CS420

Introduction to the Theory of Computation

Lecture 13: Regular expressions & NFAs

Tiago Cogumbreiro

Today we will learn...

- Converting REGEX to NFA
- Converting NFA to REGEX

Soundess

All Regexes have an equivalent NFA

$\mathrm{REGEX} \rightarrow \mathrm{NFA}$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) = \mathrm{char}(a)$
- $\mathrm{NFA}(\underline{\epsilon}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) = \mathrm{char}(a)$
- $\mathrm{NFA}(\underline{\epsilon}) = \mathrm{nil}$
- $\mathrm{NFA}(\underline{\emptyset}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) = \mathrm{char}(a)$
- $\mathrm{NFA}(\underline{\epsilon}) = \mathrm{nil}$
- $\mathrm{NFA}(\underline{\emptyset}) = \mathrm{void}$
- $\mathrm{NFA}(\underline{R_1 \cup R_2}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) = \mathrm{char}(a)$
- $\mathrm{NFA}(\underline{\epsilon}) = \mathrm{nil}$
- $\mathrm{NFA}(\underline{\emptyset}) = \mathrm{void}$
- $\mathrm{NFA}(\underline{R_1 \cup R_2}) = \mathrm{union}(\mathrm{NFA}(R_1), \mathrm{NFA}(R_2))$
- $\mathrm{NFA}(\underline{R_1 \cdot R_2}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\text{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\text{NFA}(\underline{a}) = \text{char}(a)$
- $\text{NFA}(\underline{\epsilon}) = \text{nil}$
- $\text{NFA}(\underline{\emptyset}) = \text{void}$
- $\text{NFA}(\underline{R_1 \cup R_2}) = \text{union}(\text{NFA}(R_1), \text{NFA}(R_2))$
- $\text{NFA}(\underline{R_1 \cdot R_2}) = \text{append}(\text{NFA}(\underline{R_1}), \text{NFA}(\underline{R_2}))$
- $\text{NFA}(\underline{R^\star}) =$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\mathrm{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\mathrm{NFA}(\underline{a}) = \mathrm{char}(a)$
- $\mathrm{NFA}(\underline{\epsilon}) = \mathrm{nil}$
- $\mathrm{NFA}(\underline{\emptyset}) = \mathrm{void}$
- $\mathrm{NFA}(\underline{R_1 \cup R_2}) = \mathrm{union}(\mathrm{NFA}(R_1), \mathrm{NFA}(R_2))$
- $\mathrm{NFA}(\underline{R_1 \cdot R_2}) = \mathrm{append}(\mathrm{NFA}(\underline{R_1}), \mathrm{NFA}(\underline{R_2}))$
- $\mathrm{NFA}(\underline{R^\star}) = \mathrm{star}(\mathrm{NFA}(\underline{R}))$

All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\text{NFA}(R)) = L_1$.

Given an alphabet Σ

- $\text{NFA}(\underline{a}) = \text{char}(a)$
- $\text{NFA}(\underline{\epsilon}) = \text{nil}$
- $\text{NFA}(\underline{\emptyset}) = \text{void}$
- $\text{NFA}(\underline{R_1 \cup R_2}) = \text{union}(\text{NFA}(R_1), \text{NFA}(R_2))$
- $\text{NFA}(\underline{R_1 \cdot R_2}) = \text{append}(\text{NFA}(\underline{R_1}), \text{NFA}(\underline{R_2}))$
- $\text{NFA}(\underline{R^\star}) = \text{star}(\text{NFA}(\underline{R}))$

(Proof follows by induction on the structure of R.)

