Pre-lecture brain teaser

Prove at the following languages are regular:

- All strings that end in 1011

- All strings that contain 101 or 010 as a substring.

- All strings that do **not** contain 111 as a substring.

ECE-374-B: Lecture 4 - RegExp-DFA-NFA Equivalence

Instructor: Nickvash Kani

September 05, 2023

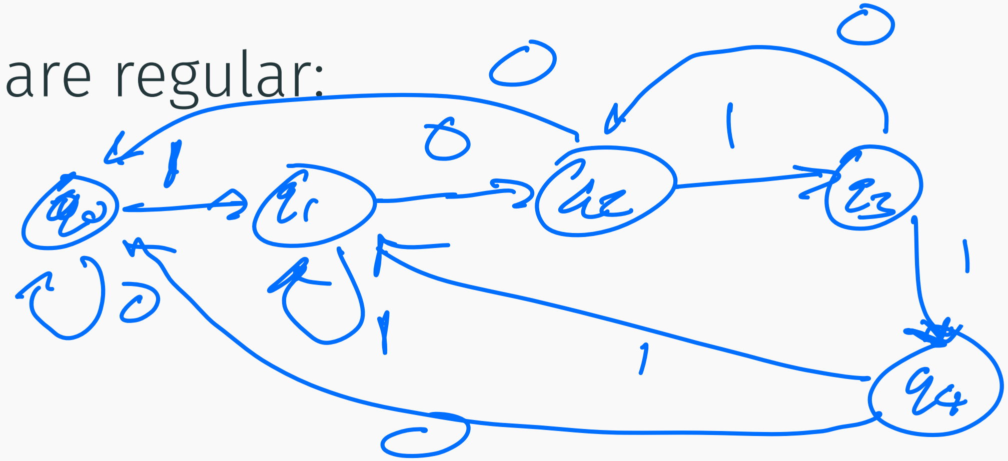
University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Prove at the following languages are regular:

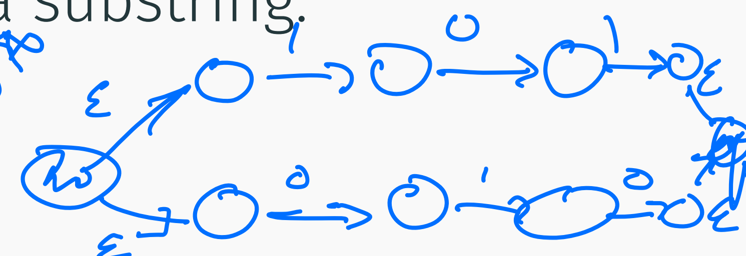
- All strings that end in 1011

$(0+1)^* \cdot 1011$

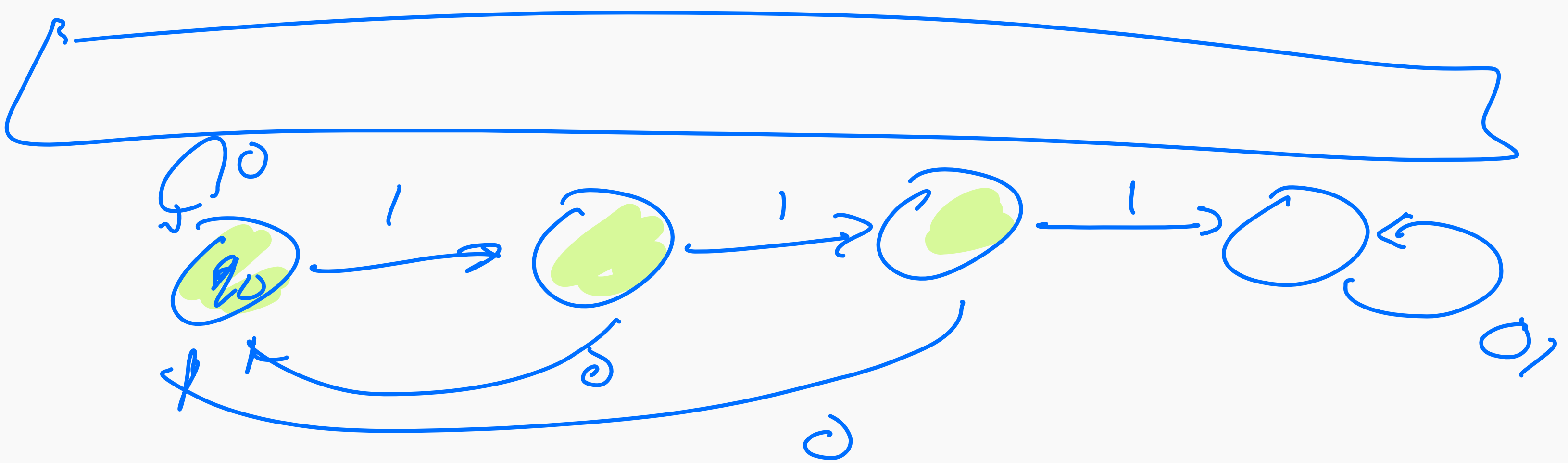


- All strings that contain 101 or 010 as a substring.

$(0+1)^*(101+010)(0+1)^*$



- All strings that do not contain 111 as a substring.



Regular Languages, DFAs, NFAs

Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.

