CS420

Introduction to the Theory of Computation

Lecture 3: Nondeterministic Finite Automaton

Tiago Cogumbreiro

How do you generate $M_1 \cup M_2$?

What does this code tells us?

```
def union(dfa1, dfa2):
  def transition(q, a):
    return (dfa1.transition_func(q[0], a), dfa2.transition_func(q[1], a))

  def is_final(q):
    return dfa1.accepted_states(q[0]) or dfa2.accepted_states(q[1])

  return DFA(
    states = set(product(dfa1.states, dfa2.states)),
    alphabet = set(dfa1.alphabet).union(dfa2.alphabet),
    transition_func = transition,
    start_state = (dfa1.start_state, dfa2.start_state),
    accepted_states = is_final
  )
```

Mathematically...

The union operation is defined as $\mathbf{union}(M_1, M_2) = (Q_{1,2}, \Gamma_1, \delta_{1,2}, q_{1,2}, F_{1,2})$ where

- $M_1 = (Q_1, \Gamma_1, \delta_1, q_1, F_1)$
- $M_2 = (Q_2, \Gamma_2, \delta_2, q_1', F_2)$
- **States:** $Q_{1,2} = Q_1 \times Q_2$
- **Alphabet:** $\Gamma_1 = \Gamma_2$
- **Transition:** $\delta_{1,2}(q, a) = (\delta_1(q|_1, a), \delta_2(q|_2, a))$
- **Initial:** $q_{1,2} = (q_1, q_1')$
- **Final:** $F_{1,2} = \{q \mid q|_1 \in F_1 \vee q|_2 \in F_2\}$

Let notation $q|_1 = x$ be defined when $q = (x, y)$. Let notation $q|_2 = y$ be defined when $q = (x, y)$.

The key point is the transition function

- **Transition:** $\delta_{1,2}(q, a) = (\delta_1(q|_1, a), \delta_2(q|_2, a))$

How do we fill a transition table?

1. For every $q \in Q_1 \times Q_2$ and for every $a \in \Gamma_1$
2. The cell in line q and column a becomes $(\delta_1(q|_1, a), \delta_2(q|_2, a))$

How do we draw the state diagram?

1. Start from the initial state q_1 and for each $a \in \Sigma$ draw an edge to each state $q_2 = \delta(q_1, a)$
2. While there are states without outgoing edges, pick one state q_i without outgoing edges: for each $a \in \Sigma$ draw an edge to state $q_i' = \delta(q_i, a)$

	I	*O*	*H*
(undef, undef)	(undef, undef)	(undef, undef)	(undef, undef)
(undef, init)	(undef, undef)	(undef, undef)	(undef, H)
(H, H)	(HI, undef)	(undef, HO)	(undef, undef)
(HI, HO)	(undef, undef)	(undef, undef)	(undef, undef)
(H, HO)	(HI, undef)	(undef, undef)	(undef, undef)
(HI, H)	(undef, undef)	(undef, HO)	(undef, undef)
(init, undef)	(undef, undef)	(undef, undef)	(H, undef)
(H, undef)	(HI, undef)	(undef, undef)	(undef, undef)
(HI, init)	(undef, undef)	(undef, undef)	(undef, H)
(init, HO)	(undef, undef)	(undef, undef)	(H, undef)
(undef, H)	(undef, undef)	(undef, HO)	(undef, undef)
(init, H)	(undef, undef)	(undef, HO)	(H, undef)
(HI, undef)	(undef, undef)	(undef, undef)	(undef, undef)
(H, init)	(HI, undef)	(undef, undef)	(undef, H)
(init, init)	(undef, undef)	(undef, undef)	(H, H)
(undef, HO)	(undef, undef)	(undef, undef)	(undef, undef)

Today we will learn

- Non-deterministic Finite Automatons (NFA)
- Nondeterministic transitions
- Epsilon transitions
- Formalizing acceptance
- Converting from NFA to DFA

> Section 1.2

Today's lecture

- motivate, introduce NFAs informally (using state diagrams)
- define NFAs mathematically
- define NFAs algorithmically
- present the relationship between NFAs and DFAs

Exercise 1

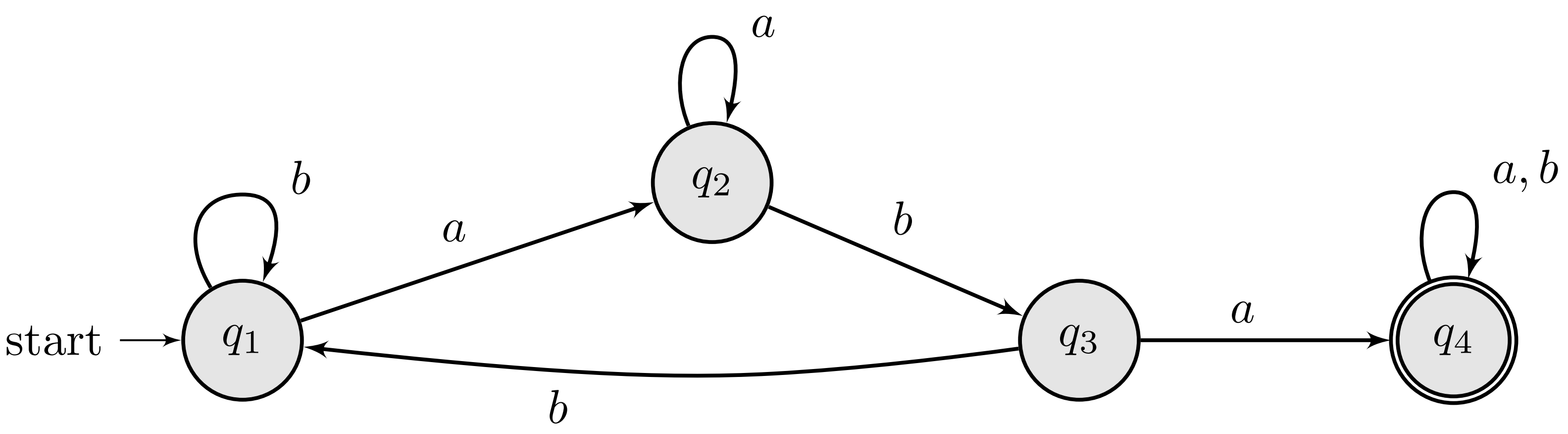
Let $\Sigma = \{a, b\}$. Give a DFA with **four** states that recognizes the following language

$$\{w \mid w \text{ contains the string } aba\}$$

Exercise 1

Let $\Sigma = \{a, b\}$. Give a DFA with **four** states that recognizes the following language

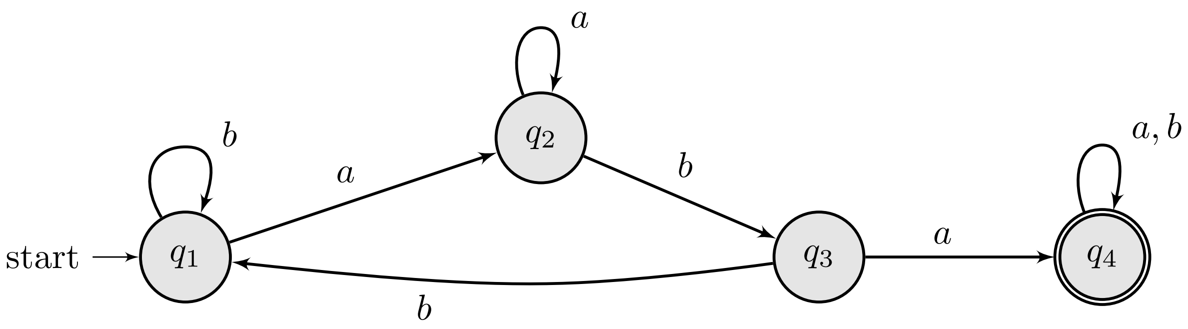
$$\{w \mid w \text{ contains the string } aba\}$$



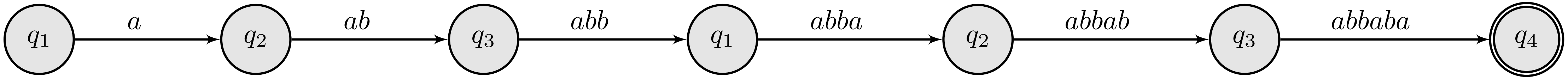
Acceptance in a DFA

Acceptance is path finding

The given string must follow a path from the starting node into an accepting node.



Acceptance of abbaba



DFA summary

- simple to analyze because each transition is deterministic
- implementing a DFA is quite trivial (because of the above)
- not very intuitive to design, because they are also limited
- each states must have a transition for all inputs (verbosity)
- using sink states to represent inputs we want to reject (verbosity)

Introducing Nondeterminstic Finite Automata (NFA)

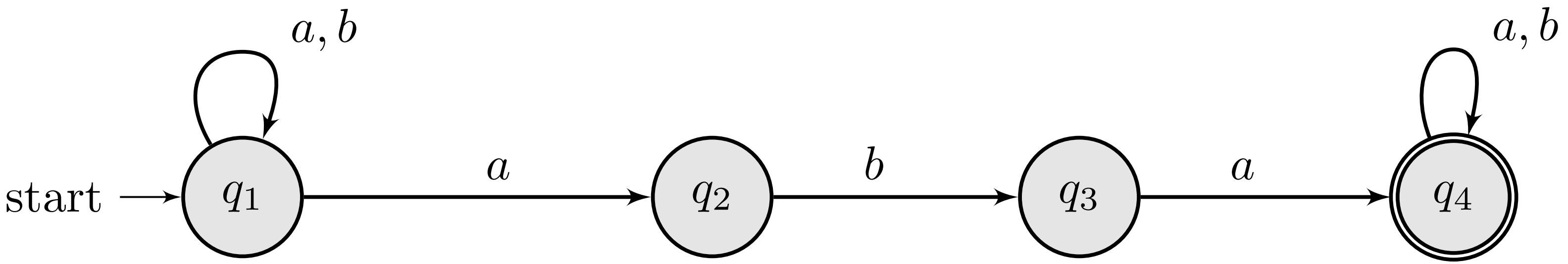
Introducing NFAs

- harder to analyze due to nondeterminism
- harder to implement (because of the above)
- may be more intuitive to design
- states may omit transitions they do not care
- sink states are unneeded

Exercise 1 with an NFA

Let $\Sigma = \{a, b\}$. Give an NFA with **four states** that recognizes the following language

$$\{w \mid w \text{ contains the string } aba\}$$

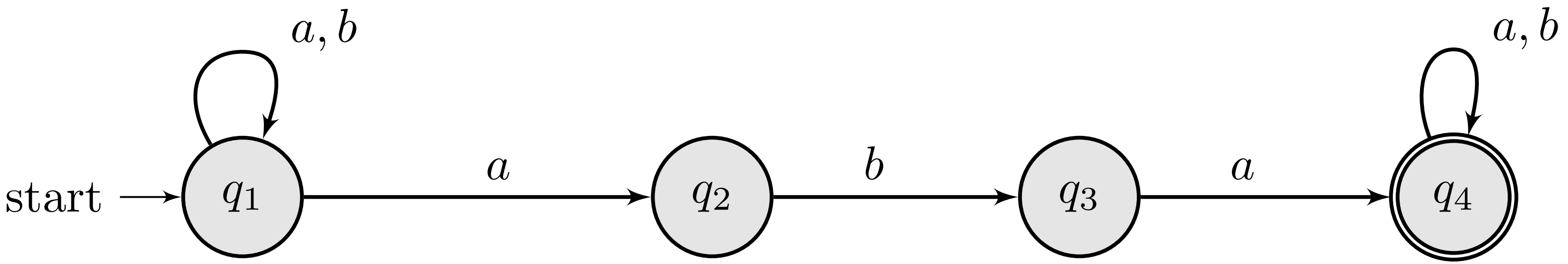


State diagram differences versus DFA?