The **void** NFA

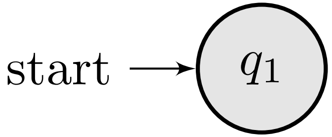
$$L(\text{void}) = \emptyset$$

The **void** NFA

$$L(\textbf{void}) = \emptyset$$

The **void** NFA

$L(\mathbf{void}) = \emptyset$



The **nil** operator

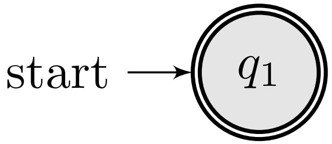
$$L(\mathrm{nil}) = \{\epsilon\}$$

The **nil** operator

$$L(\mathbf{nil}) = \{\epsilon\}$$

The **nil** operator

$$L(\mathbf{nil}) = \{\epsilon\}$$



The $\mathbf{char}(\boldsymbol{c})$ operator

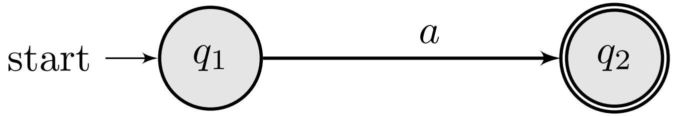
$$L(\text{char}(c)) = \{[c]\}$$

The $\mathbf{char}(a)$ operator

$$L(\mathrm{char}(a)) = \{[a]\}$$

The $\mathbf{char}(a)$ operator

$$L(\text{char}(a)) = \{[a]\}$$



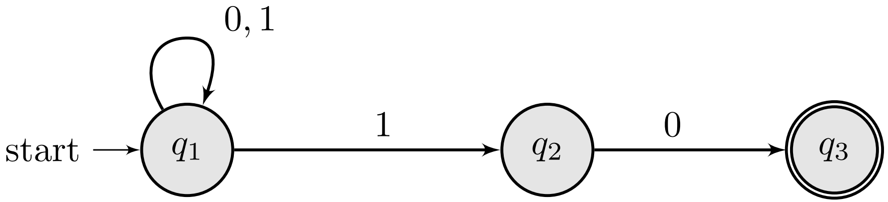
The $\mathbf{union}(M, N)$ automaton

$$L(\text{union}(M, N)) = L(M) \cup L(N)$$

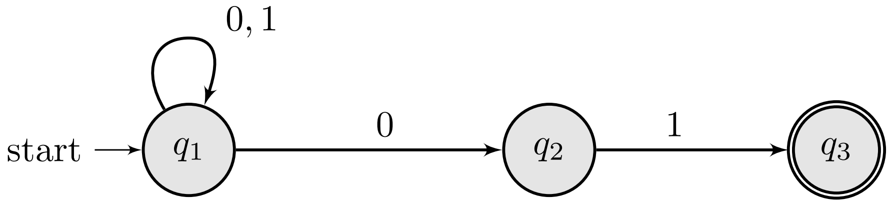
The **union**(M, N) automaton

$$L(\textbf{union}(M, N)) = L(M) \cup L(N)$$

N_1



N_2

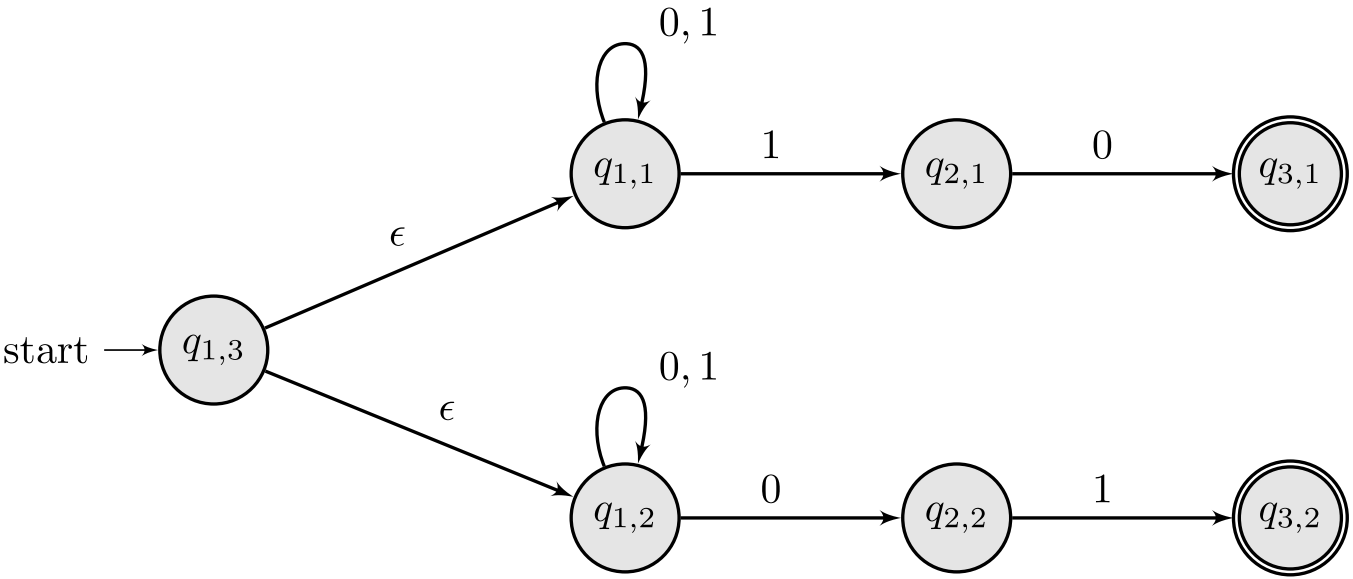


$$\textbf{union}(N_1, N_2) = ?$$

The $\mathbf{union}(M, N)$ operator

$$L(\mathrm{union}(M, N)) = L(M) \cup L(N)$$

Example $\mathbf{union}(N_1, N_2)$



- Add a new initial state
- Connect new initial state to the initial states of N_1 and N_2 via ϵ-transitions.

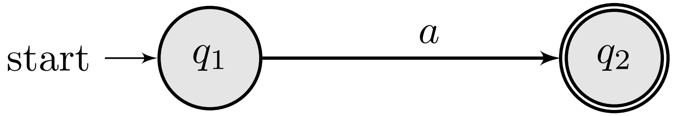
The $\mathbf{append}(M, N)$ operator

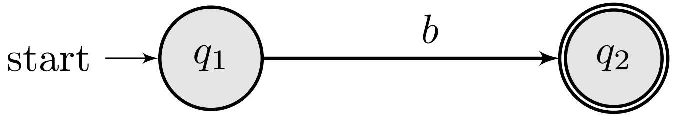
$$L(\text{append}(M, N)) = L(M) \cdot L(N)$$

The $\textbf{append}(M, N)$ operator

$$L(\textbf{append}(M, N)) = L(M) \cdot L(N)$$

Example 1: $L(\textbf{concat}(\textbf{char}(\texttt{a}), \textbf{char}(\texttt{b}))) = \{\texttt{ab}\}$

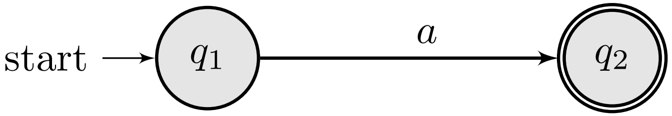


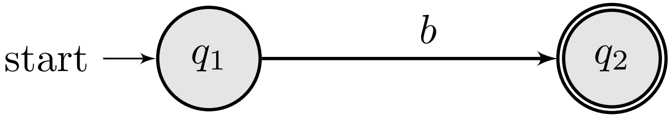


The $\mathbf{append}(M, N)$ operator

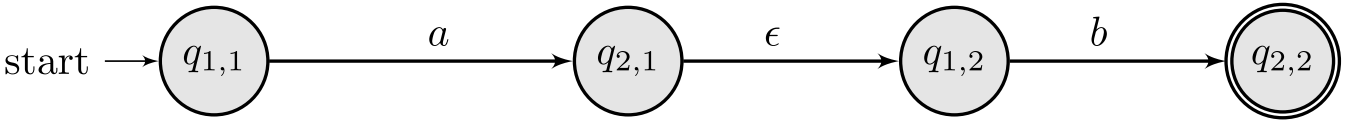
$L(\mathbf{append}(M, N)) = L(M) \cdot L(N)$

Example 1: $L(\mathbf{concat}(\mathbf{char}(\mathtt{a}), \mathbf{char}(\mathtt{b}))) = \{\mathtt{ab}\}$





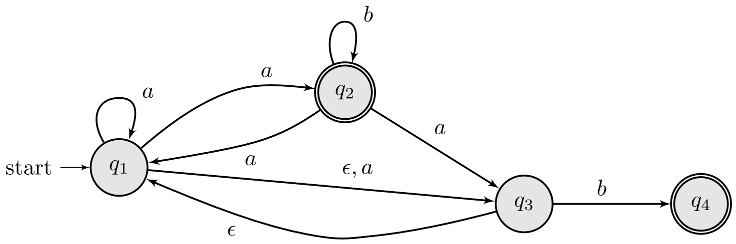
Solution

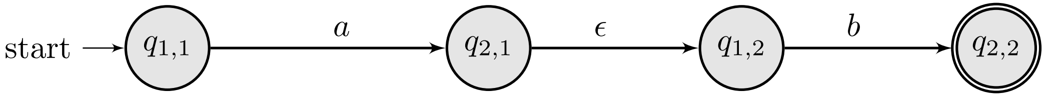


What did we do? Connect the accepted states of N_1 to the initial state of N_2 via ϵ-transitions.

> Why bot connect directly from $q_{1,1}$ into $q_{1,2}$? See next slide.