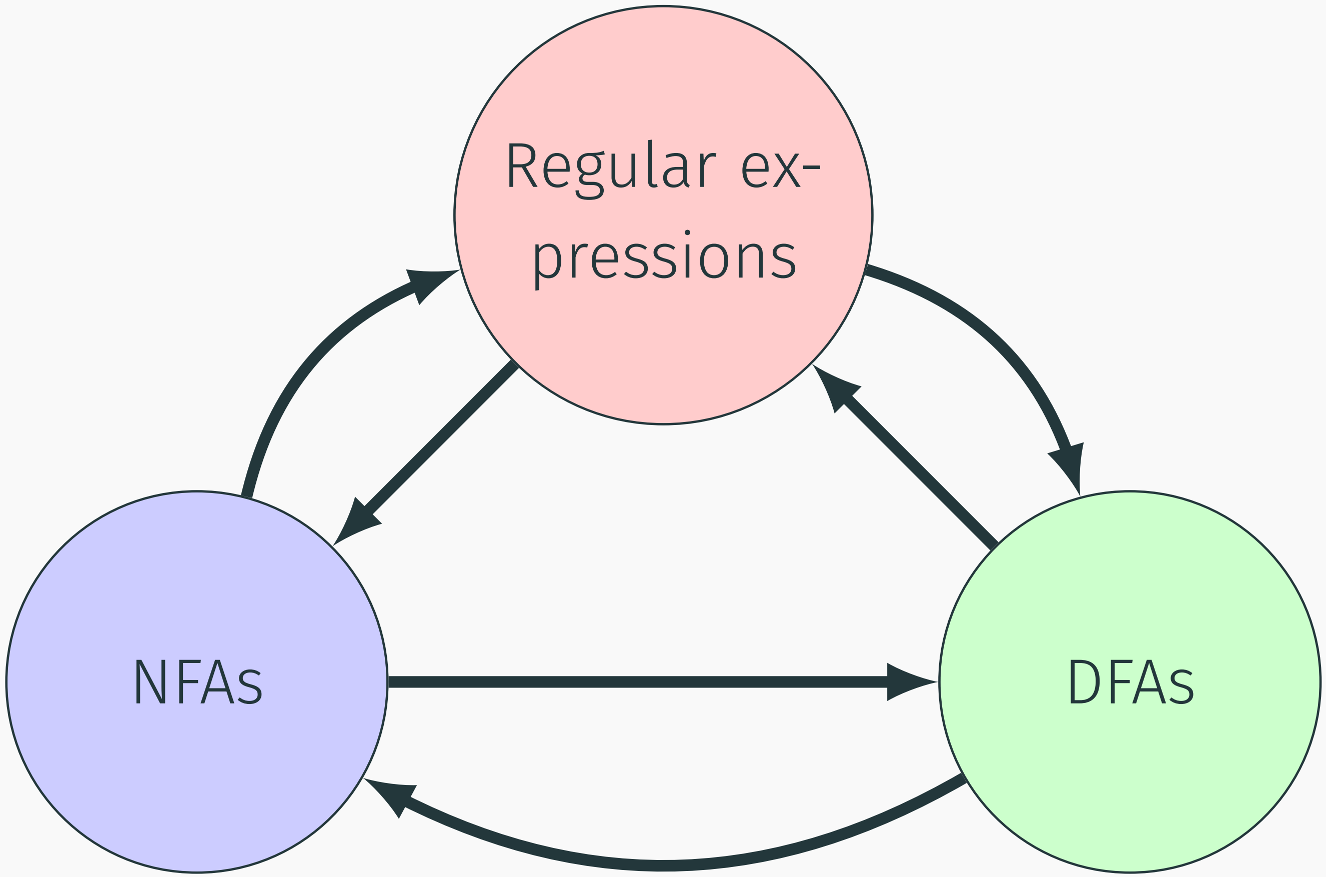
Regular Languages, DFAs, NFAs

Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.

- DFAs are special cases of NFAs (easy)
- NFAs accept regular expressions (seen)
- DFAs accept languages accepted by NFAs (shortly)
- Regular expressions for languages accepted by DFAs (shown previously)

Regular Languages, DFAs, NFAs

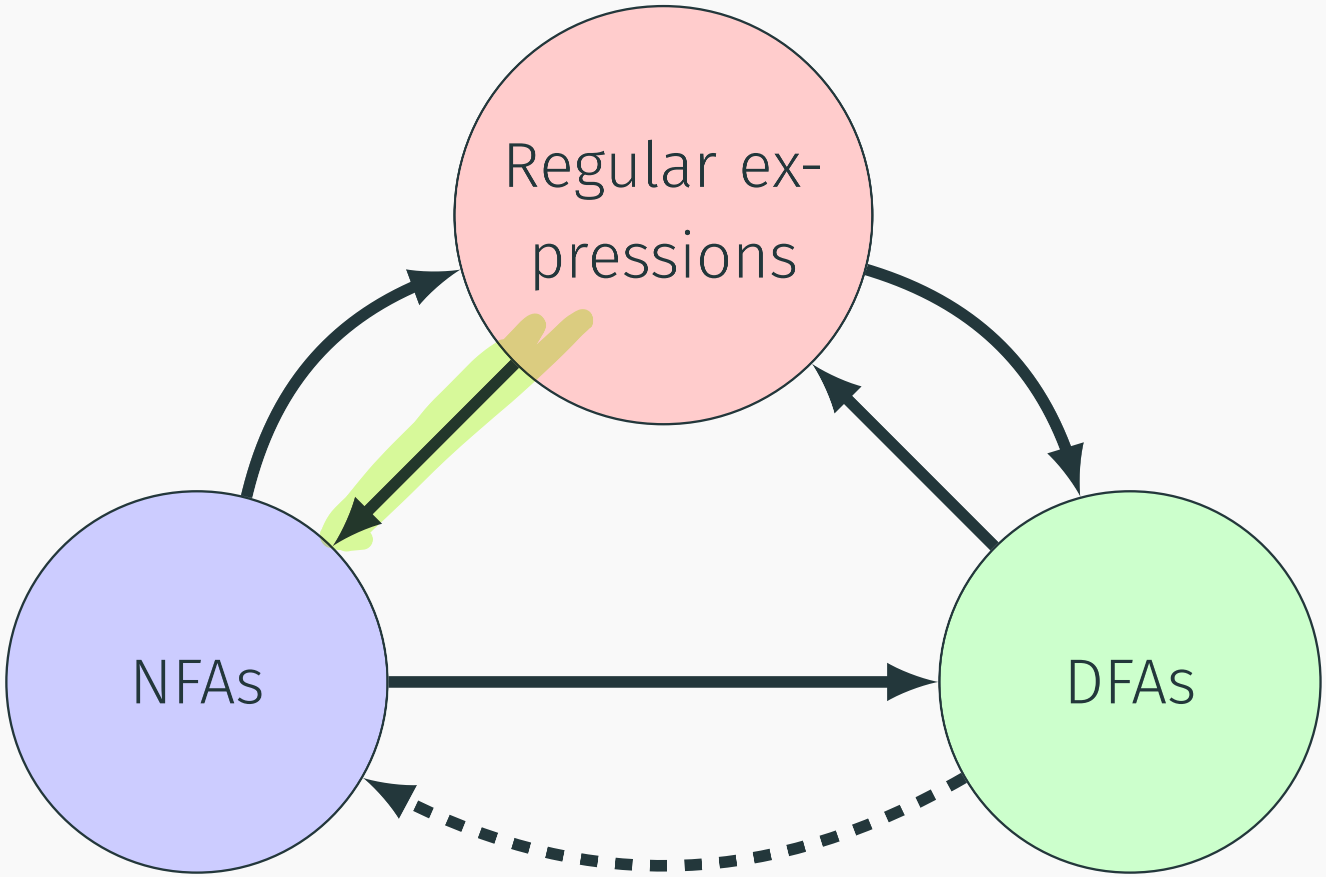
Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.



