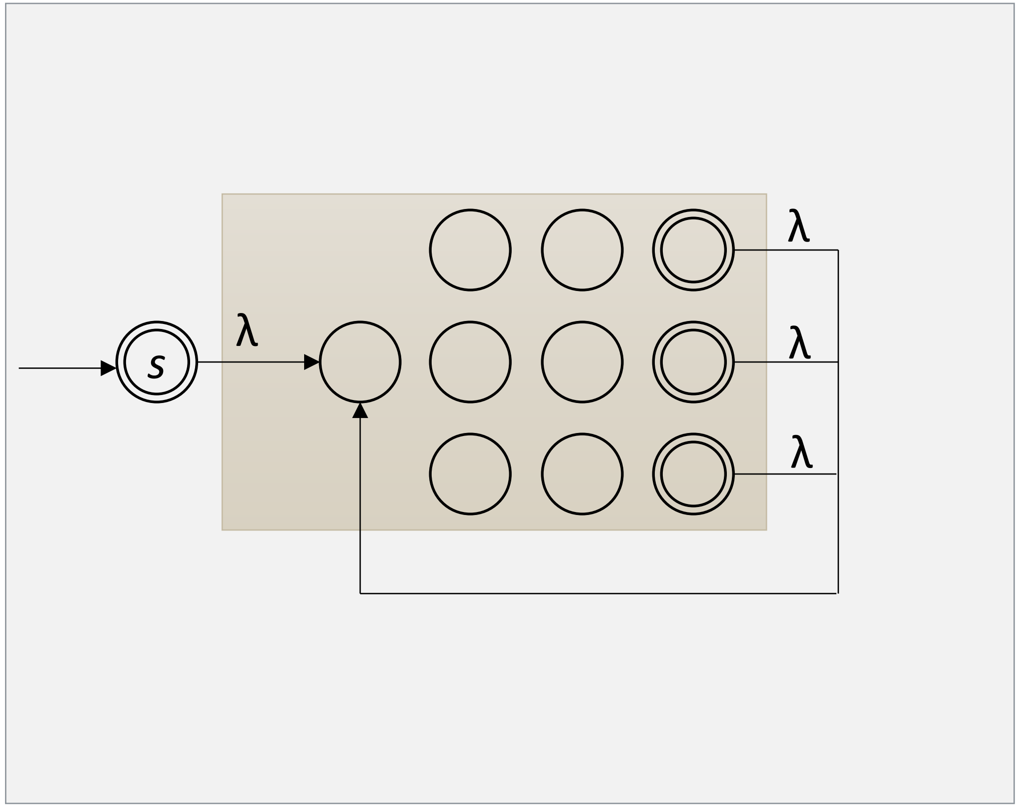


Let $N_A = \{\, Q_A, \Sigma, \delta_A, q_{0A}, F_A \,\}$ be an NFA recognizing regular language A.

Construct a new NFA $N = \{\, Q, \Sigma, \delta, s, F \,\}$ with:

- $Q = Q_A \cup \{\, s \,\}$
- Start state s
- $F = F_A \cup \{\, s \,\}$

Proof: Closure of Regular Languages – Star



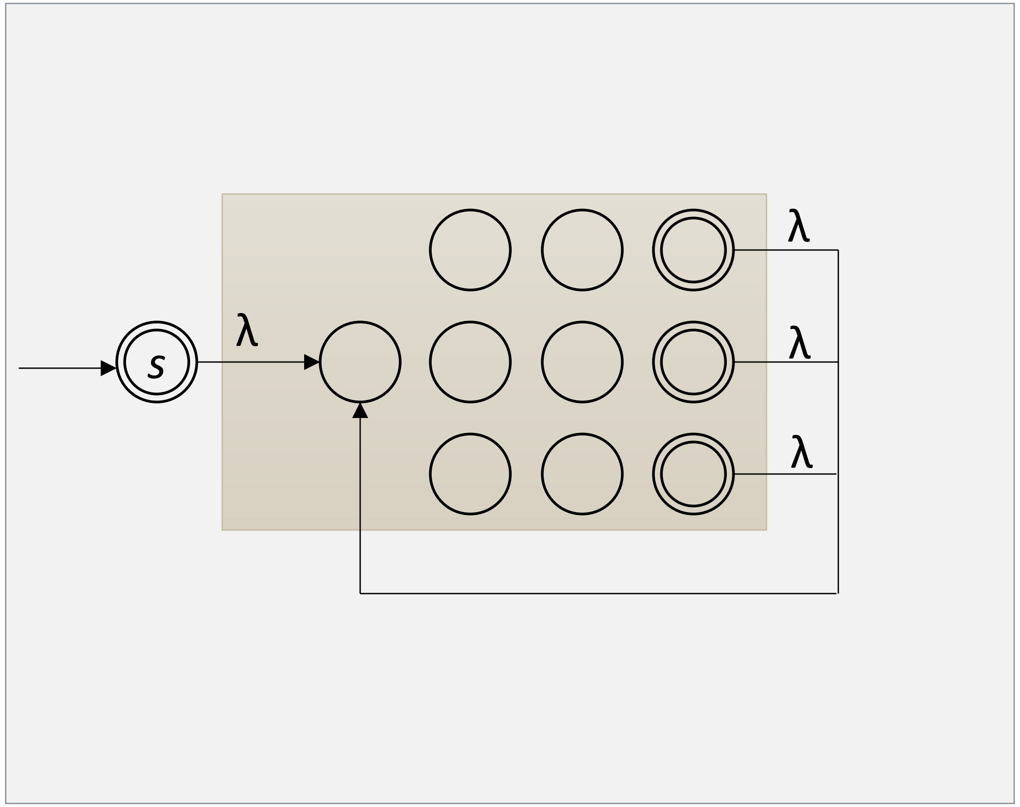
Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ be an NFA recognizing regular language A.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup \{ s \}$
- Start state s
- $F = F_A \cup \{ s \}$

- $\delta(q,a) = \begin{cases} \delta_A(q,a) & q \in Q_A \text{ and } q \notin F_A \\ \delta_A(q,a) & q \in F_A \text{ and } a \neq \lambda \\ \delta_A(q,\lambda) \cup \{q_{0A}\} & q \in F_A \text{ and } a = \lambda \\ \{q_{0A}\} & q = s \text{ and } a = \lambda \\ \emptyset & \text{otherwise} \end{cases}$

Proof: Closure of Regular Languages – Star



- *N* clearly accepts every string consisting of zero or more strings accepted by N_A.
- Therefore by recognition, *N* accepts every string consisting of zero or more strings in *A*.
- Therefore by star, *N* accepts every string in *A**.
- Therefore by recognition, *N* recognizes *A**.
- Therefore, there is an *NFA* recognizing *A**.
- Therefore, *A** is regular. □

DFA vs NFA Differences

Deterministic Finite Automata	Nondeterministic Finite Automata
Each state must have a transition for each symbol in a given alphabet Σ.	Each state is not required to have a transition for each symbol in a given alphabet Σ.
Each state can only have 1 transition for each symbol in a given alphabet Σ.	Each state can have multiples transitions for each symbol in a given alphabet Σ.
λ/ϵ transitions are NOT allowed.	λ/ϵ transitions are allowed.

Acknowledgement

Some Notes and content come from Dr. Gerber, Dr. Hughes, and Mr. Guha's COT4210 class and the Sipser Textbook, *Introduction to the Theory of Computation*, 3rd ed., 2013