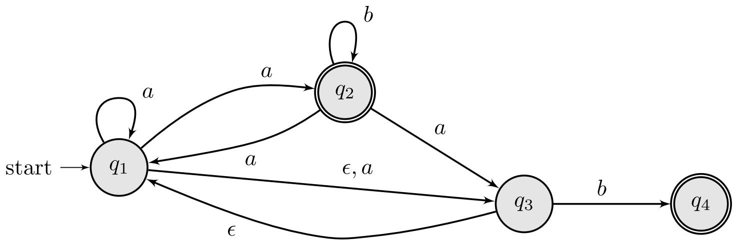
Concatennation example 2

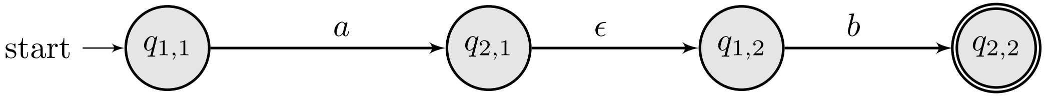




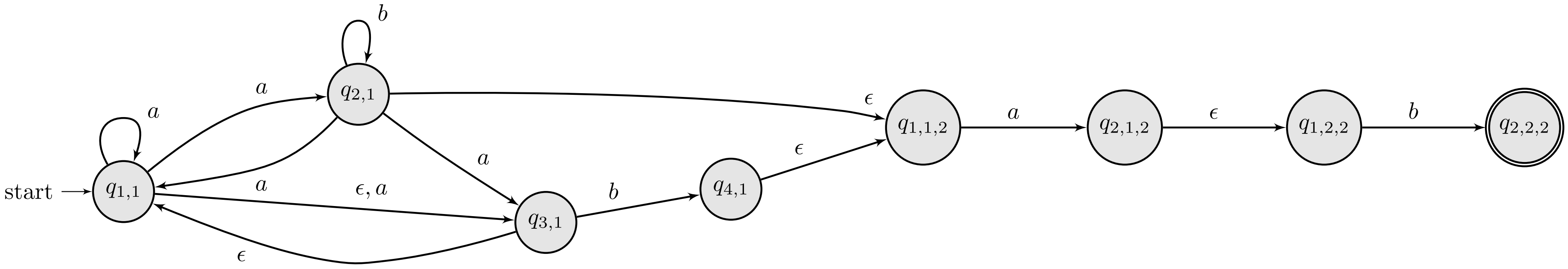
Solution

Concatennation example 2





Solution



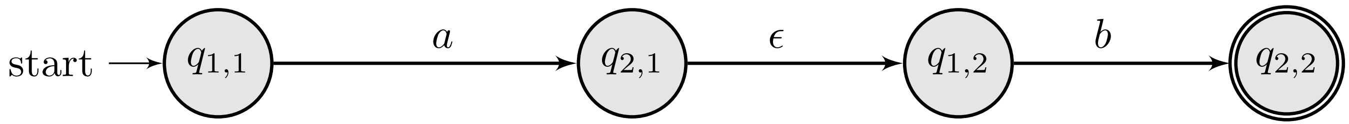
The $\mathbf{star}(N)$ operator

$$L(\mathrm{star}(N)) = L(N)^\star$$

The $\mathbf{star}(N)$ operator

$L(\mathrm{star}(N)) = L(N)^\star$

Example: $L(\mathrm{star}(\mathrm{concat}(\mathrm{char}(\texttt{a}), \mathrm{char}(\texttt{b})))) = \{w \mid w \text{ is a sequence of } \texttt{ab} \text{ or empty}\}$

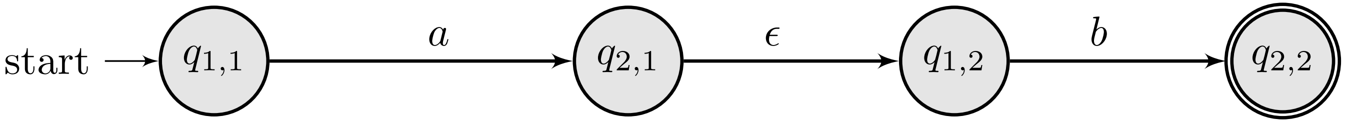


Solution

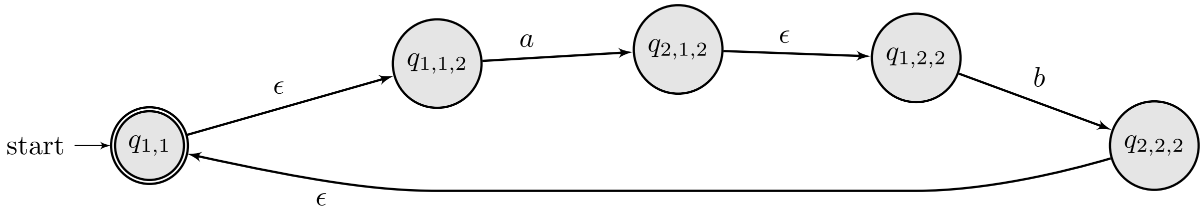
The $\mathbf{star}(N)$ operator

$L(\mathrm{star}(N)) = L(N)^\star$

Example: $L(\mathrm{star}(\mathrm{concat}(\mathrm{char}(\mathtt{a}), \mathrm{char}(\mathtt{b})))) = \{w \mid w \text{ is a sequence of } \mathtt{ab} \text{ or empty}\}$



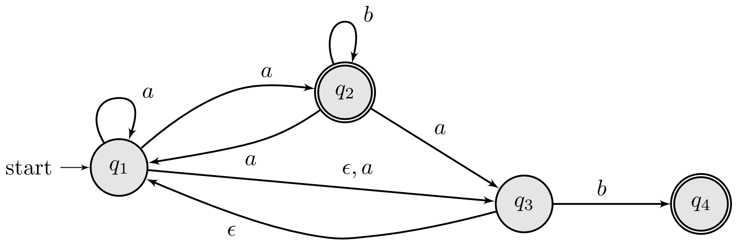
Solution



- create a new state $q_{1,1}$
- ϵ-transitions from $q_{1,1}$ to initial state
- ϵ-transitions from accepted states to $q_{1,1}$
- $q_{1,1}$ is the only accepted state

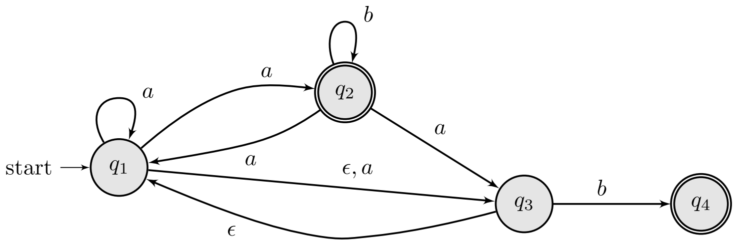
The $\mathbf{star}(N)$ operator

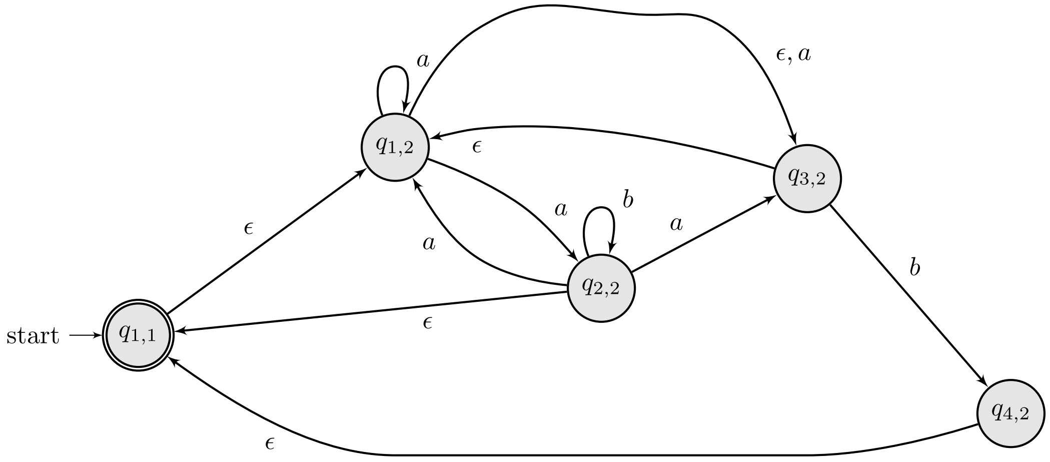
$$L(\mathbf{star}(N)) = L(N)^\star$$



The $\mathbf{star}(N)$ operator

$L(\mathbf{star}(N)) = L(N)^{\star}$





Completeness

All NFAs have an equivalent Regex

NFA → REGEX

Completeness

All NFAs have an equivalent Regex

Why is this result important?