Exercise 1 with an NFA

Let $\Sigma = \{a, b\}$. Give an NFA with **four states** that recognizes the following language

$$\{w \mid w \text{ contains the string } aba\}$$



State diagram differences versus DFA?

- **Nondeterminism:** q_1 may transition via a to q_1 and also to q_2
  $q_1 \xrightarrow{a} q_1$ and $q_1 \xrightarrow{a} q_2$
- **Absent transitions:** state q_2 is missing an outgoing edge labelled by a!
  $q_2 \xrightarrow{b} q_3$ and $q_2 \not\xrightarrow{a}$

Acceptance in an NFA

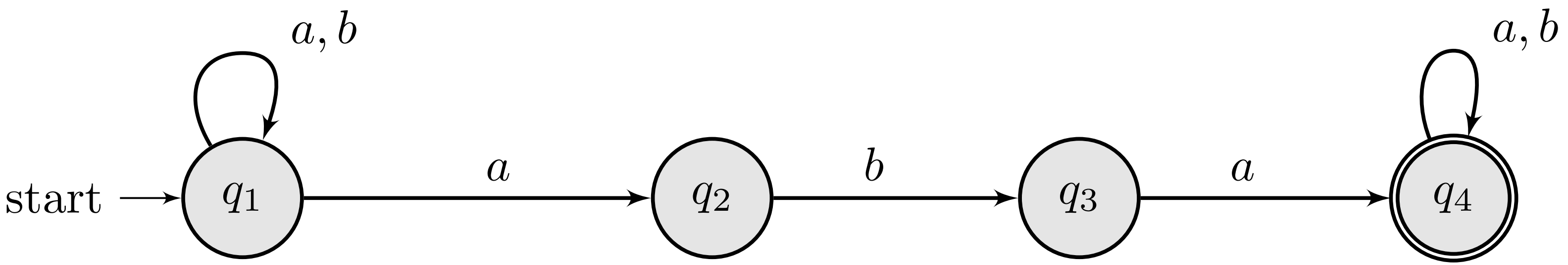
Acceptance is path finding

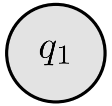
The given string must be a path from the starting node into the accepting node.

> NFAs can have **multiple** possible paths because of nondeterminism, contrary to DFAs!