Regular Languages, DFAs, NFAs

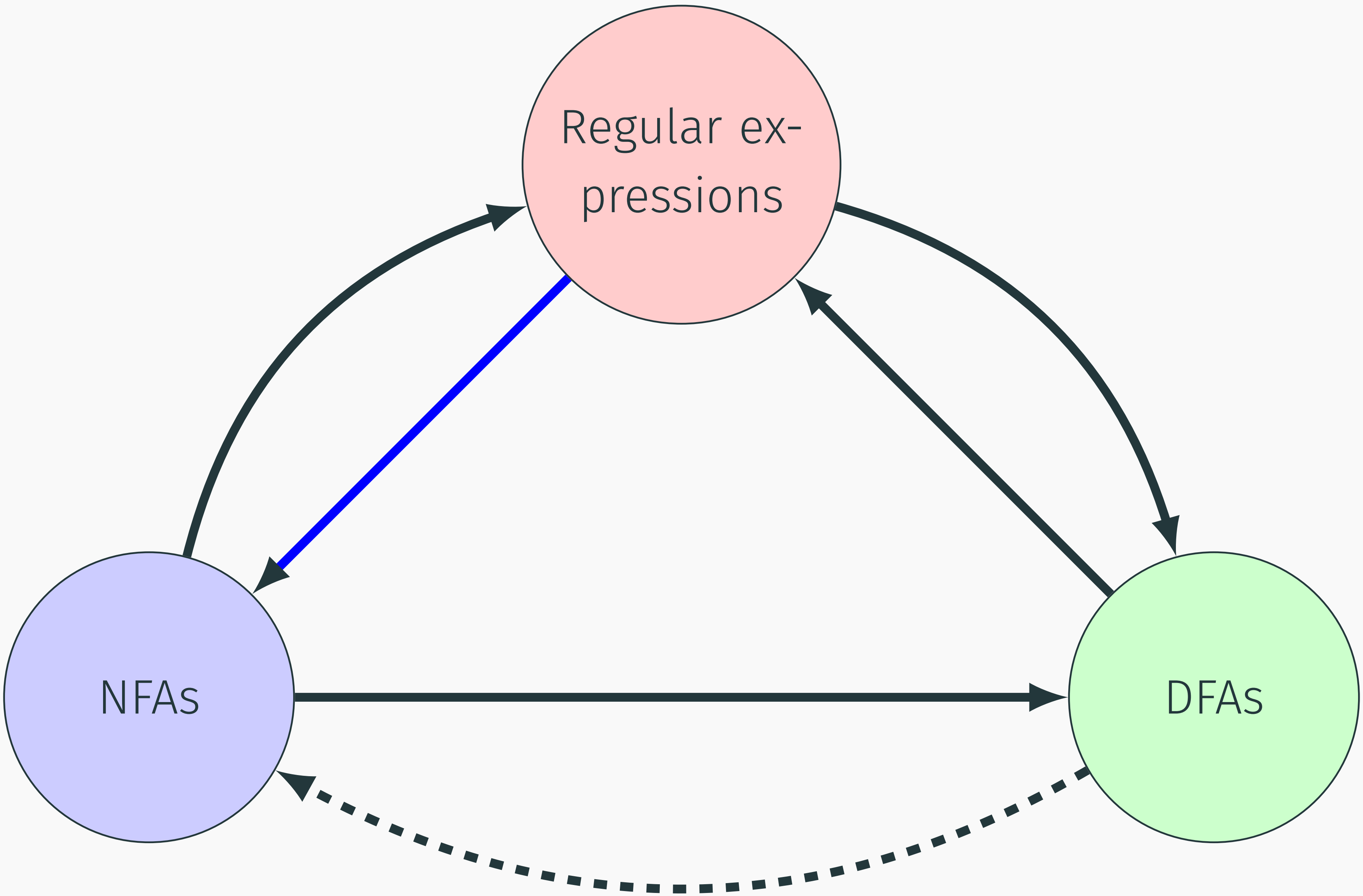
Theorem

Languages accepted by DFAs, NFAs, and regular expressions are the same.



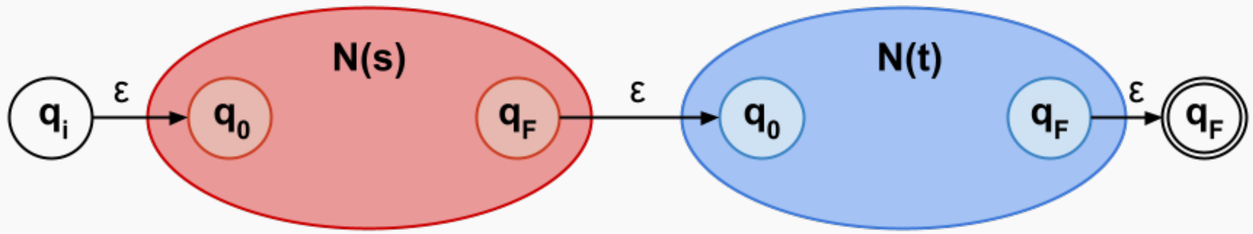
Regular Expression to NFA

Proving equivalence



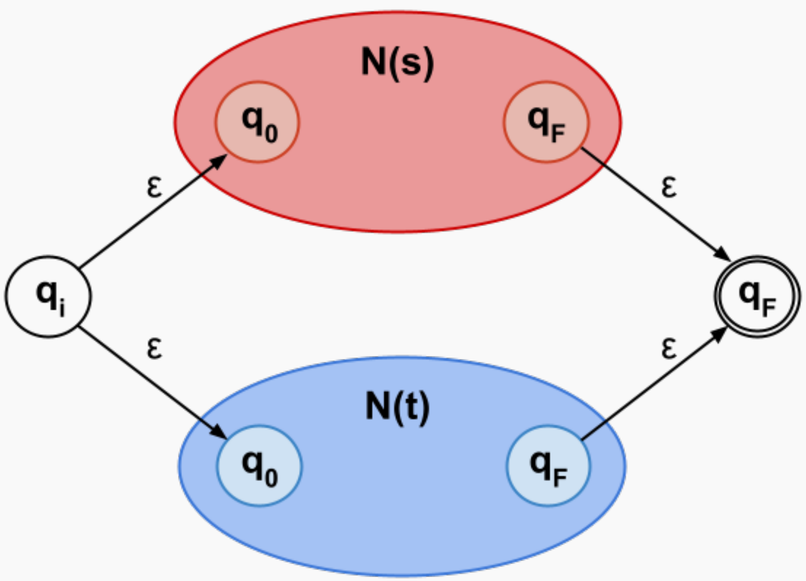
Thompson's algorithm

Given two NFAs s and t:

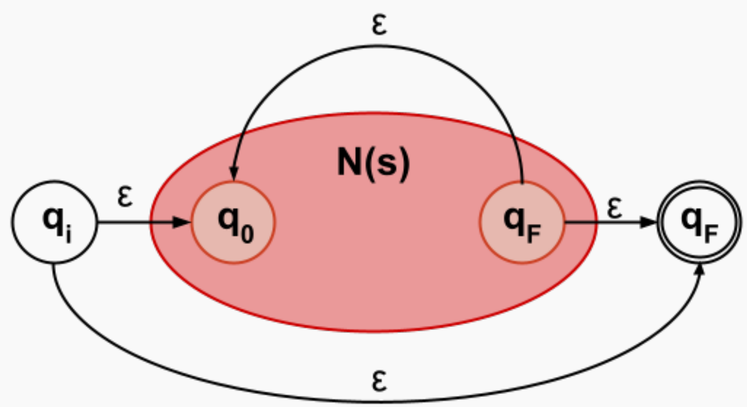


$L = L_s \cdot L_t$

$L = L_s \cup L_t$



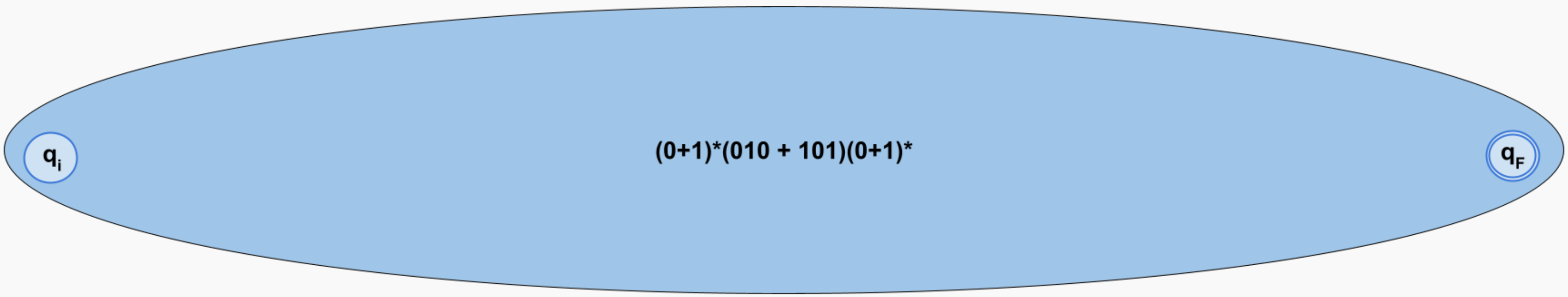
$L = (L_s)^*$



Regular expression to NFA example

Let's take a regular expression and convert it to a DFA.

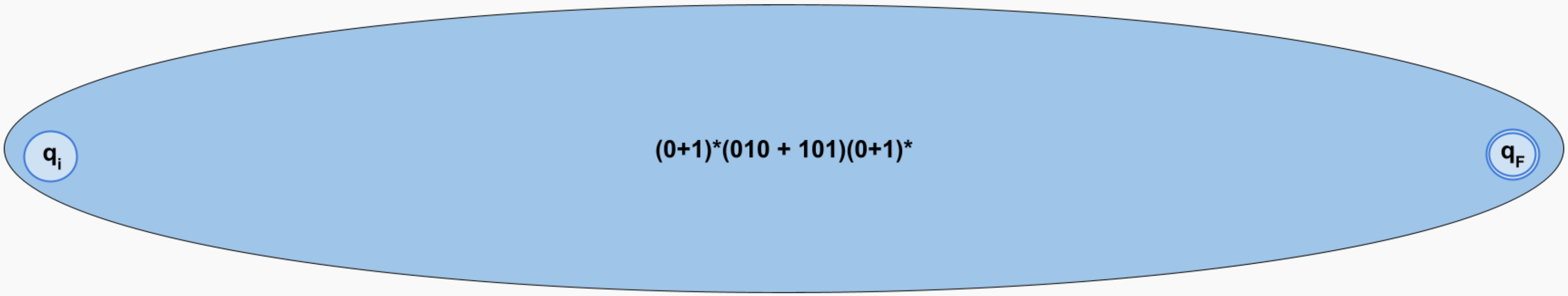
Example: $(0+1)^*(101+010)(0+1)^*$



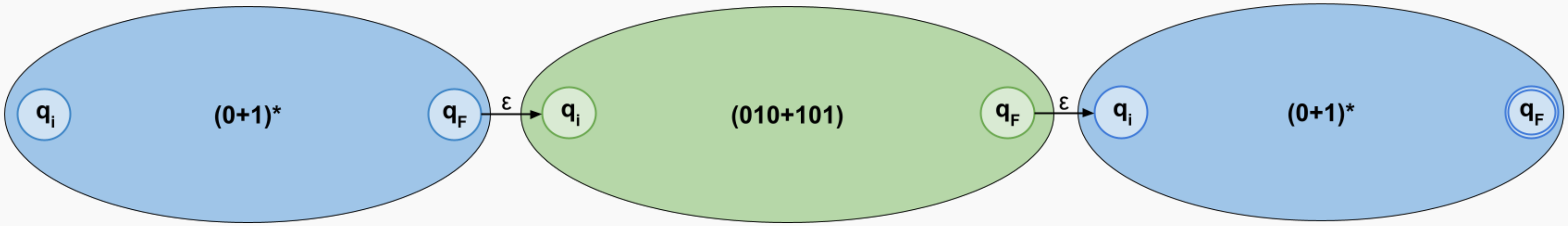
Regular expression to NFA example

Let's take a regular expression and convert it to a DFA.