Completeness

All NFAs have an equivalent Regex

Why is this result important?

> If we can derive an equivalent regular expression from any NFA, then our regular expression are enough to describe whatever can be described using finite automatons.

Overview:

Converting an NFA into a regular expression

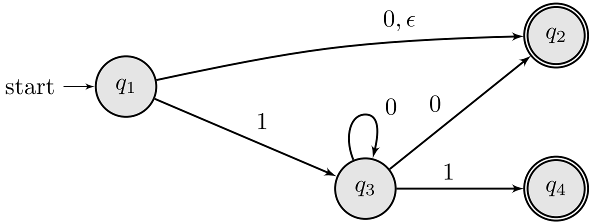
> There are many algorithms of converting an NFA into a Regex. Here is the algorithm we find in the book.

1. Wrap the NFA
2. Convert the NFA into a GNFA
3. Reduce the GNFA
4. Extract the Regex

Step 1: wrap the NFA

> Given an NFA N, add two new states q_{start} and q_{end} such that q_{start} transitions via ϵ to the initial state of N, and every accepted state of N transitions to q_{end} via ϵ. State q_{end} becomes the new accepted state.

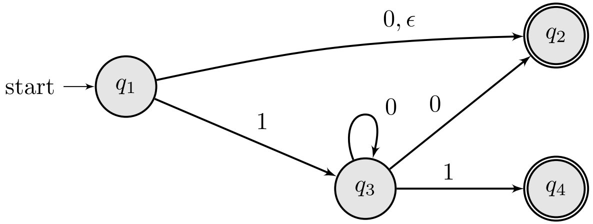
Input



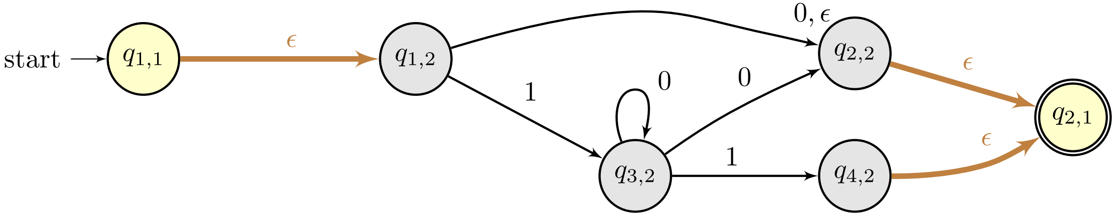
Step 1: wrap the NFA

> Given an NFA N, add two new states q_{start} and q_{end} such that q_{start} transitions via ϵ to the initial state of N, and every accepted state of N transitions to q_{end} via ϵ. State q_{end} becomes the new accepted state.

Input



Output

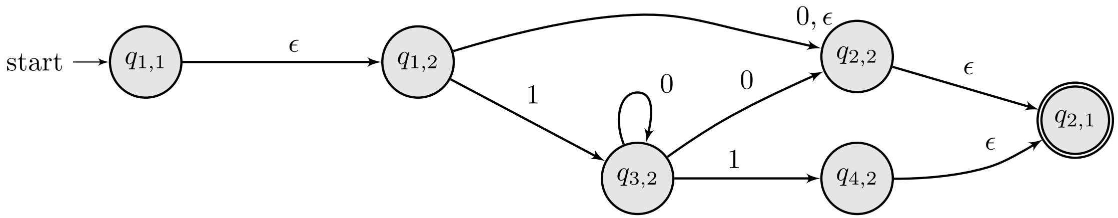


Step 2: Convert an NFA into a GNFA

A GNFA has regular expressions in the transitions, rather than the inputs.

> For every edge with $a_1, \ldots, a_n$ convert into $a_1 + \cdots + a_n$

Input

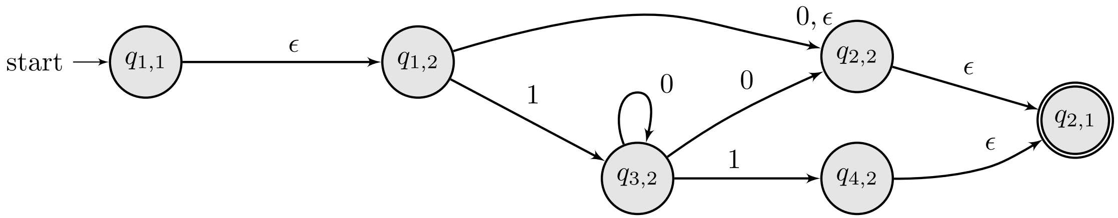


Step 2: Convert an NFA into a GNFA

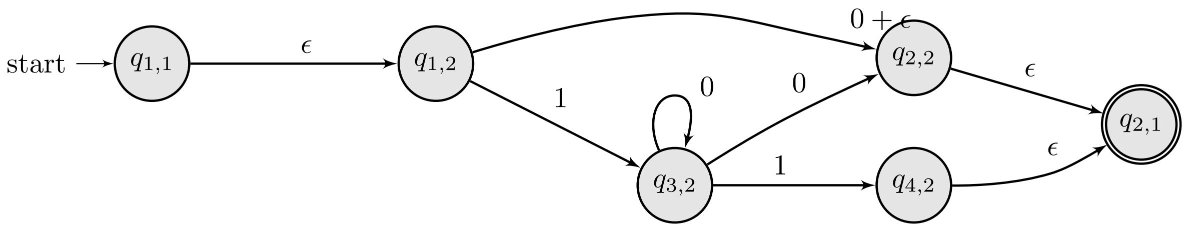
A GNFA has regular expressions in the transitions, rather than the inputs.