Acceptance in an NFA

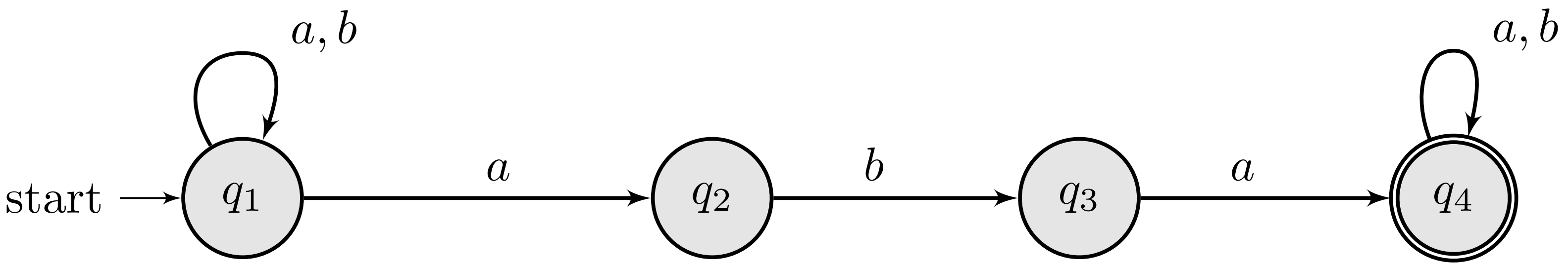
Acceptance of **a**bbaba

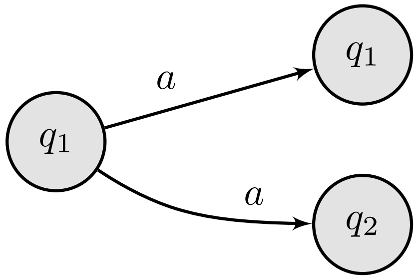




Acceptance in an NFA

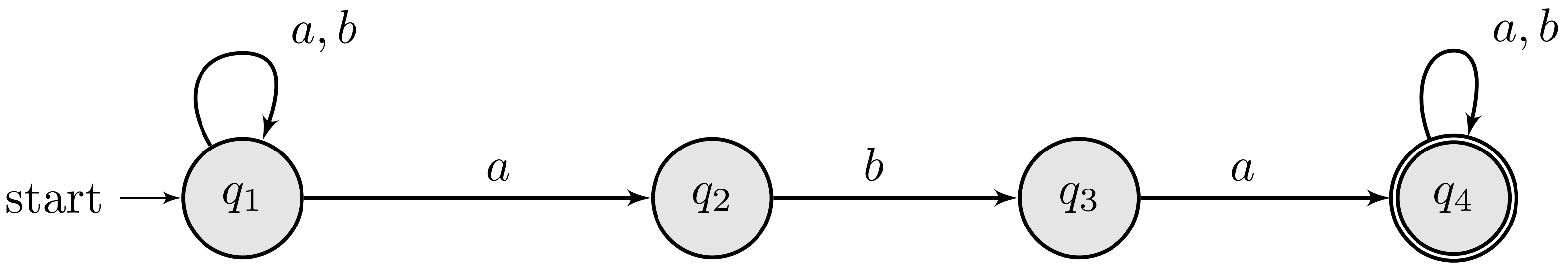
Acceptance of a**b**baba

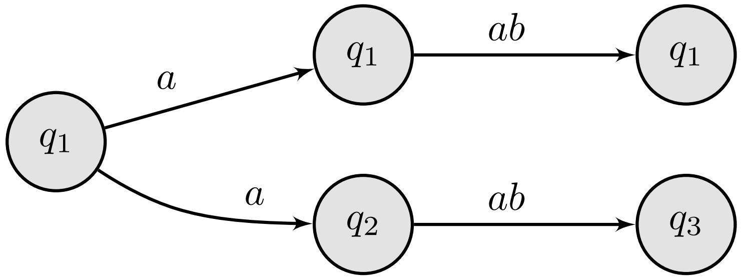




Acceptance in an NFA

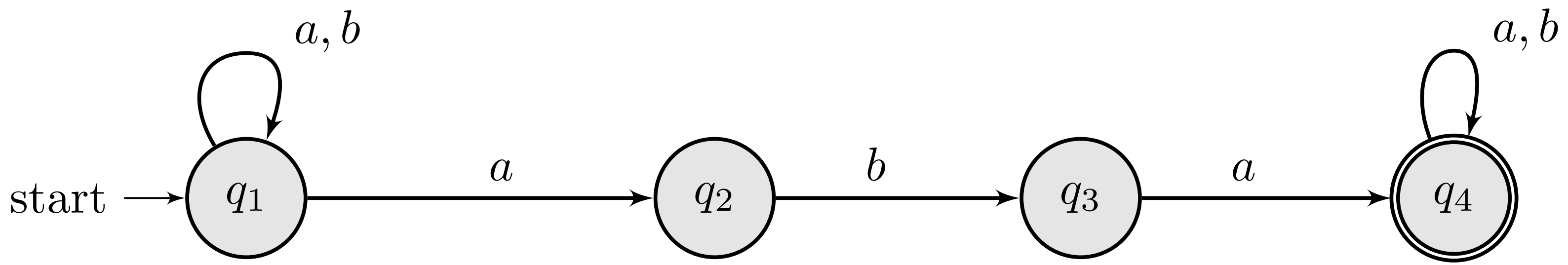
Acceptance of ab**b**aba

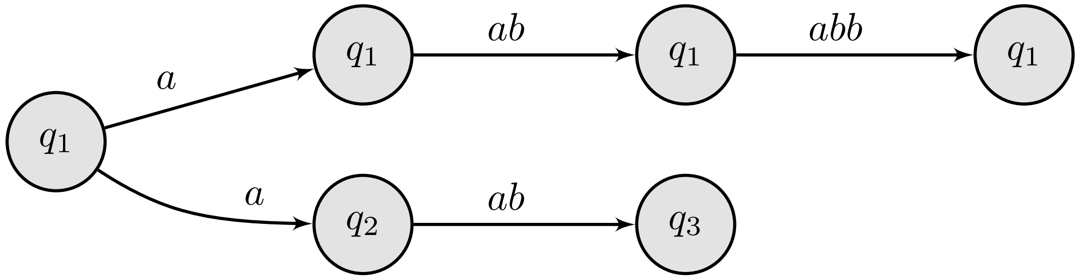




Acceptance in an NFA

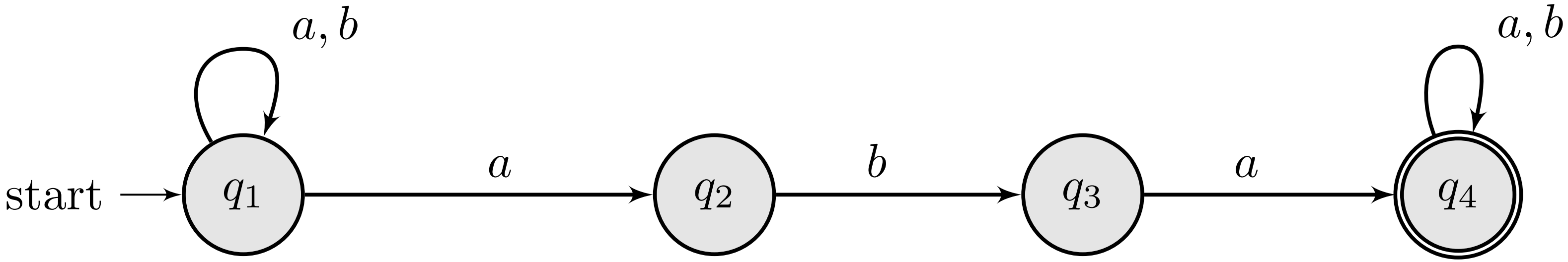
Acceptance of abb**a**ba

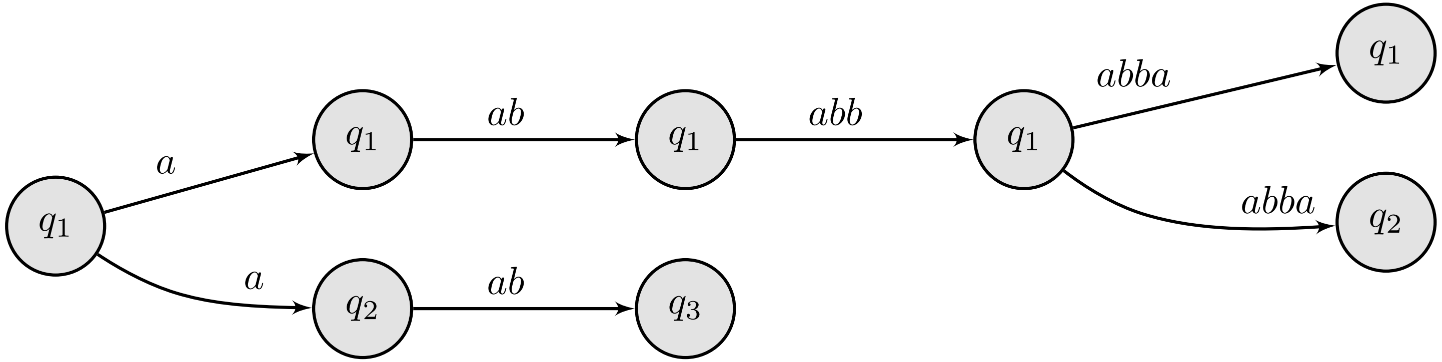




Acceptance in an NFA

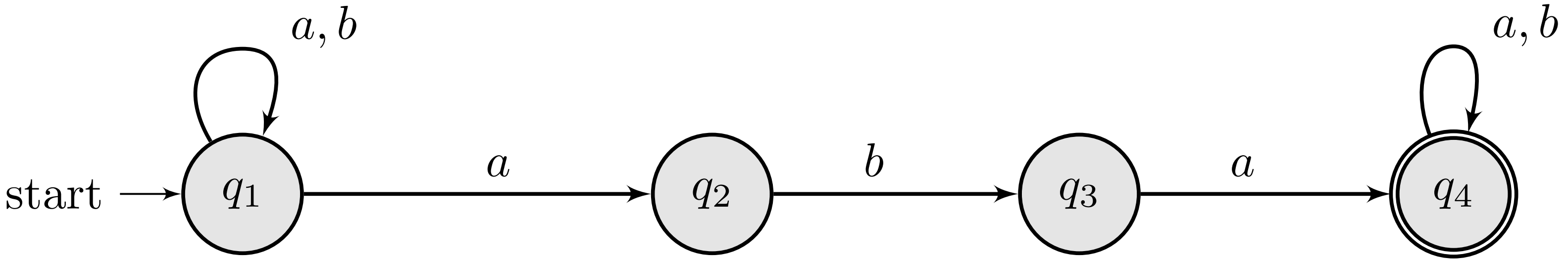
Acceptance of abba**b**a

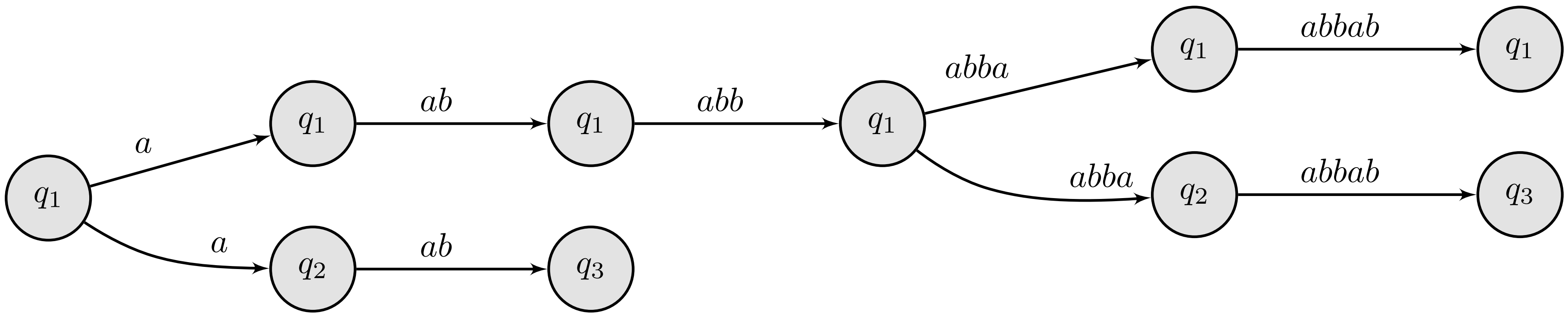




Acceptance in an NFA

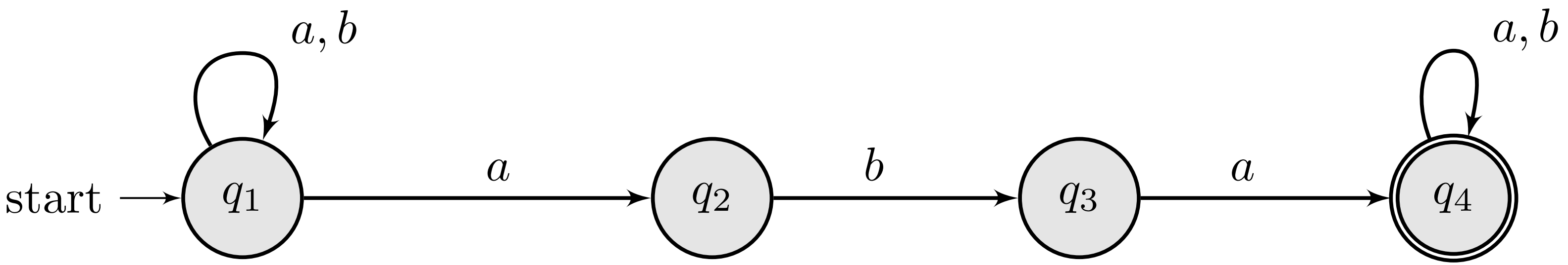
Acceptance of abbab**a**

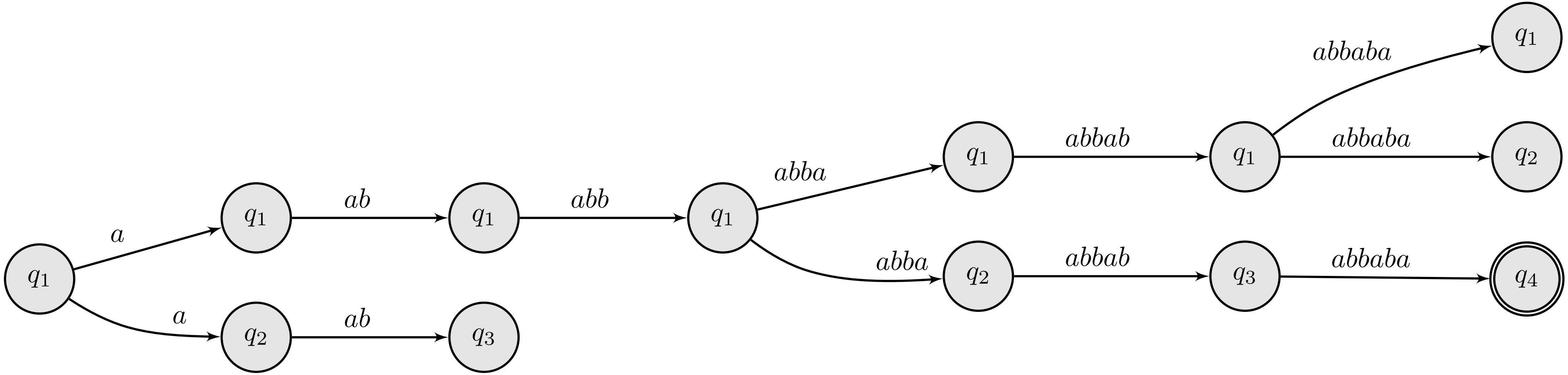




Acceptance in an NFA

Acceptance of abbaba





Acceptance in an NFA

- There are multiple concurrent possible paths and a current state
- Given a current state, if there are no transitions for a given input, the path ends
- Once we reach the final path, we check if the there are accepting states

Exercise 2

Exercise 2

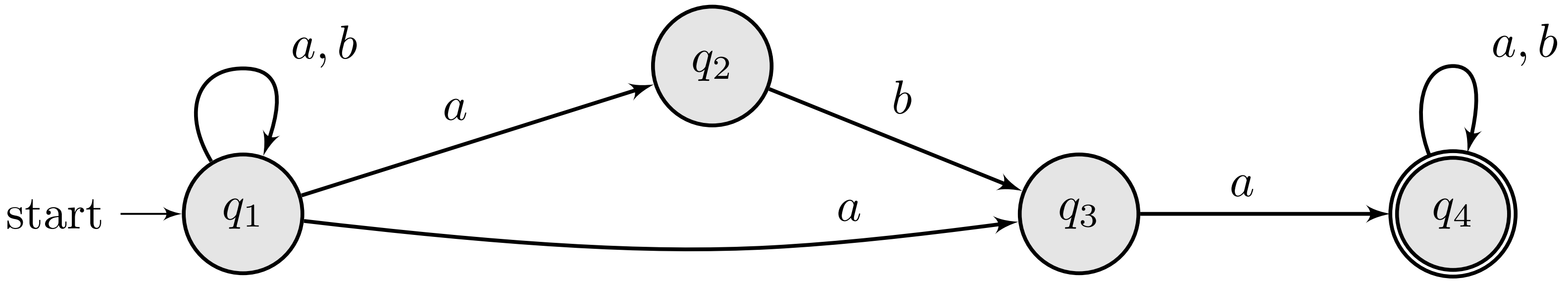
Let $\Sigma = \{a, b\}$. Give an NFA with **four states** that recognizes the following language

$$\{w \mid w \text{ contains the strings } aba \text{ or } aa\}$$

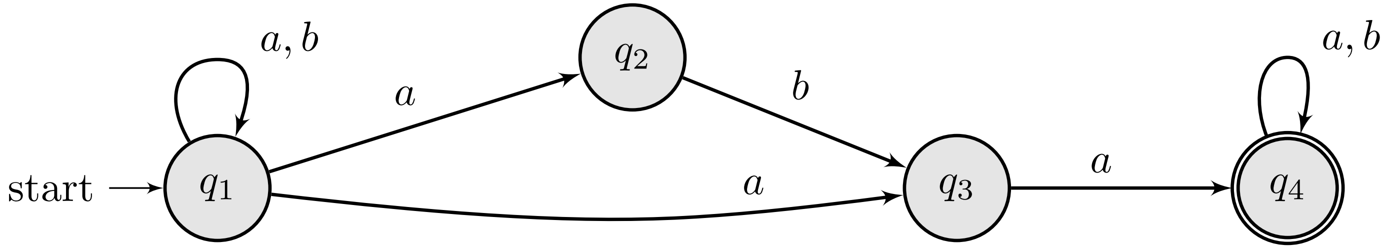
Exercise 2

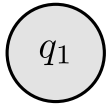
Let $\Sigma = \{a, b\}$. Give an NFA with **four states** that recognizes the following language