Example: $(0 + 1)^*(101 + 010)(0 + 1)^*$

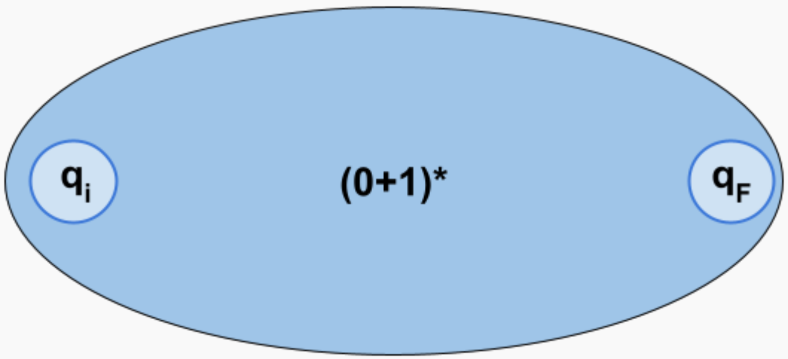


Using the concatenation rule:



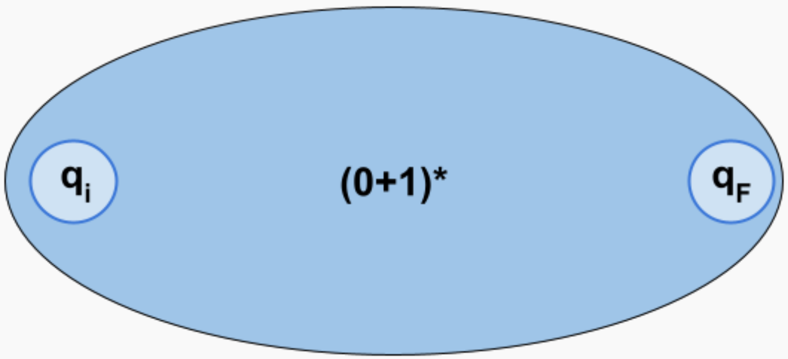
Regular expression to NFA example

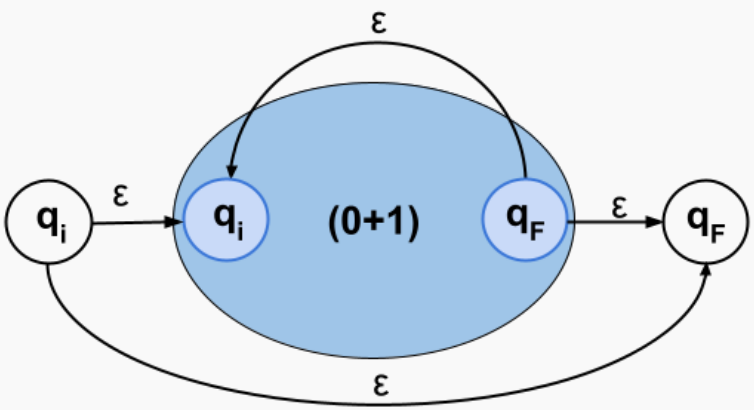
Find NFA for $(0 + 1)^*$



Regular expression to NFA example

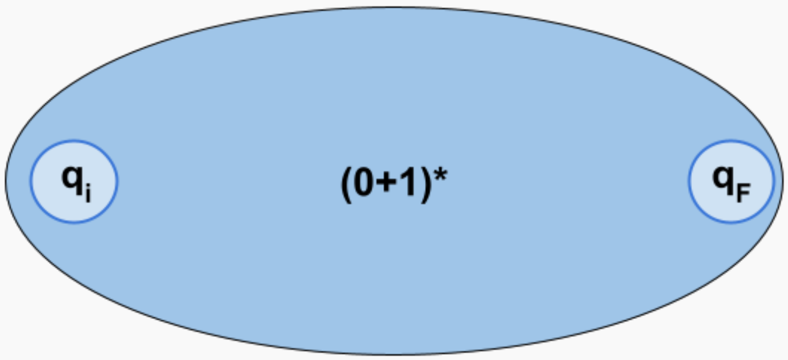
Find NFA for $(0 + 1)^*$

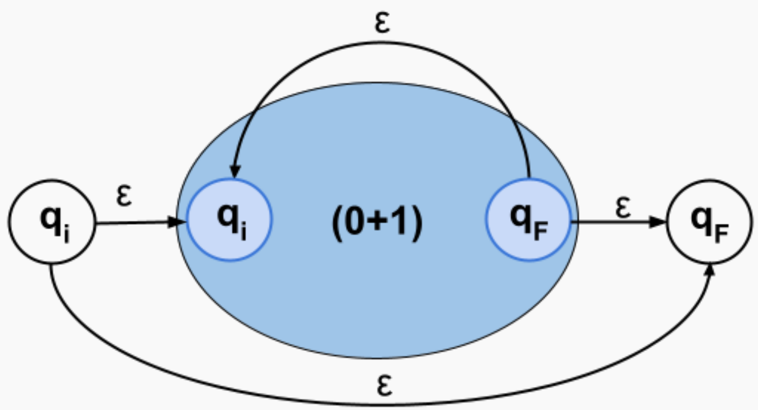


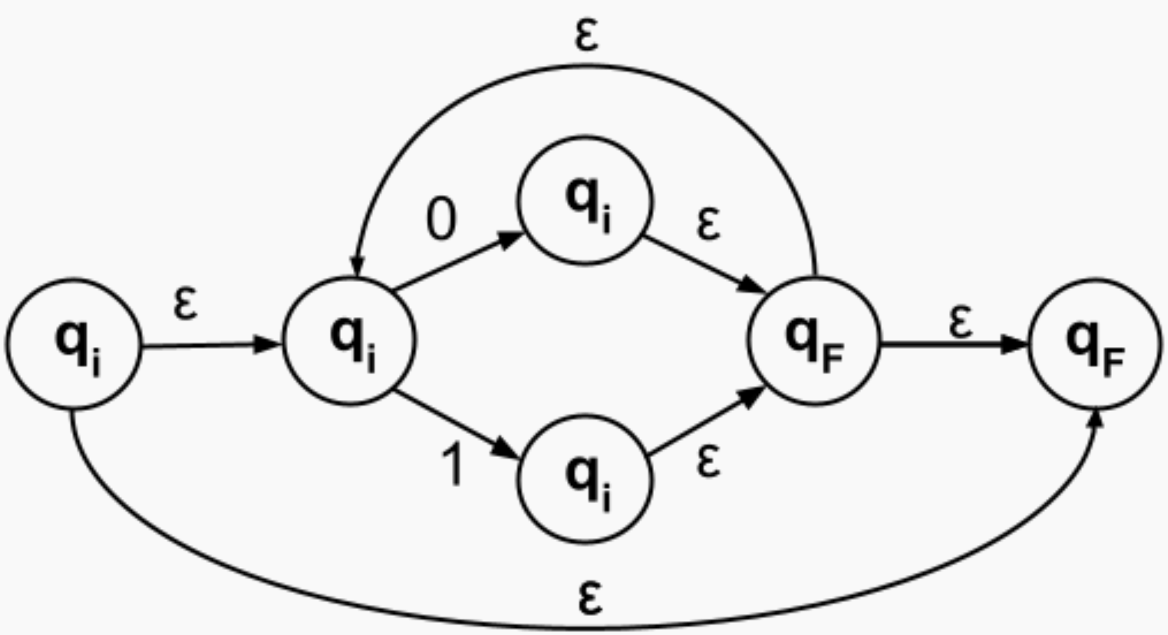


Regular expression to NFA example

Find NFA for $(0 + 1)^*$