For every edge with $a_1, \ldots, a_n$ convert into $a_1 + \cdots + a_n$

Input



Output



Step 3: Reduce the GNFA

While there are more than 2 states:

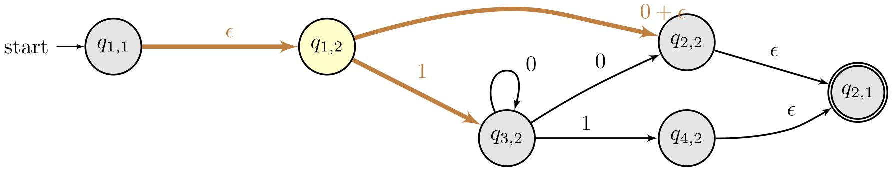
- pick a state and its incoming/outgoing edges, and convert it to transitions

Step 3.1: compress state $q_{1,2}$

$$\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{0+\epsilon} q_{2,2}) = q_{1,1} \xrightarrow{\epsilon\cdot(0+\epsilon)} q_{2,2}$$

$$\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{1} q_{3,2}) = q_{1,1} \xrightarrow{\epsilon\cdot 1} q_{3,2}$$

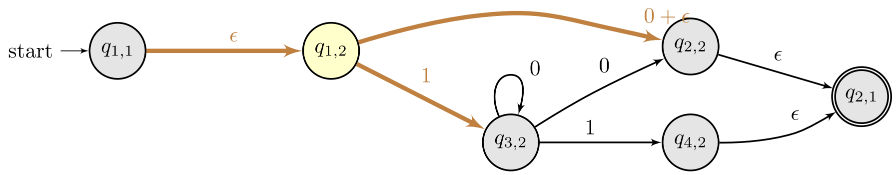
Input



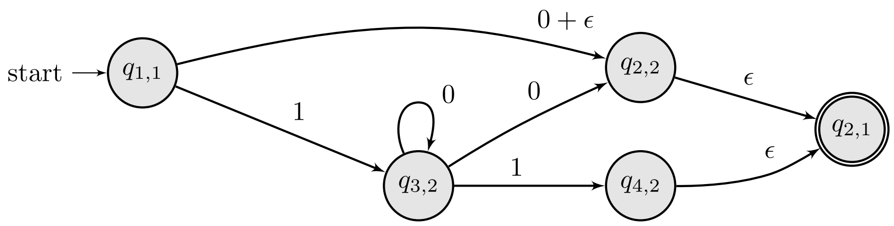
Step 3.1: compress state $q_{1,2}$

$$\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{0+\epsilon} q_{2,2}) = q_{1,1} \xrightarrow{\epsilon\cdot(0+\epsilon)} q_{2,2}$$
$$\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{1} q_{3,2}) = q_{1,1} \xrightarrow{\epsilon\cdot 1} q_{3,2}$$

Input



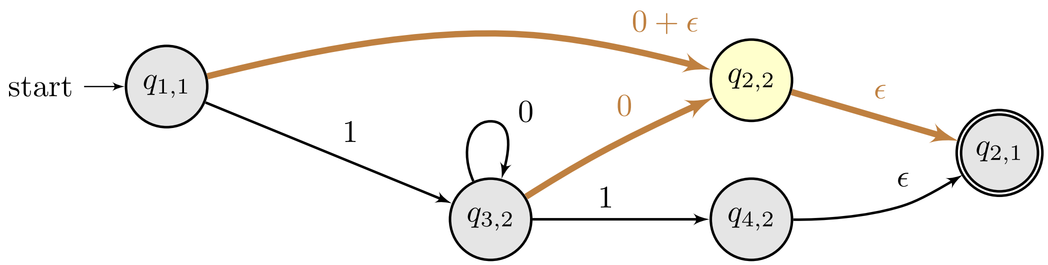
Output



> Each state that connects to $q_{1,2}$ must connect to every state that $q_{1,2}$ connects to. Som $q_{1,1}$ must connect with $q_{2,2}$ and $q_{1,1}$ must connect with $q_{3,2}$.

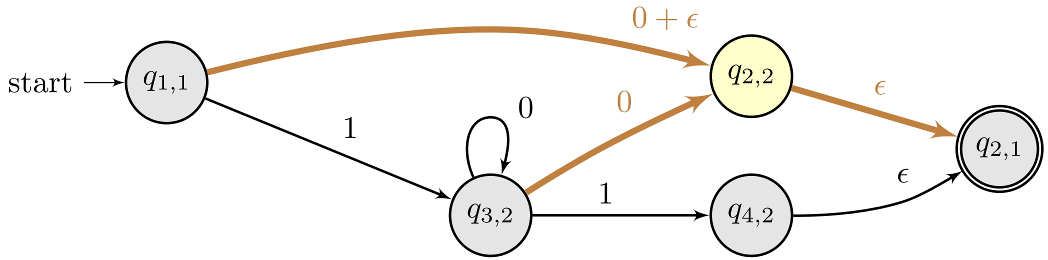
Step 3.2: compress state $q_{2,2}$

Input

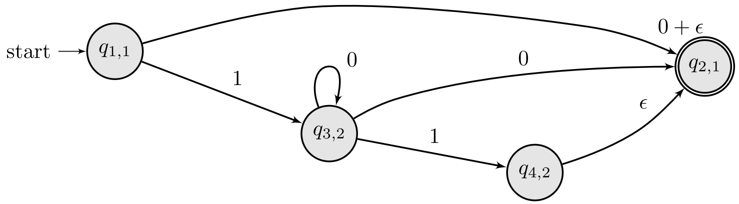


Step 3.2: compress state $q_{2,2}$

Input



Output



$$\text{compress}(q_{1,1} \xrightarrow{0+\epsilon} q_{2,2} \xrightarrow{\epsilon} q_{2,1}) = q_{1,1} \xrightarrow{(0+\epsilon)\cdot\epsilon} q_{2,2}$$
$$\text{compress}(q_{3,2} \xrightarrow{0} q_{2,2} \xrightarrow{\epsilon} q_{2,1}) = q_{3,2} \xrightarrow{0\cdot\epsilon} q_{2,1}$$

> Each state that connects to $q_{2,2}$ must connect to every state that $q_{2,2}$ connects to. Som $q_{1,1}$ must connect with $q_{2,1}$ and $q_{3,2}$ must connect with $q_{2,1}$.

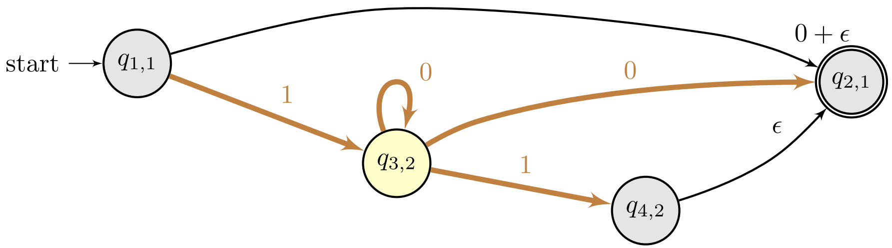
Step 3.3: compress state $q_{3,2}$

After compressing a state, we must merge the new node with any old node (in red).