$$\{w \mid w \text{ contains the strings } aba \text{ or } aa\}$$

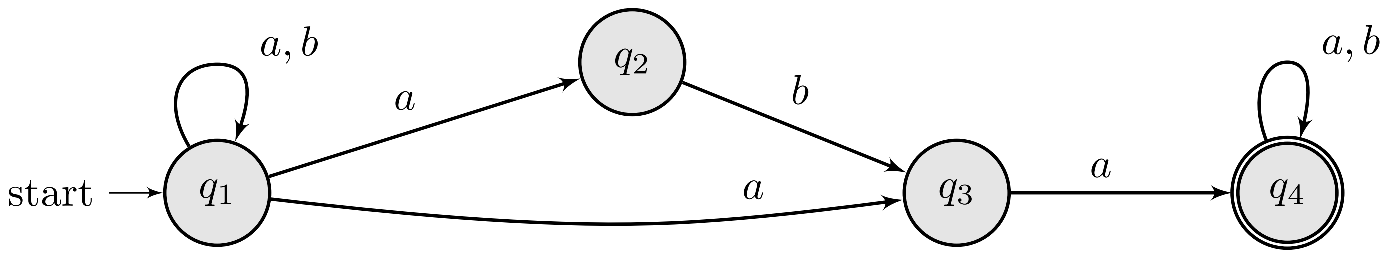


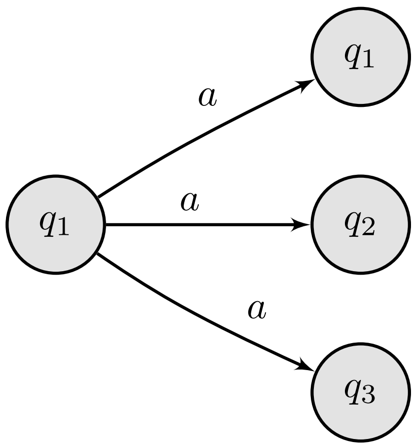
Exercise 2: acceptance of **a**aba



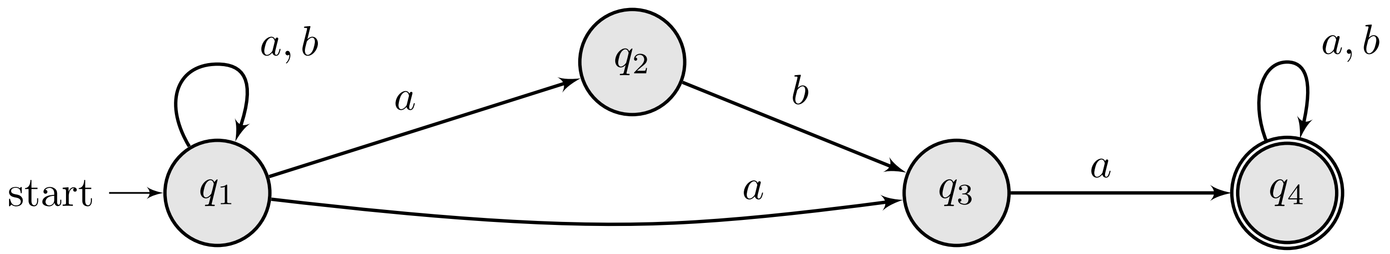


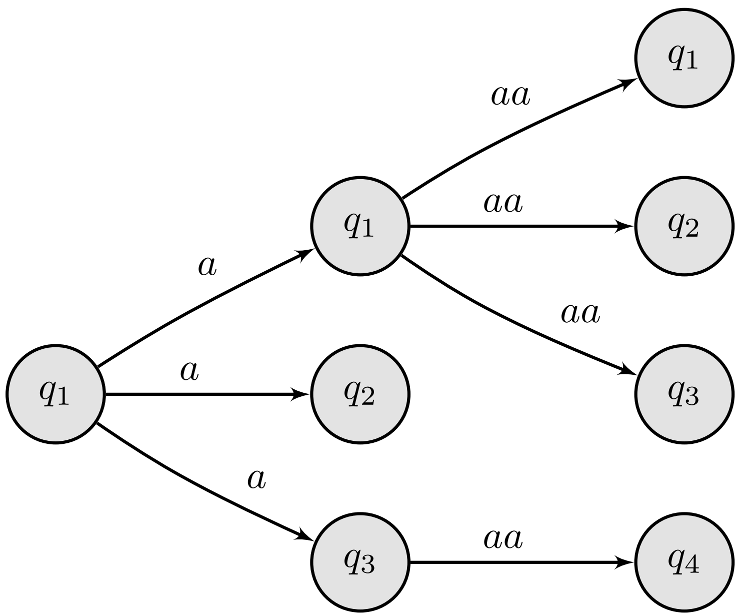
Exercise 2: acceptance of a**a**ba



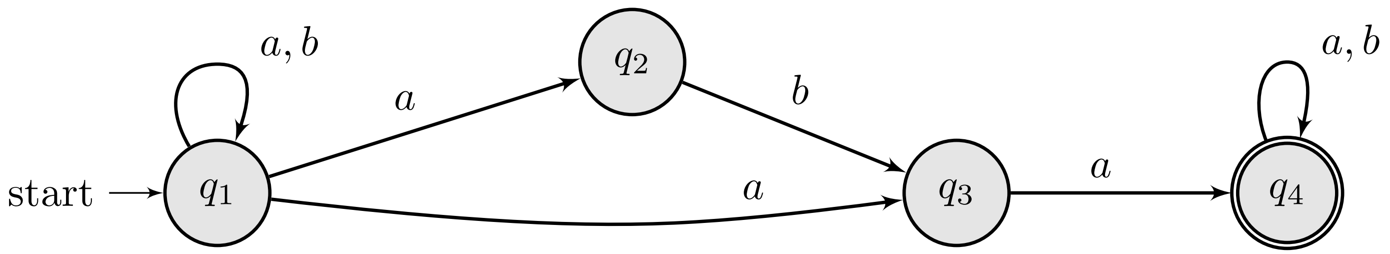


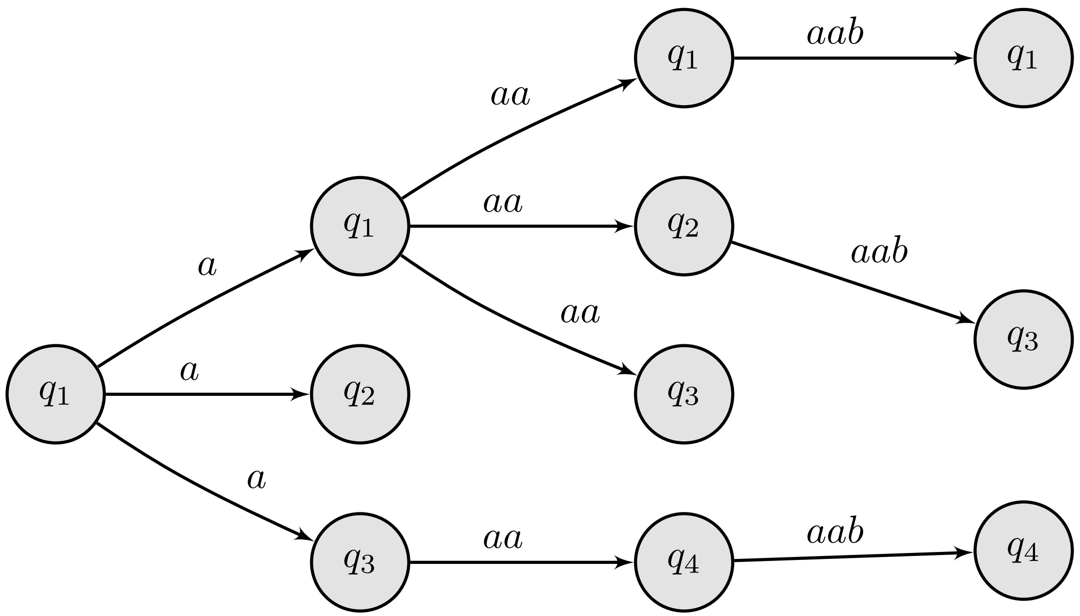
Exercise 2: acceptance of aa**b**a



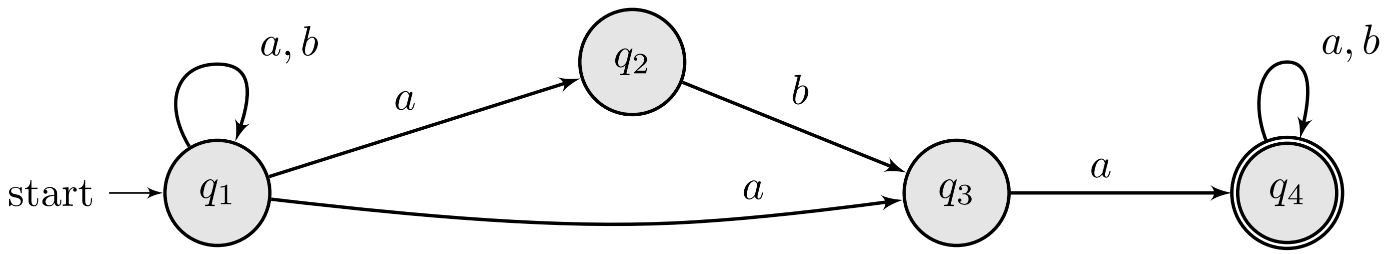


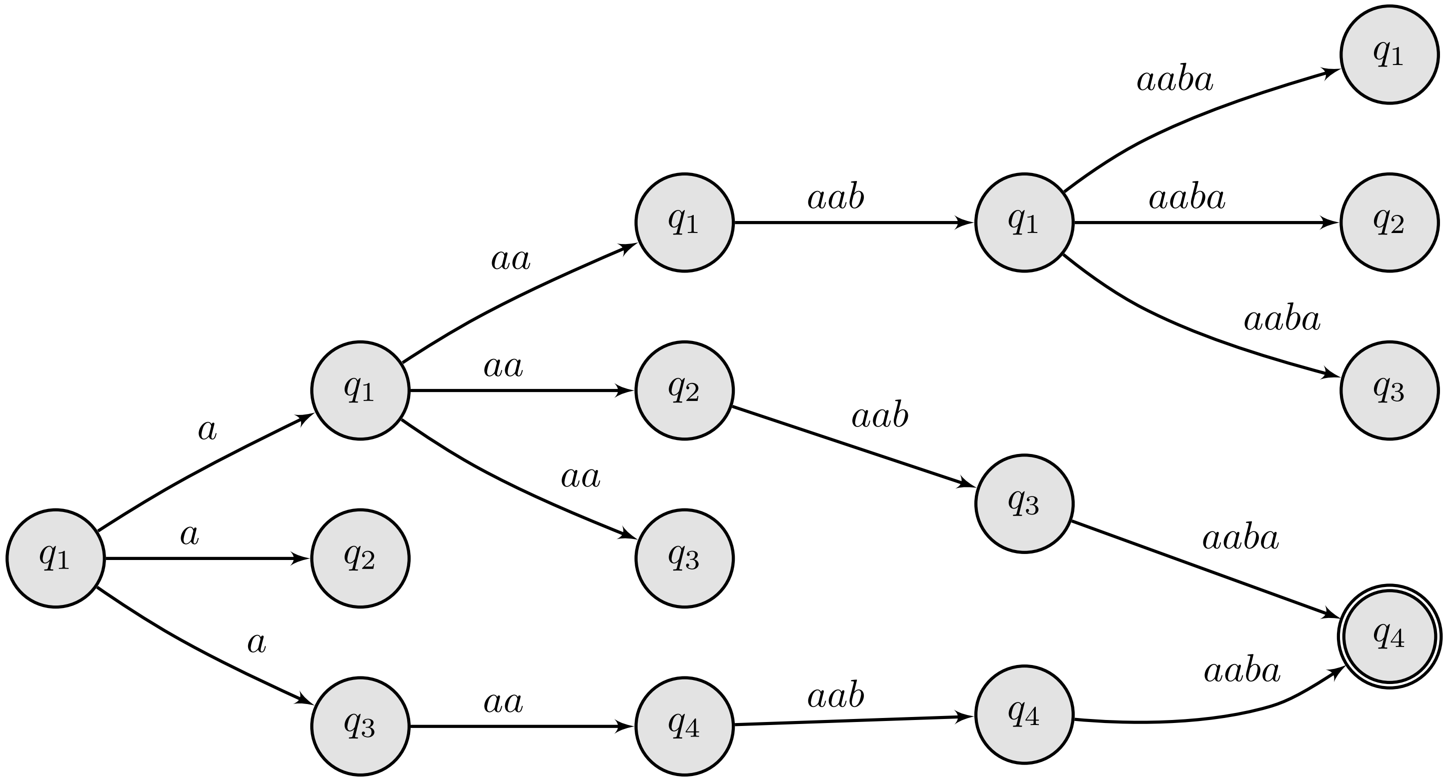
Exercise 2: acceptance of aab**a**





Exercise 2: acceptance of aaba





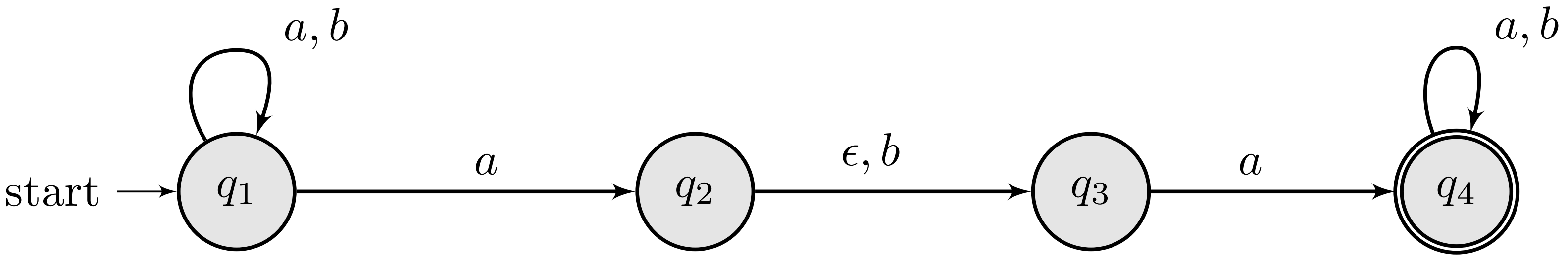
Epsilon transitions

Epsilon transitions

Exercise 2

Let $\Sigma = \{a, b\}$. Give an NFA with four states that recognizes the following language