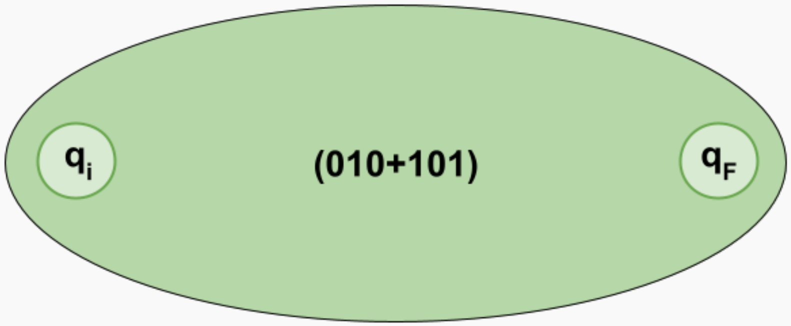






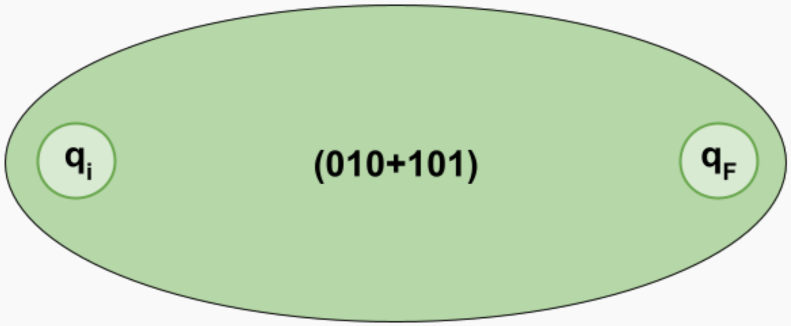
Regular expression to NFA example

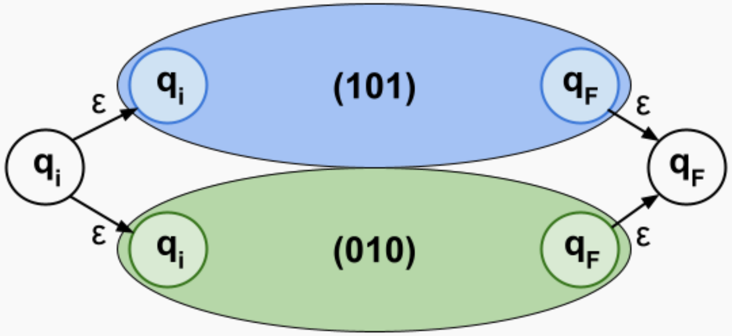
Find NFA for $(101 + 010)$



Regular expression to NFA example

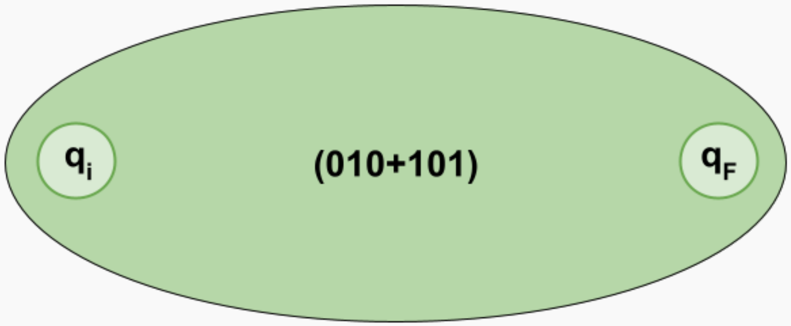
Find NFA for $(101 + 010)$

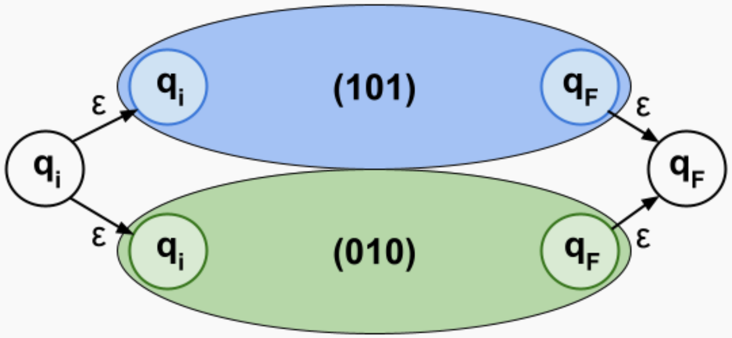


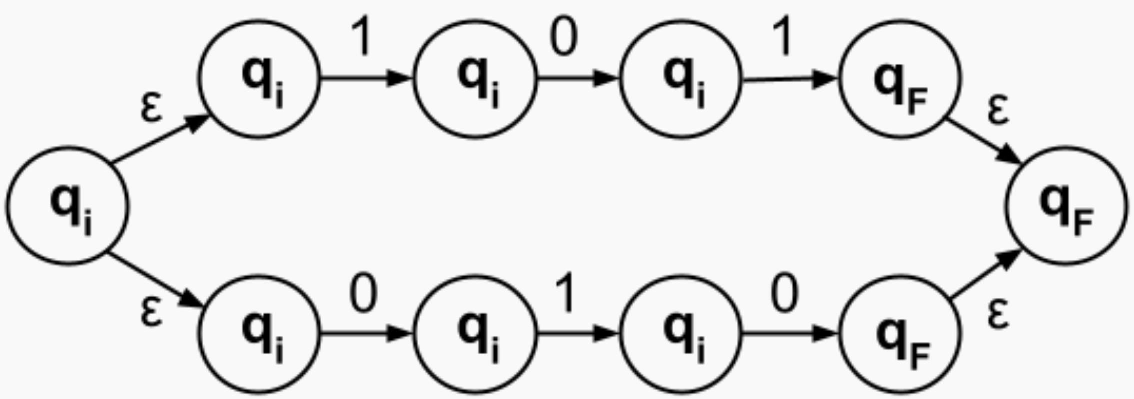


Regular expression to NFA example

Find NFA for $(101 + 010)$



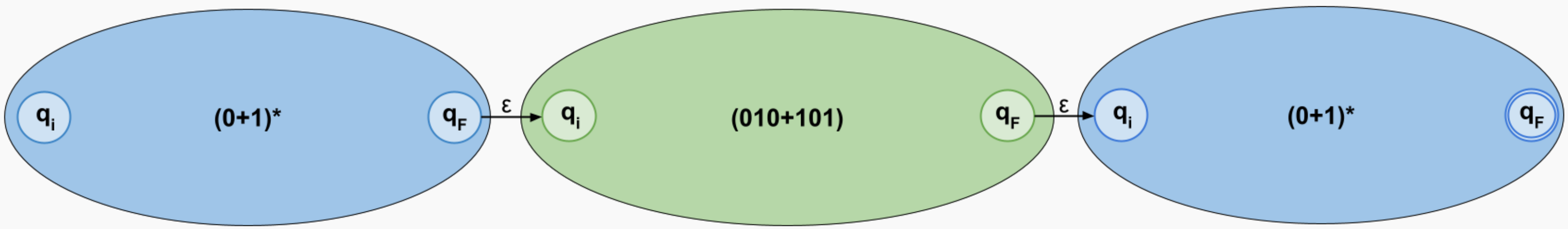




Regular expression to NFA example