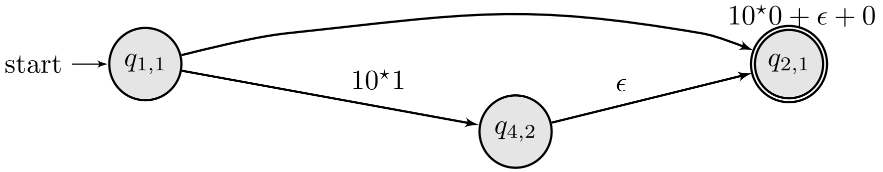
$$\text{compress}(q_{1,1} \xrightarrow{1} q_{3,2} \xrightarrow{0} q_{3,2} \xrightarrow{0} q_{2,1}) + q_{1,1} \xrightarrow{0+\epsilon} q_{2,1} = q_{1,1} \xrightarrow{(10^\star 0)+(0+\epsilon)} q_{2,2}$$

$$\text{compress}(q_{1,1} \xrightarrow{1} q_{3,2} \xrightarrow{0} q_{3,2} \xrightarrow{1} q_{4,2}) = q_{3,2} \xrightarrow{10^\star 1} q_{2,1}$$

Input



Output

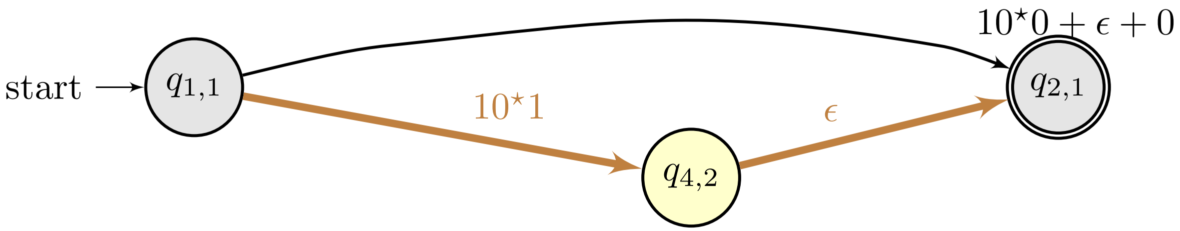


Step 3.3: compress state $q_{4,2}$

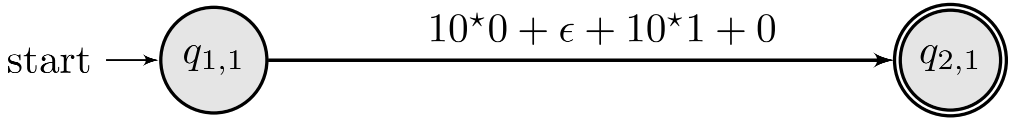
After compressing a state, we must merge the new node with any old node (in red).

$$\text{compress}(q_{1,1} \xrightarrow{10^\star 1} q_{4,2} \xrightarrow{\epsilon} q_{2,1}) + q_{1,1} \xrightarrow{10^\star 1 + 0 + \epsilon} q_{2,1} = q_{1,1} \xrightarrow{\left(10^\star 1 \cdot \epsilon\right) + \left(10^\star 0 + 0 + \epsilon\right)} q_{2,2}$$