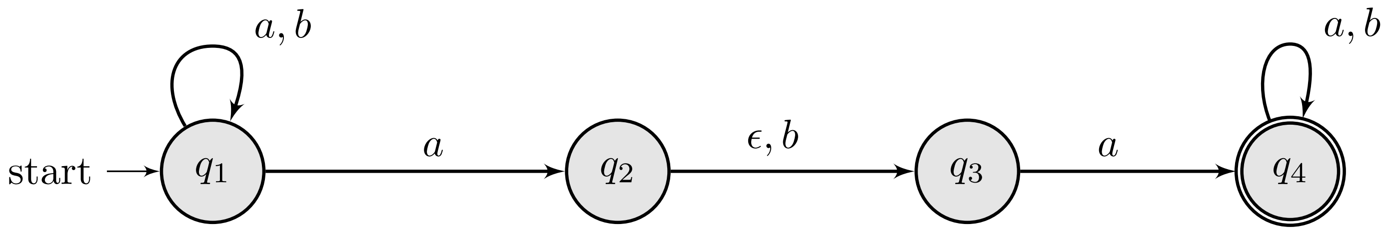
$$\{w \mid w \text{ contains the strings } aba \text{ or } aa\}$$



Note

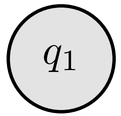
- NFAs can also include ϵ transitions, which may be taken without consuming an input

Exercise 2: acceptance of **a**aba

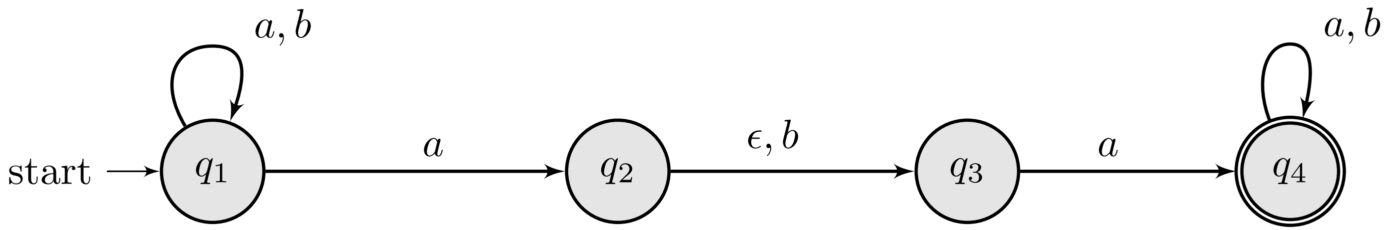


Interleave input with ϵ.

Read a

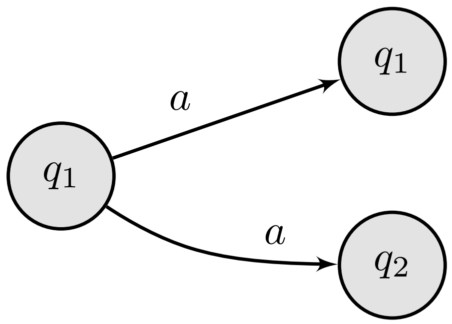


Exercise 2: acceptance of a**ϵ**aba

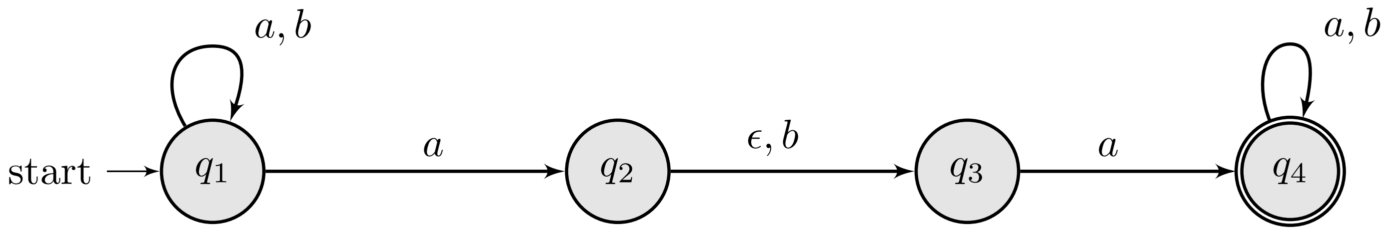


Interleave input with ϵ.

Read ϵ

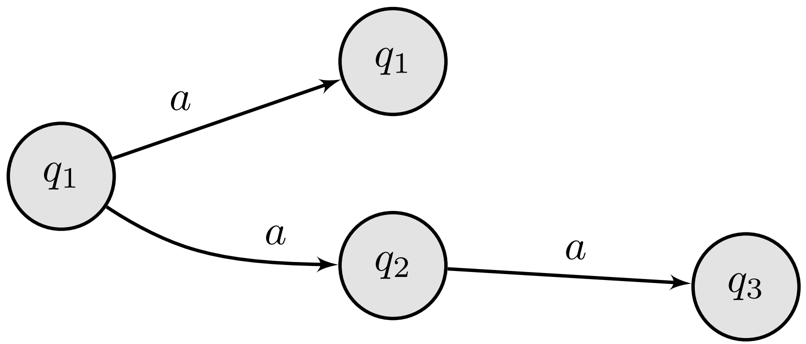


Exercise 2: acceptance of a**a**ba

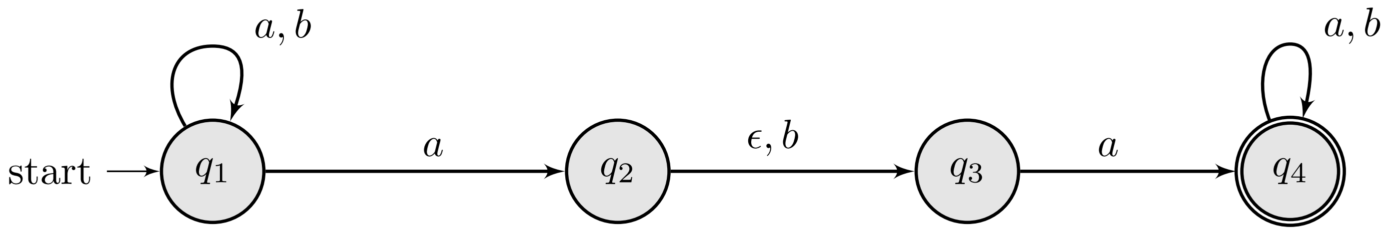


Interleave input with ϵ.