Let's take a regular expression and convert it to a NFA.

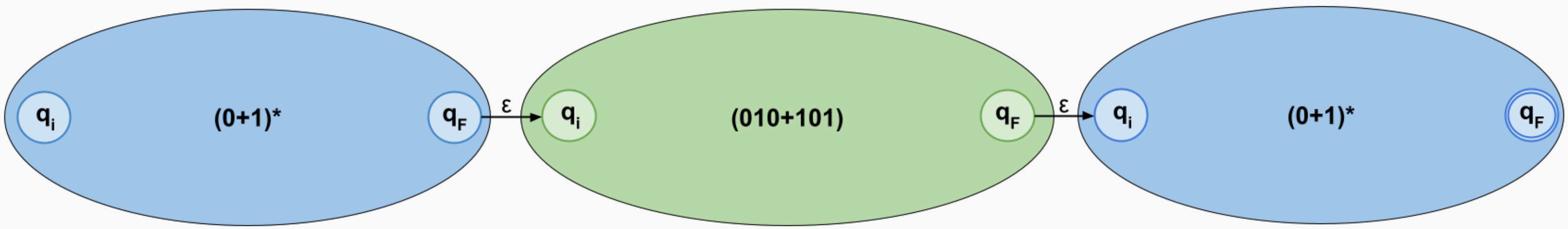
Example: $(0+1)^*(101+010)(0+1)^*$



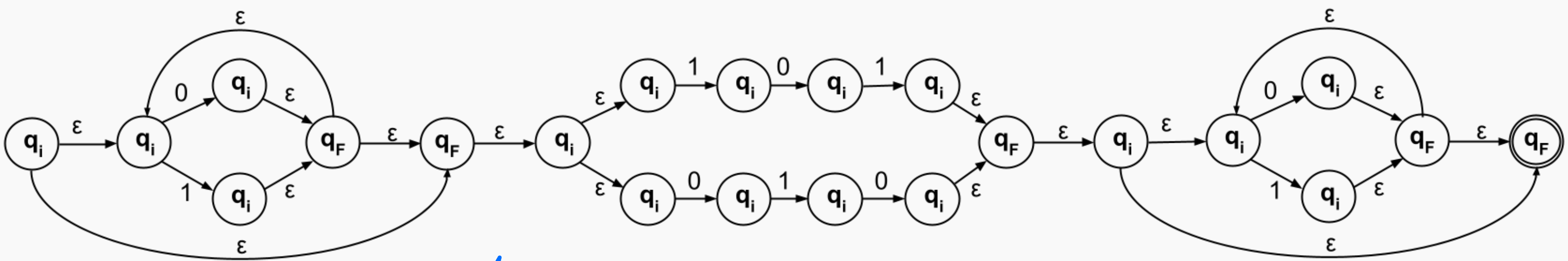
Regular expression to NFA example

Let’s take a regular expression and convert it to a NFA.

Example: $(0+1)^*(101+010)(0+1)^*$



Using the concatenation rule:

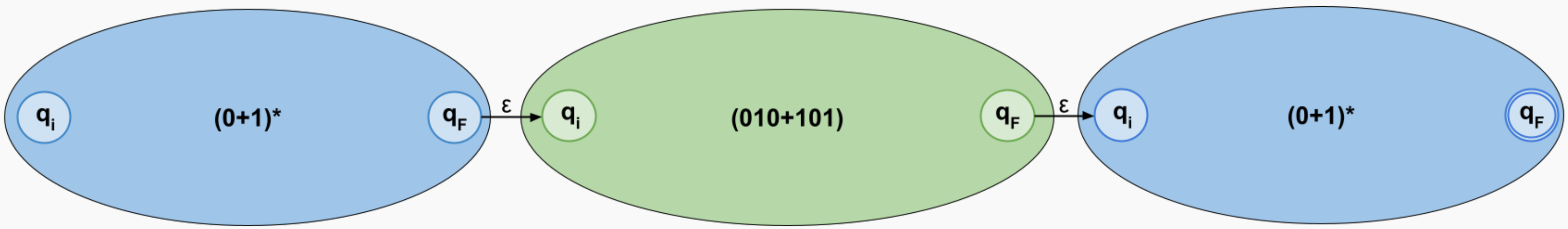


Thompson's Algorithm

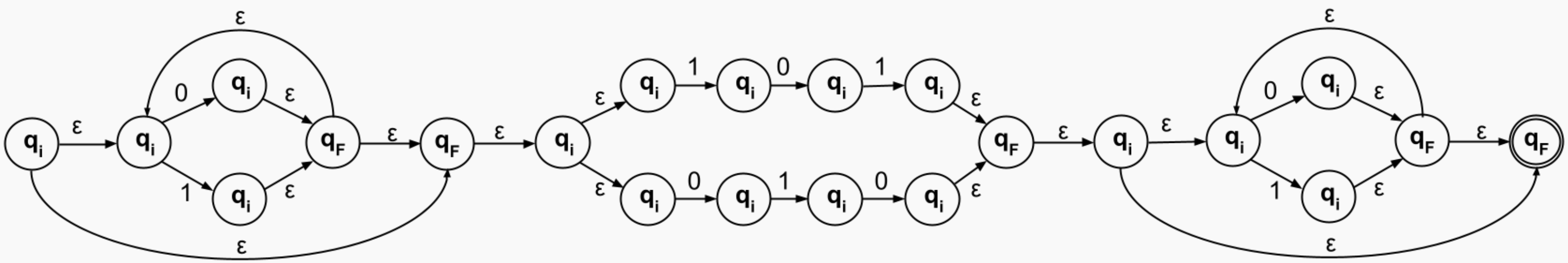
Regular expression to NFA example

Let's take a regular expression and convert it to a NFA.