Input



Output

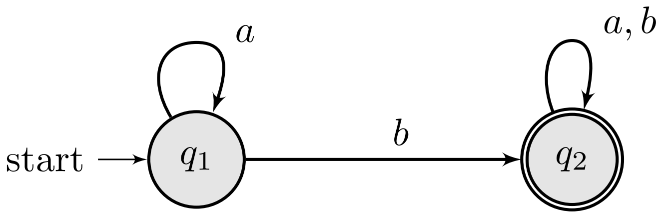


Result: $10^\star 1 + 10^\star 0 + 0 + \epsilon$

Exercise 1.66

Convert an NFA into a Regex

1. Convert the NFA into an NFA (same)

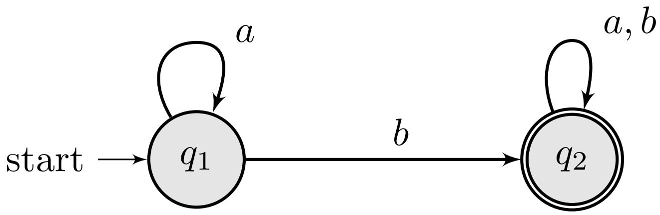


2. Wrap the NFA

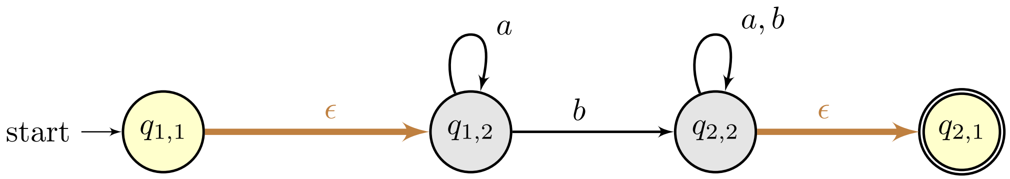
Exercise 1.66

Convert an NFA into a Regex

1. Convert the NFA into an NFA (same)



2. Wrap the NFA

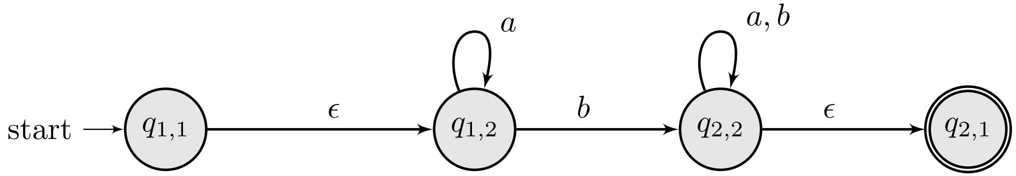


Exercise 1.66

Convert an NFA into a Regex

3. Convert NFA into GNFA

Before

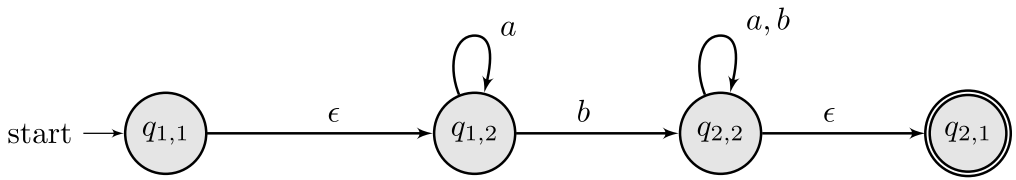


Exercise 1.66

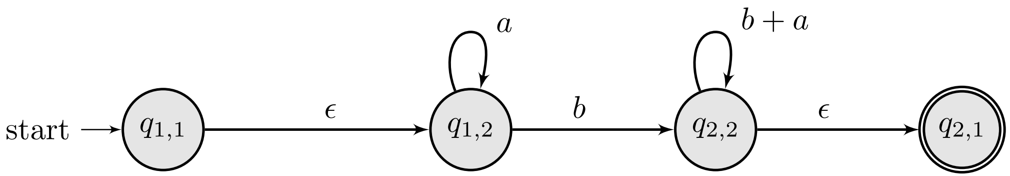
Convert an NFA into a Regex

3. Convert NFA into GNFA

Before



After

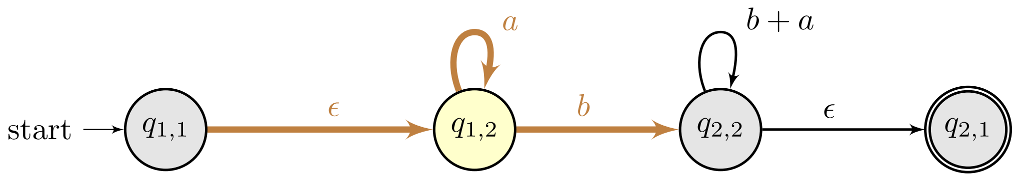


Exercise 1.66

Convert an NFA into a Regex

4. Compress state.

Before

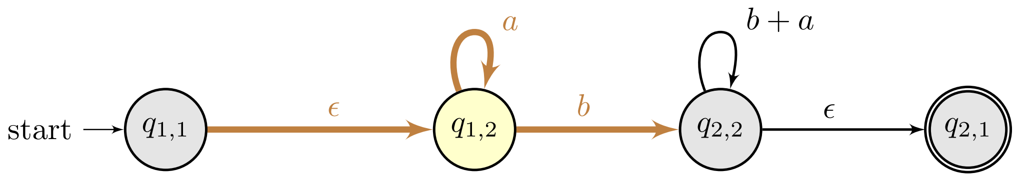


Exercise 1.66

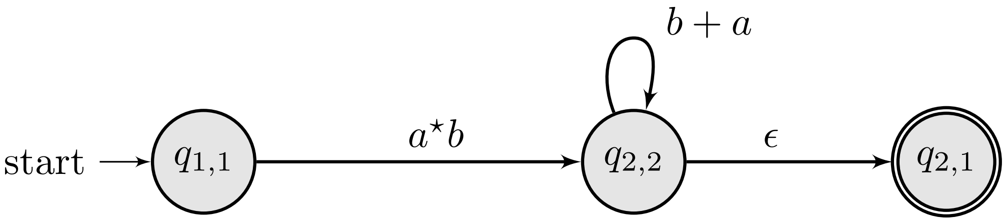
Convert an NFA into a Regex

4. Compress state.

Before



After

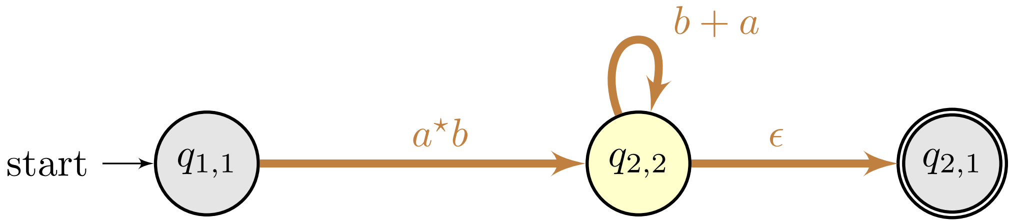


Exercise 1.66

Convert an NFA into a Regex

5. Compress state.