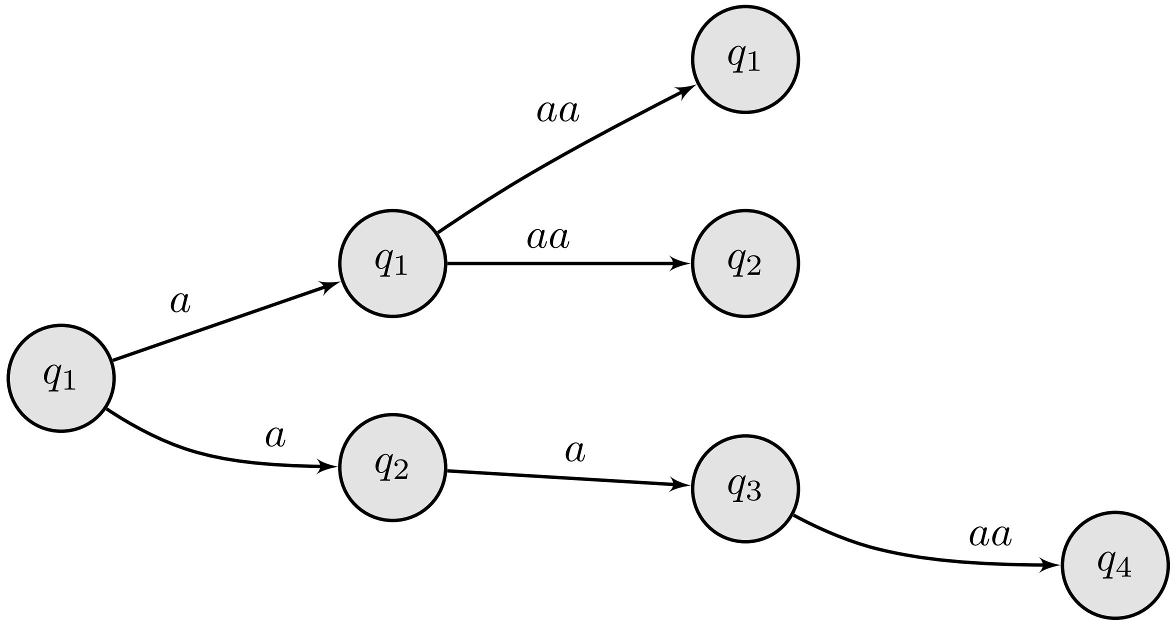
Read a



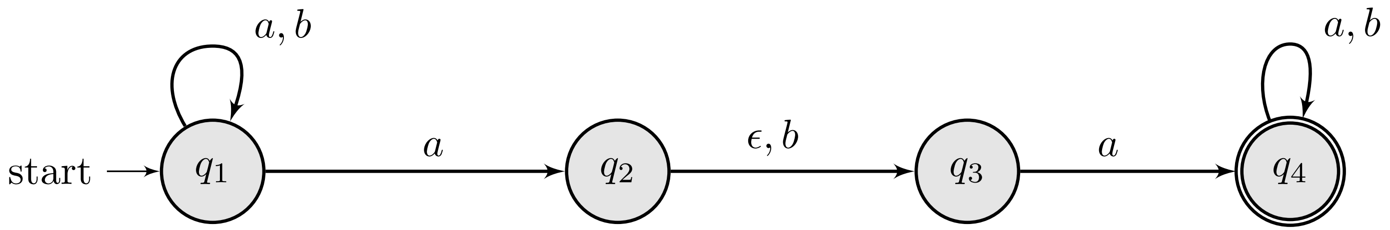
Exercise 2: acceptance of aab**ϵ**a



Interleave input with ϵ.
Read ϵ

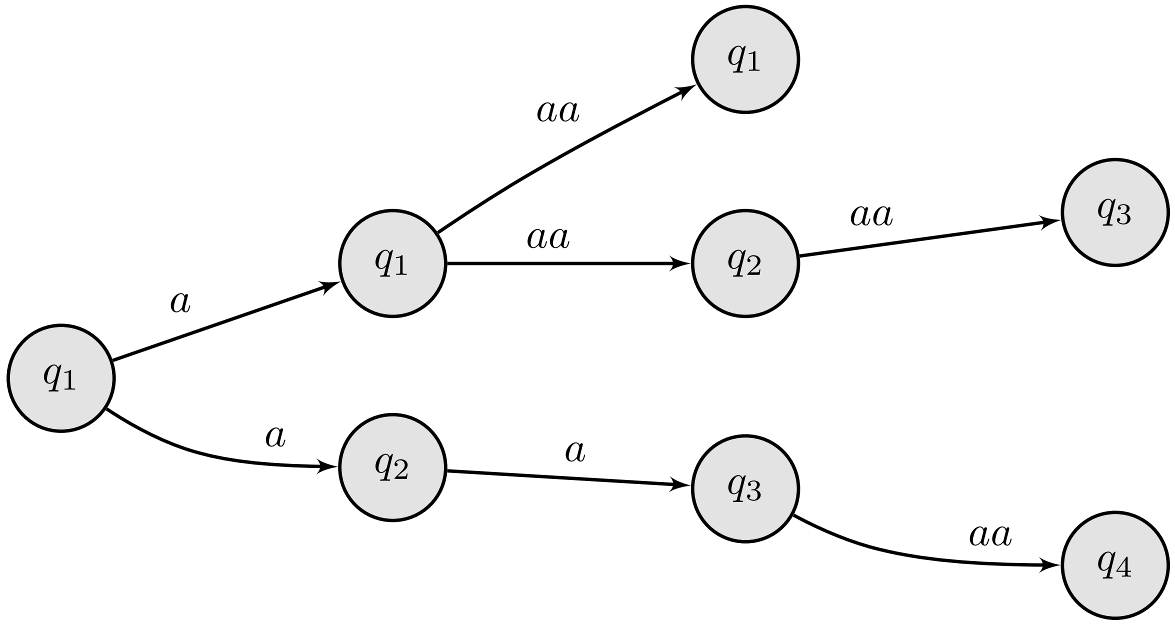


Exercise 2: acceptance of aa**b**a

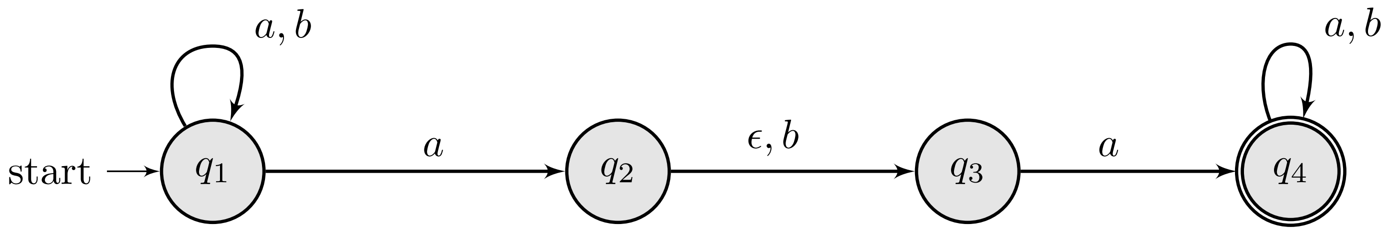


Interleave input with ϵ.

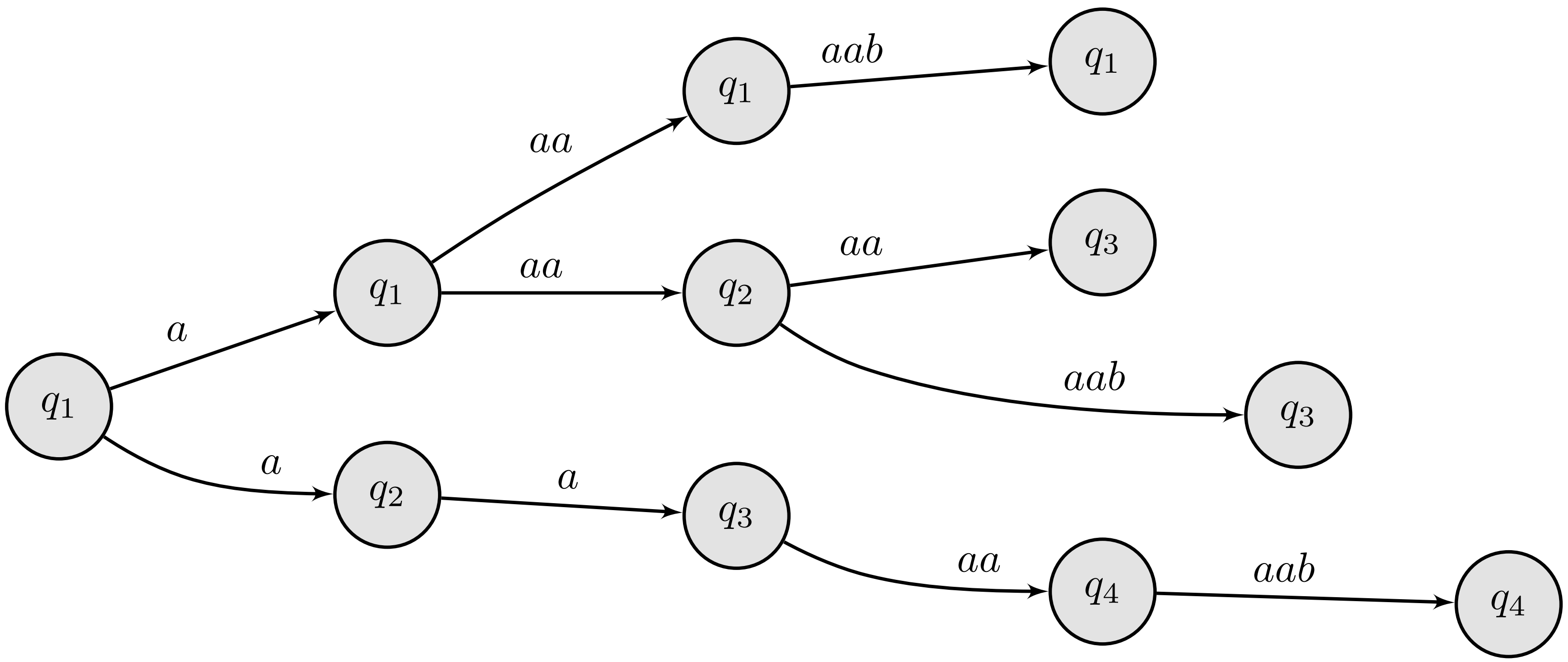
Read b



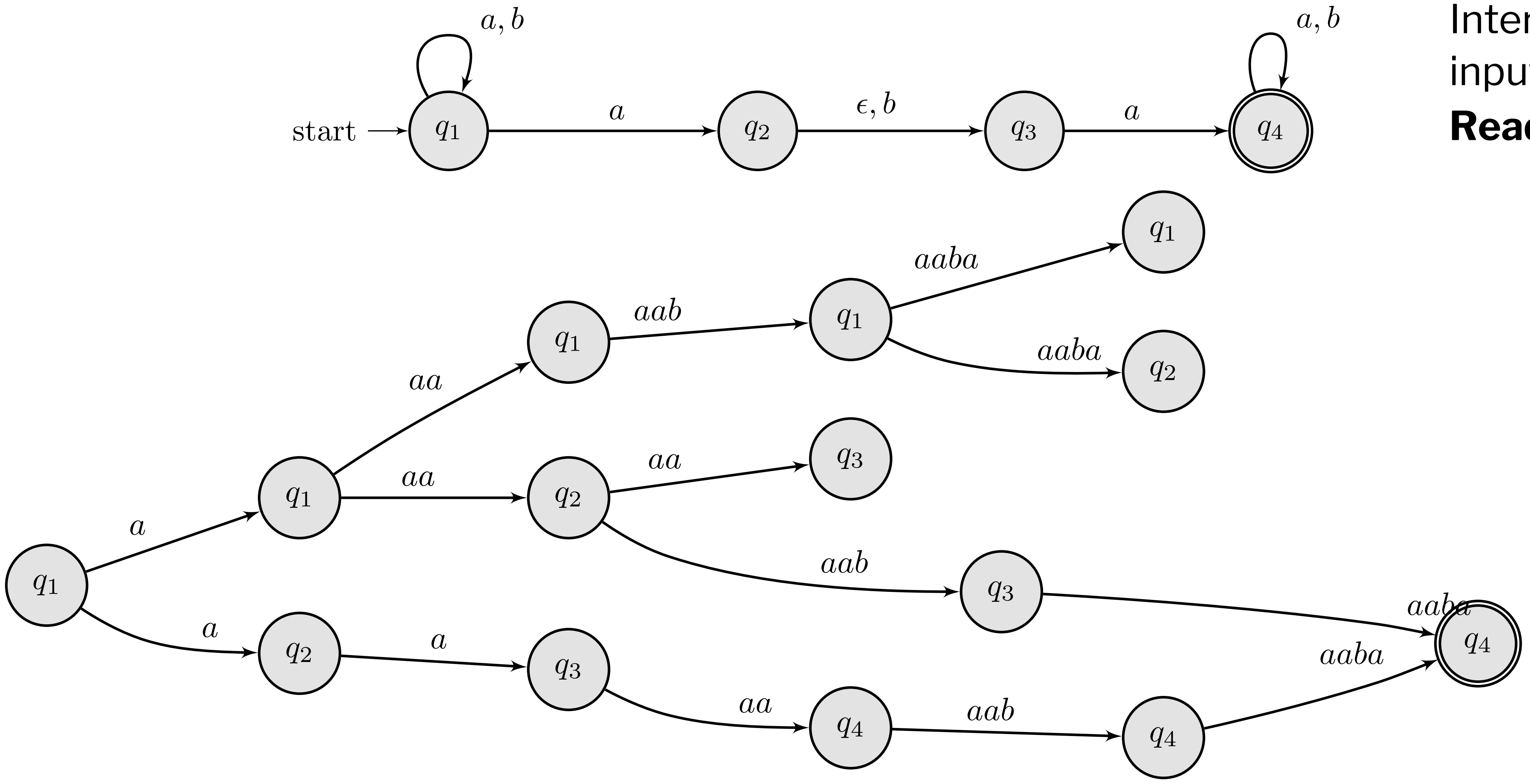
Exercise 2: acceptance of aab**a**



Interleave input with ϵ.
Read a



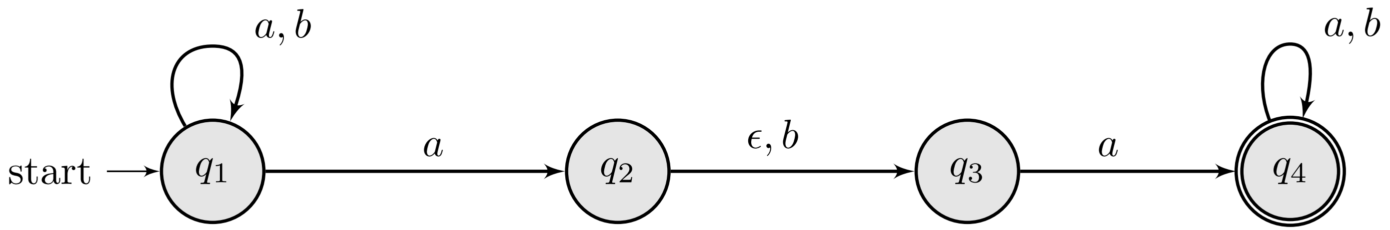
Exercise 2: acceptance of aabaϵ



Interleave input with ϵ.