Before

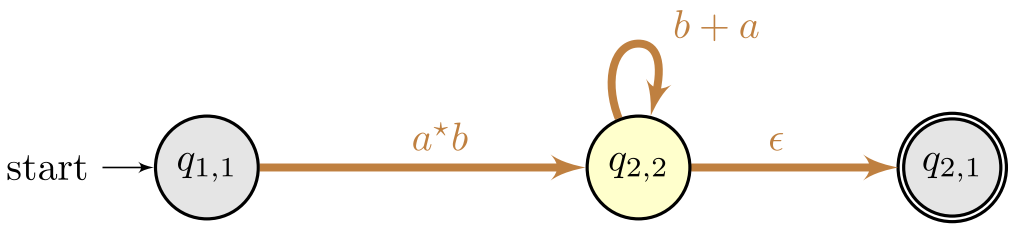


Exercise 1.66

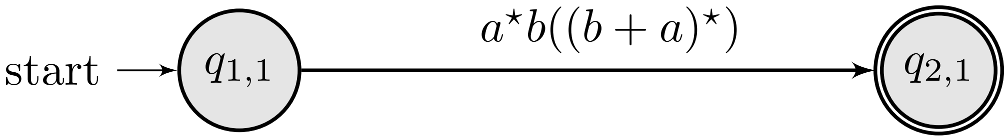
Convert an NFA into a Regex

5. Compress state.

Before



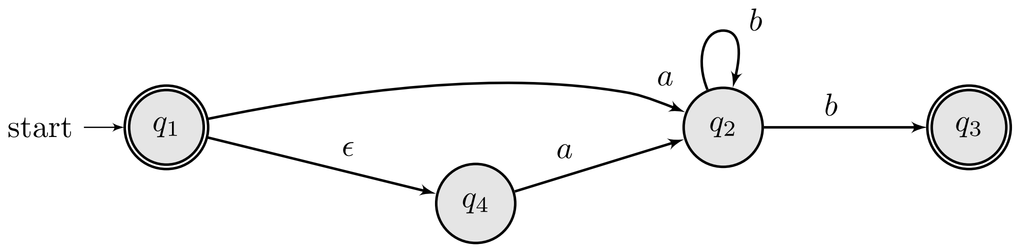
After



Exercise 8

Convert an NFA into a Regex

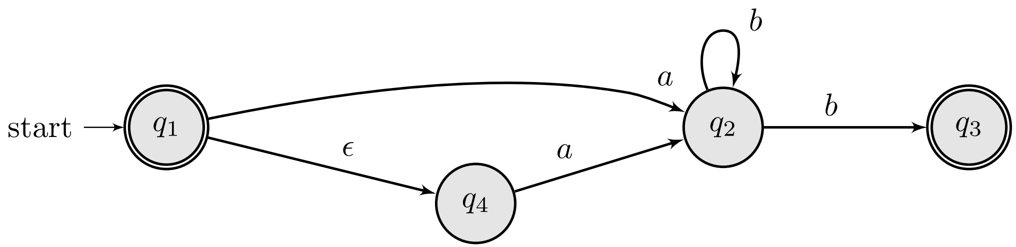
Before



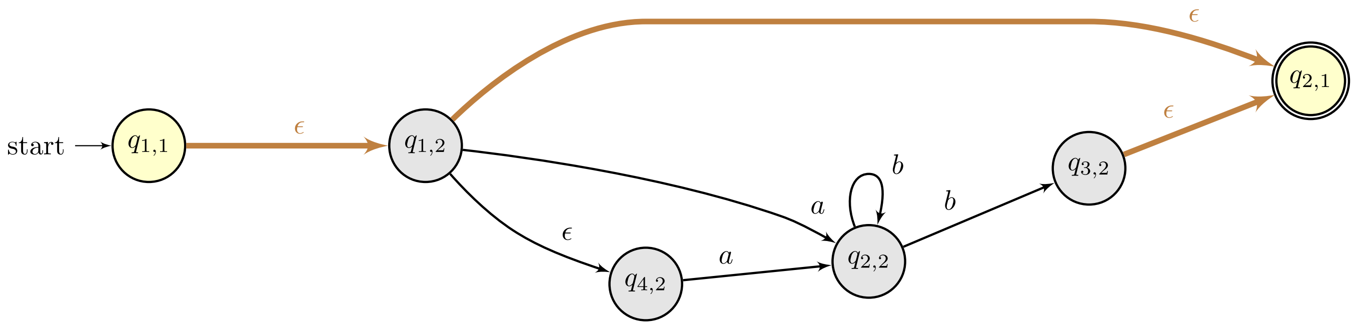
Exercise 8

Convert an NFA into a Regex

Before



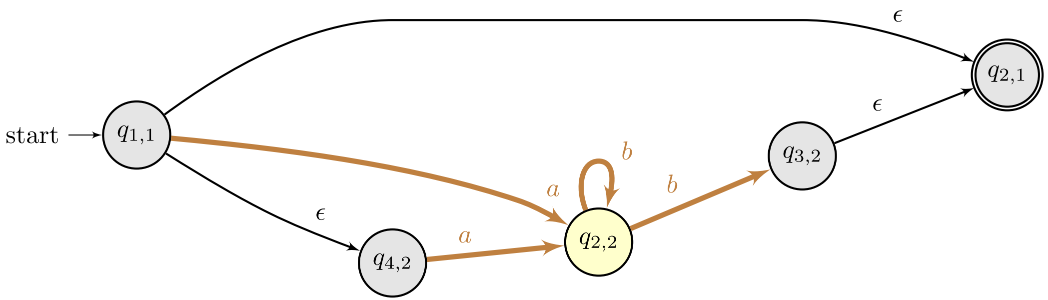
After



Exercise 8

Convert an NFA into a Regex

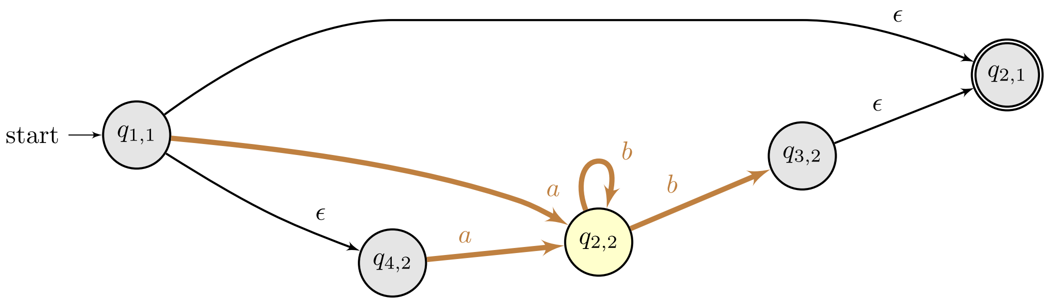
Before



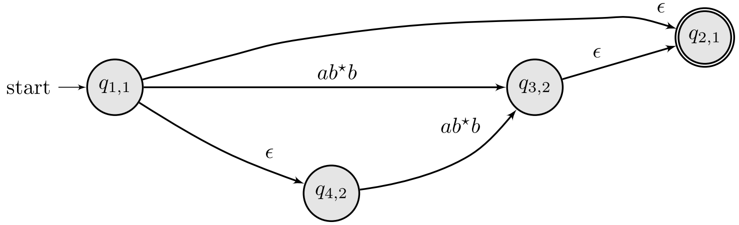
Exercise 8

Convert an NFA into a Regex

Before



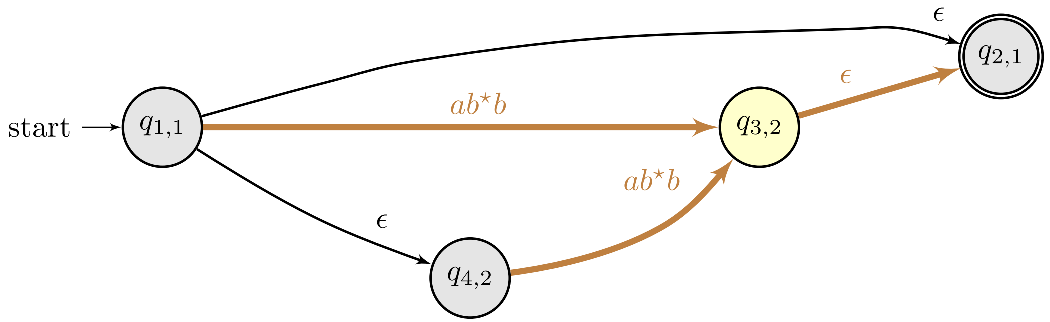
After



Exercise 8

Convert an NFA into a Regex

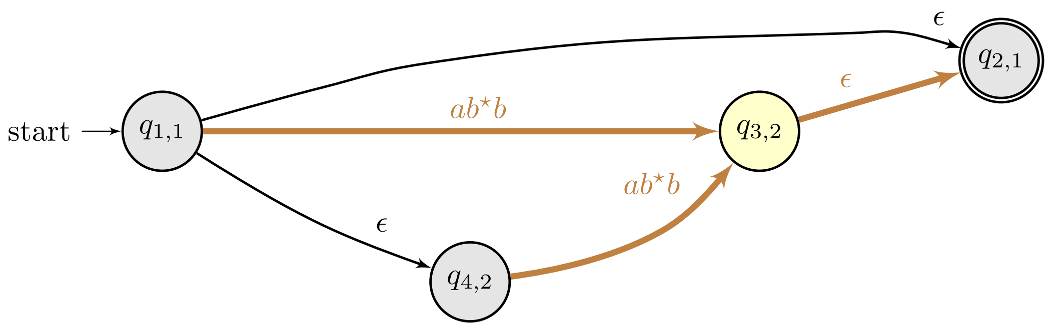
Before



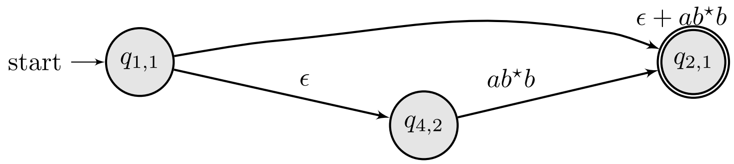
Exercise 8

Convert an NFA into a Regex

Before



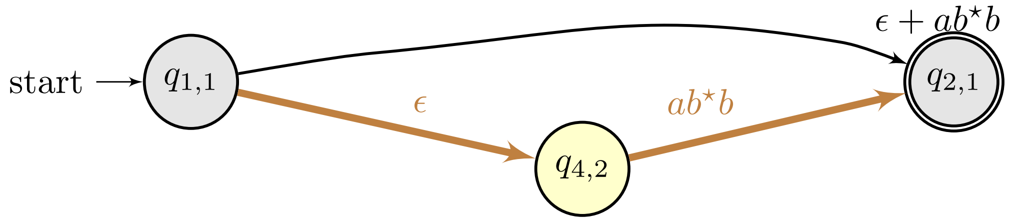
After



Exercise 8

Convert an NFA into a Regex

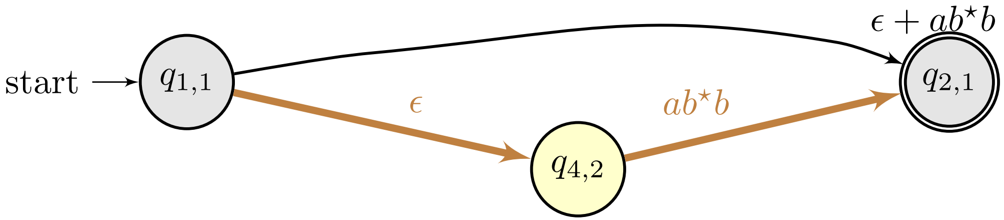
Before



Exercise 8

Convert an NFA into a Regex

Before



After