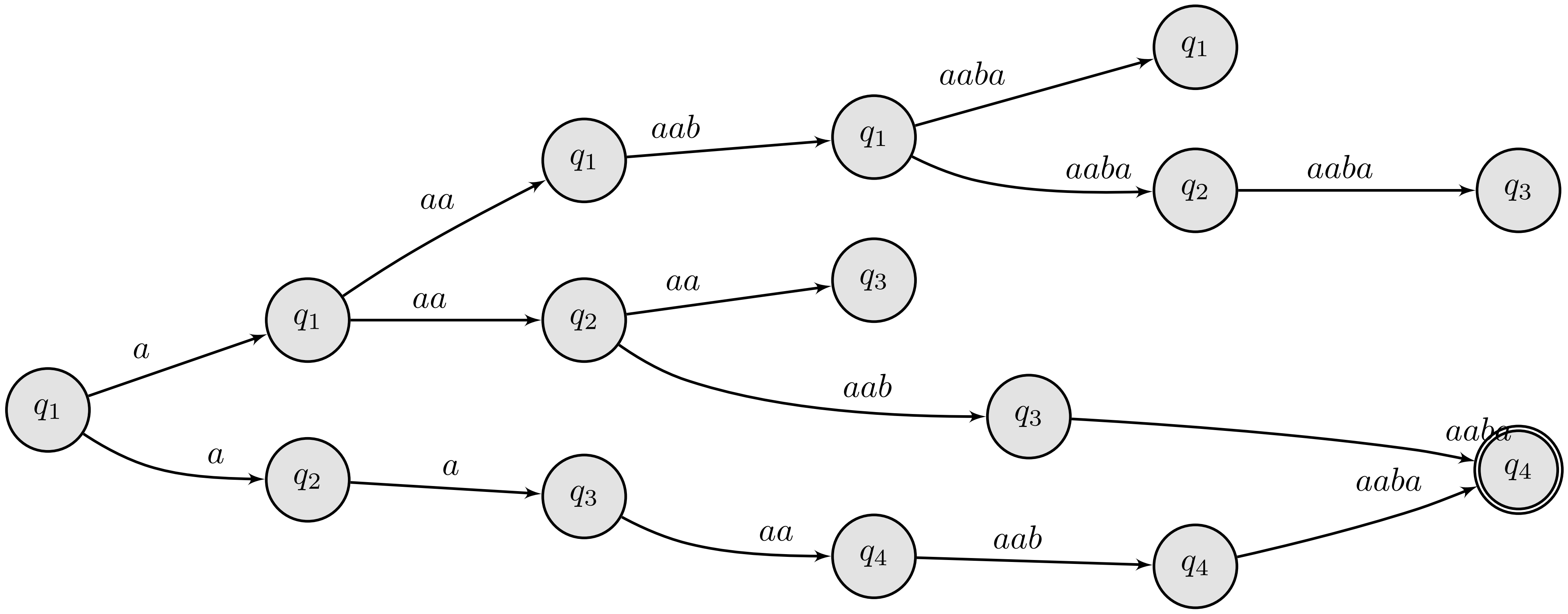
Read ϵ

Exercise 2: acceptance of aaba



Interleave input with ϵ.

Read ϵ



Formalizing NFA

Formalizing an NFA

I am now going to

- introduce the **NFA** definition (as a tuple)
- introduce an definition of **acceptance**
- introduce the definition of **acceptance**

> The two definitions of acceptance are equivalent.

Formalizing an NFA

Definition 1.37

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set called states
2. Σ is a finite set called alphabet
3. $\delta \colon Q \times \Sigma_\epsilon \rightarrow \quad (Q)$ is the transition function
   δ ϵ
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accepted states

Notes

- Function is known as the power-set, it takes a type and yields a set of elements of that type. Intuitively, you can think of it as type $\quad \langle T \rangle$, ie, a set of type T.
- Notation Σ_ϵ is an abbreviation of $\Sigma \cup \{\epsilon\}$

Formalizing acceptance of an NFA

Let $M = (Q, \Sigma, \delta, q_0, F)$, let the **steps through** relation, notation $q \quad {}_M\ w$, be defined as:

$$\frac{q \in F}{q \quad {}_M\ []}$$

Rule 1. State q steps through $[]$ if q is a final state.

$$\frac{q' \in \delta(q, y) \qquad q' \quad {}_M\ w}{q \quad {}_M\ y :: w}$$

Rule 2. If we can go from q to q' with y and q' steps through w, then q steps through $y :: w$.

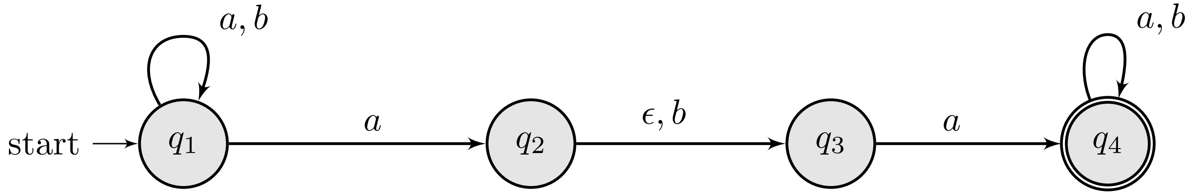
$$\frac{q' \in \delta(q, \epsilon) \qquad q' \quad {}_M\ w}{q \quad {}_M\ w}$$

Rule 3. If we can go from q to q' with ϵ and q' steps through w, then q also steps through w.

Acceptance. We say that M accepts w if, and only if, $q_0 \quad {}_M\ w$.

Example

Let $M = (\{q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, \{q_4\})$.

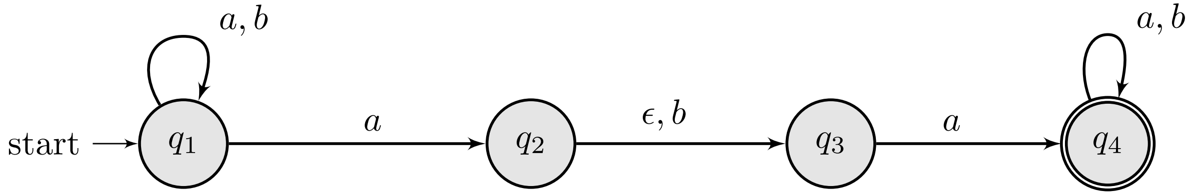


δ			ϵ
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_4\}$	$\emptyset$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Accept [b,a, a]? Proof:

Example

Let $M = (\{q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, \{q_4\})$.



δ			ϵ
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_4\}$	$\emptyset$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Accept [b,a, a]? Proof:

$$(R1)\ \frac{q \in F}{q \quad {}_M \; []}$$

$$(R2)\ \frac{q' \in \delta(q, y) \qquad q' \quad {}_M \; w}{q \quad {}_M \; y :: w}$$

$$(R3)\ \frac{q' \in \delta(q, \epsilon) \qquad q' \quad {}_M \; w}{q \quad {}_M \; w}$$

$$\frac{q_1 \in \delta(q_1, b) \qquad \dfrac{q_2 \in \delta(q_1, a) \qquad \dfrac{q_3 \in \delta(q_2, \epsilon) \qquad \dfrac{q_4 \in \delta(q_3, a) \qquad \dfrac{q_4 \in \{q_4\}}{q_4 \quad {}_M \; []}}{q_3 \quad {}_M \; [a]}}{q_2 \quad {}_M \; [a, a]}}{q_1 \quad {}_M \; [a, a]}}{q_1 \quad {}_M \; [b, a, a]}$$

NFA Acceptance (book version)

We say that M accepts w if there exists a sequence of states $r_0, \ldots, r_m$ such that $w =^{\star} y_1, \ldots, y_m, \forall y_i \in \Sigma_\epsilon, \forall r_i \in Q$, and:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for $i = 0, \ldots, m-1$
3. $r_m \in F$

***Warning:** The book implicitly assumes equality up to removing ϵ. For instance, the book's definition assumes that $[b, a, \epsilon, a] = [b, a, a]$,

Example

According to the definition above M accepts $[b, a, a]$ with the sequence of states

$q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{a} q_4$ or just $q_1 q_1 q_2 q_3 q_4$

Implementing an NFA

Implementing an NFA

I am now going to

- implement an NFA as a Python class
- implement the acceptance algorithm
- show that we can translate a DFA into an NFA
- show that we can translate an NFA into a DFA

> The implementation may serve as an intuition to understand the translation from an NFA into a DFA.

Implementing an NFA

An NFA $(Q, \Sigma, \delta, q_0, F)$ can be implemented with:

```
class NFA:
  def __init__(self, states, alphabet, transition_func, start_state, accepted_states):
    assert start_state in states, "%r in %r" % (start_state, states)
    self.states = states
    self.alphabet = alphabet
    self.transition_func = transition_func
    self.start_state = start_state
    self.accepted_states = accepted_states
```

Nondeterministic acceptance

Intuition

- **States**:

Nondeterministic acceptance

Intuition

- **States**: Each state becomes a set of all possible concurrent states of the NFA
  For instance state $\{q_1, q_2, q_3\}$ says that the acceptance algorithm is concurrently on these three states
- **Alphabet:**

Nondeterministic acceptance

Intuition

- **States**: Each state becomes a set of all possible concurrent states of the NFA
  For instance state $\{q_1, q_2, q_3\}$ says that the acceptance algorithm is concurrently on these three states
- **Alphabet:** same alphabet
- **Initial state:**

Nondeterministic acceptance

Intuition

- **States**: Each state becomes a set of all possible concurrent states of the NFA
  For instance state $\{q_1, q_2, q_3\}$ says that the acceptance algorithm is concurrently on these three states
- **Alphabet:** same alphabet
- **Initial state:** The start from an initial step and perform all possible ϵ transitions.
- **Transition:**

Nondeterministic acceptance

Intuition

- **States**: Each state becomes a set of all possible concurrent states of the NFA
  For instance state $\{q_1, q_2, q_3\}$ says that the acceptance algorithm is concurrently on these three states
- **Alphabet:** same alphabet
- **Initial state:** The start from an initial step and perform all possible ϵ transitions.
- **Transition:** Read one input on each "sub-state" (the input step) and then perform ϵ-transitions (the epsilon-step)

NFA Implementation

```python
def accepts(self, inputs):

  states = self.epsilon({self.start_state})
  for i in inputs:
    if len(states) == 0:

      return False

    states = self.epsilon(self.transition(states, i))

  states = set(filter(self.accepted_states, states))
  return len(states) > 0
```

- **Input-step:** method `transition` performs a transition for every state (function $\delta_\cup$)
- **Epsilon-step:** method `epsilon` performs all possible ϵ-transitions from a given set Q (function E)

Nondeterministic transition $\delta_{\cup}$

$$\delta_{\cup}(R, a) = \underset{q \in R}{\quad} \delta(r, a)$$

```python
def transition(self, states, input):
  new_states = set()
  for st in states:
    new_states.update(self.transition_func(st, input))
  return frozenset(new_states)
```

(See Theorem 1.39; in the book $\delta_{\cup}$ is δ')

Epsilon transition

$$E(R) = \{q \mid q \text{ can be reached from R by travelling along 0 or more } \epsilon \text{ arrows}\}$$

```python
def epsilon(self, states):
  states = set(states)
  while True:
    count = len(states)
    states.update(self.transition(states, None))
    if count == len(states):
      return states
```

(See Theorem 1.39)

Are all DFAs also NFAs?

Are all DFAs also NFAs?

- **Yes,** DFAs can be trivially converted into NFAs.
  The state diagram of a DFA is equivalent to the same state diagram as an NFA.
- We only need to slightly change the transition function to handle ϵ inputs.

Are all DFAs also NFAs?

- **Yes,** DFAs can be trivially converted into NFAs.
  The state diagram of a DFA is equivalent to the same state diagram as an NFA.
- We only need to slightly change the transition function to handle ϵ inputs.

Implementation

```
def convert_to_nfa(dfa):
  return NFA(
    states=dfa.states,
    alphabet=dfa.alphabet,
    transition_func=lambda q, a: {dfa.transition_func(q, a),} if a is not None else {},
    start_state=dfa.start_state,
    accepted_states=dfa.accepted_states
  )
```

Are all NFAs also DFAs?

Are all NFAs also DFAs?

Yes!

Theorem 1.39

Every NFA has an equivalent DFA

- We study the algorithm that converts an NFA into a DFA
  This algorithm will be examined in Mini-Test 1.
- **Tip:** understanding the implementation of the acceptance algorithm, helps understanding the conversion and vice-versa

Intuion

- **States:**

Theorem 1.39

Every NFA has an equivalent DFA

- We study the algorithm that converts an NFA into a DFA
  This algorithm will be examined in Mini-Test 1.
- **Tip:** understanding the implementation of the acceptance algorithm, helps understanding the conversion and vice-versa

Intuion

- **States:** Each state becomes a set of all possible concurrent states of the NFA
- **Alphabet:**

Theorem 1.39

Every NFA has an equivalent DFA

- We study the algorithm that converts an NFA into a DFA
  This algorithm will be examined in Mini-Test 1.
- **Tip:** understanding the implementation of the acceptance algorithm, helps understanding the conversion and vice-versa

Intuion

- **States:** Each state becomes a set of all possible concurrent states of the NFA
- **Alphabet:** same alphabet
- **Initial state:**

Theorem 1.39

Every NFA has an equivalent DFA

- We study the algorithm that converts an NFA into a DFA
  This algorithm will be examined in Mini-Test 1.
- **Tip:** understanding the implementation of the acceptance algorithm, helps understanding the conversion and vice-versa

Intuion

- **States:** Each state becomes a set of all possible concurrent states of the NFA
- **Alphabet:** same alphabet
- **Initial state:** The state that consists of an epsilon-step on the initial state.
- **Transition:**

Theorem 1.39

Every NFA has an equivalent DFA

- We study the algorithm that converts an NFA into a DFA
  This algorithm will be examined in Mini-Test 1.
- **Tip:** understanding the implementation of the acceptance algorithm, helps understanding the conversion and vice-versa

Intuion

- **States:** Each state becomes a set of all possible concurrent states of the NFA
- **Alphabet:** same alphabet
- **Initial state:** The state that consists of an epsilon-step on the initial state.
- **Transition:** One input-step followed by one epsilon-step

Are all NFAs also DFAs?

```python
def nfa_to_dfa(nfa):
  def transition(q, c):
    return nfa.epsilon(nfa.transition(q, c))

  def accept_state(qs):
    for q in qs:
      if nfa.accepted_states(q):
        return True
    return False

  return DFA(
    powerset(nfa.states),
    nfa.alphabet,
    transition,
    nfa.epsilon({nfa.start_state}),
    accept_state)
```

Theorem 1.39

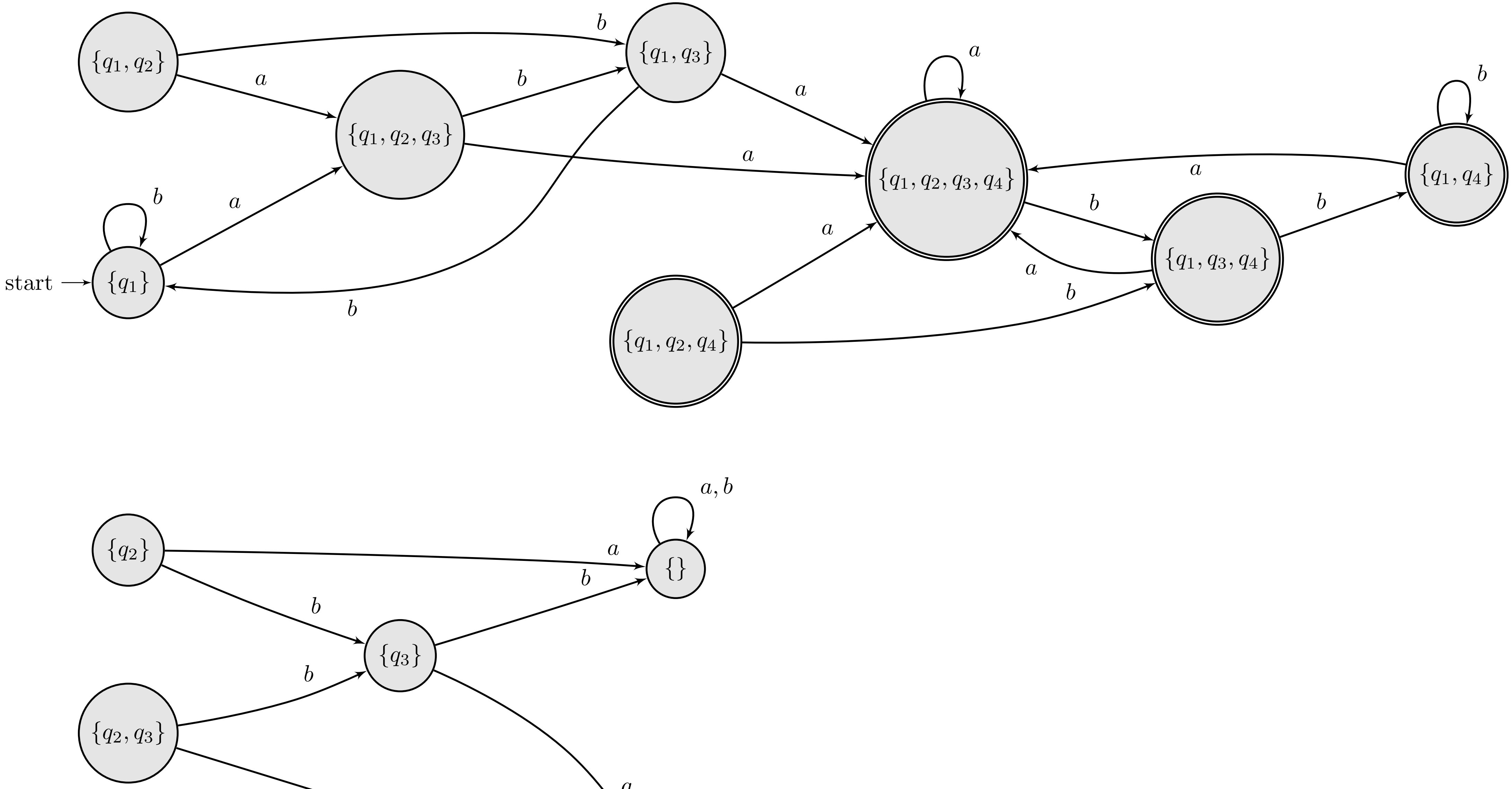
Every NFA has an equivalent DFA

Formally, we introduce function $\mathrm{nfa2dfa}$ that converts an NFA into a DFA.

$\mathrm{nfa2dfa}((Q, \Gamma, \delta, q_1, F)) = (\quad (Q), \Gamma, \delta_D, E(q_1), F_D)$ where

- $\delta_D(Q, c) = E(\delta_\cup(Q, c))$
- $F_D = \{Q \mid Q \cap F \neq \emptyset\}$

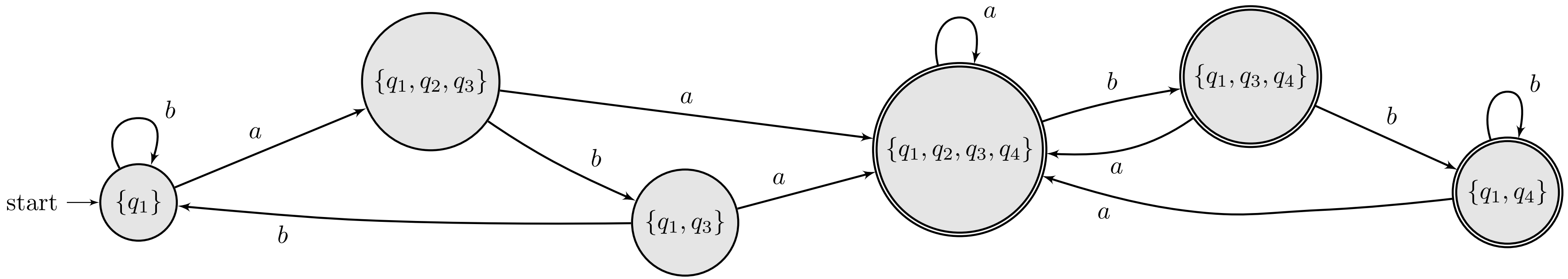
Producing a DFA from an NFA

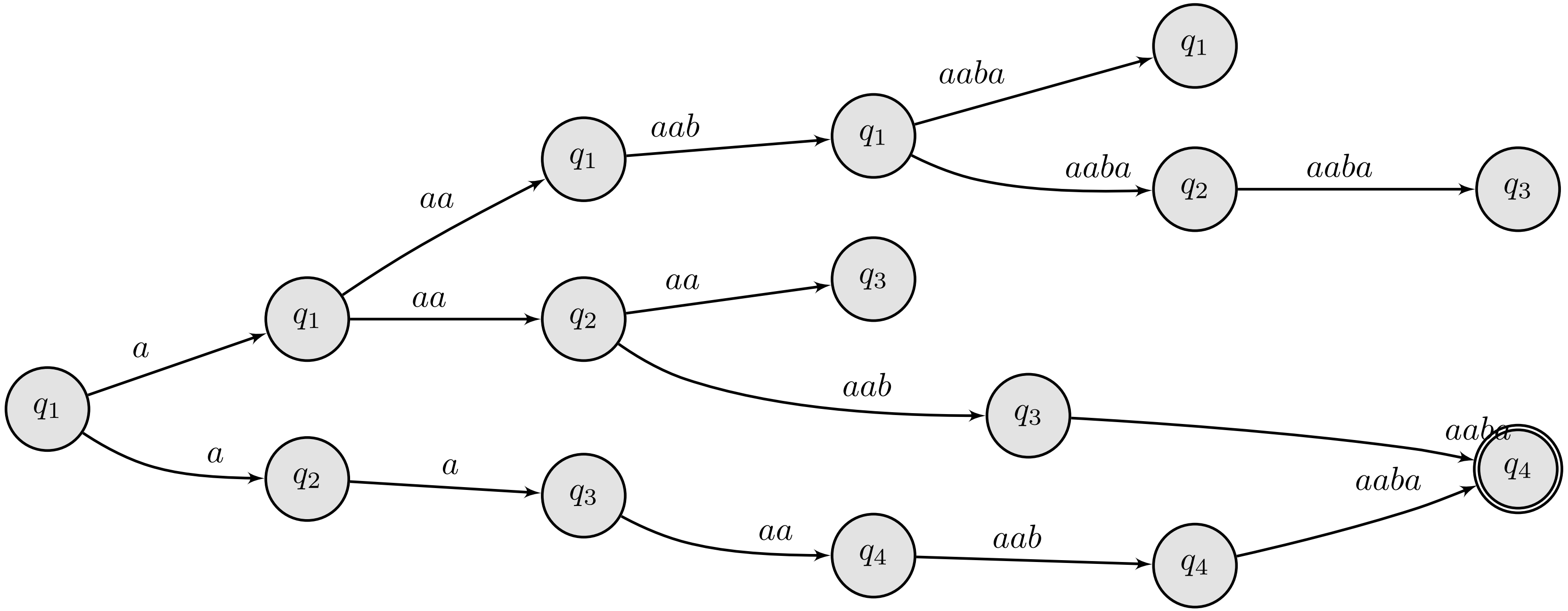


Producing a DFA from an NFA

- The algorithm we implemented yields unreachable states
- We can eliminate such states with a standard graph operation: we obtain the strongly connected component of the initial state: the subgraph that consists of every reachable state from the initial state.

Example


start
{q1}
{q1, q2, q3}
{q1, q3}
{q1, q2, q3, q4}
{q1, q3, q4}
{q1, q4}
a
b


q1
q2
q3
q4
a
aa
aab
aaba