Example: $(0+1)^*(101+010)(0+1)^*$



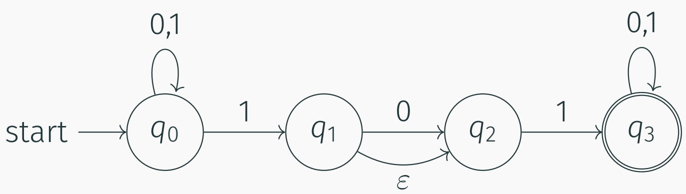
Using the concatenation rule:



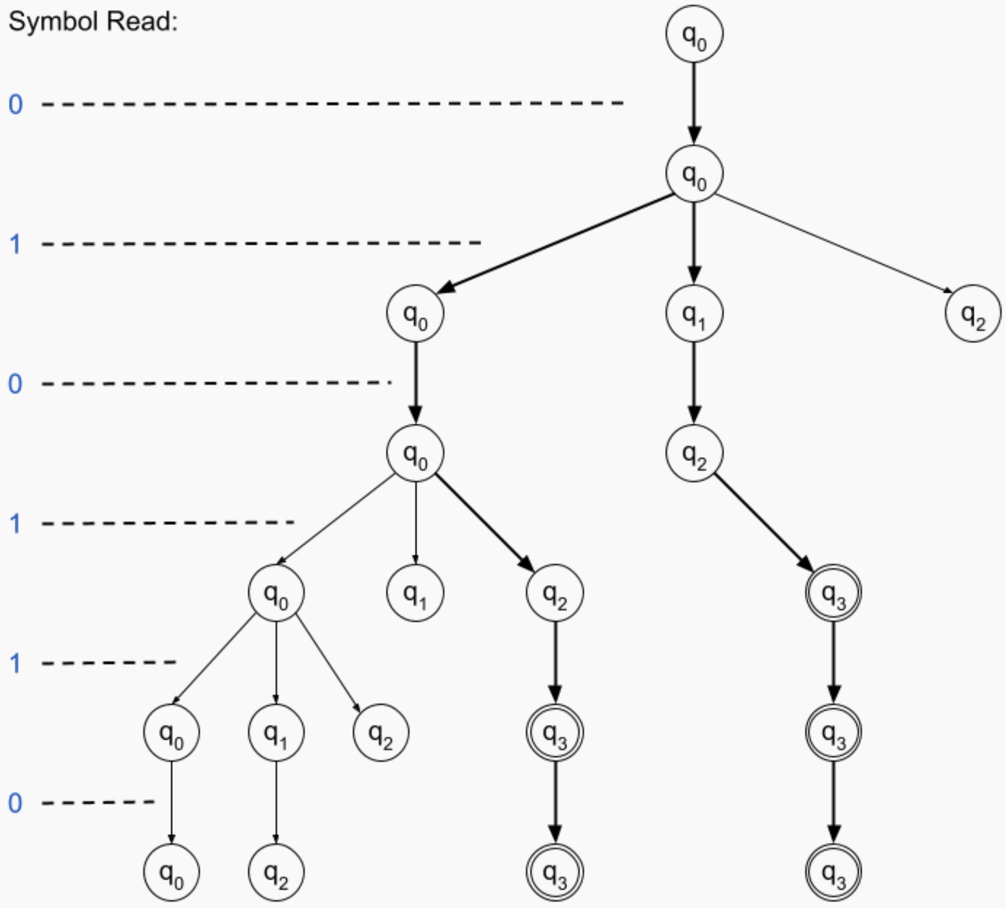
What does Thompson's algorithm mean?!

Equivalence of NFAs and DFAs

Another Way to look at NFAs

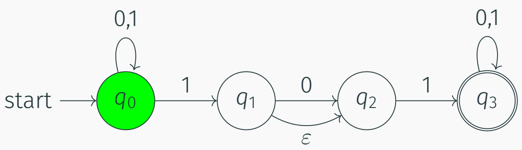


Is 010110 accepted?

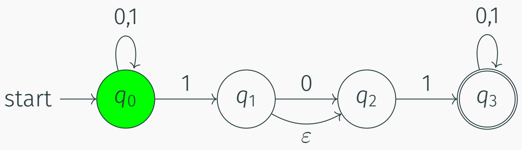


Another Way to look at NFAs

Is 010110 accepted?

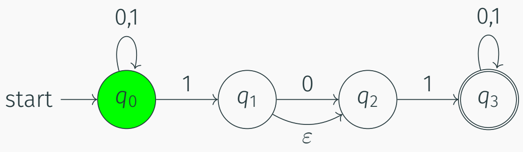


Another Way to look at NFAs

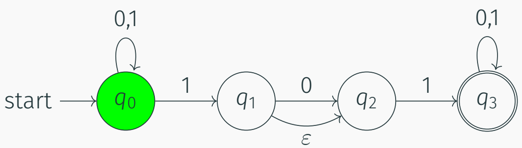


Is 010110 accepted?

0

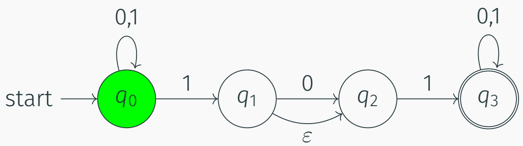


Another Way to look at NFAs

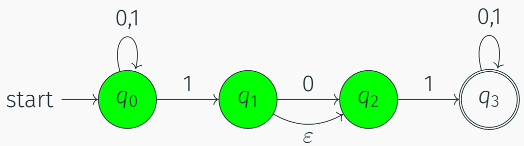


Is 010110 accepted?

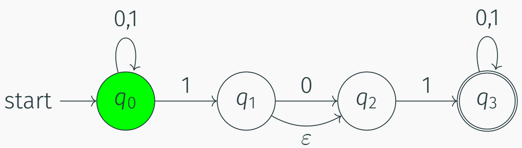
0



1

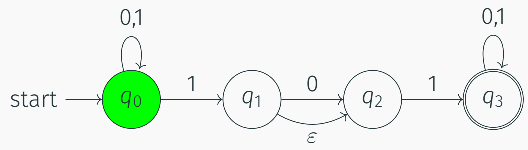


Another Way to look at NFAs

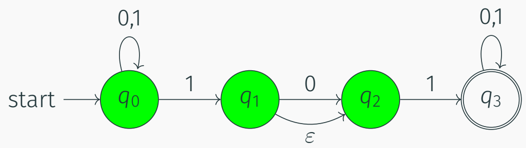


Is 010110 accepted?

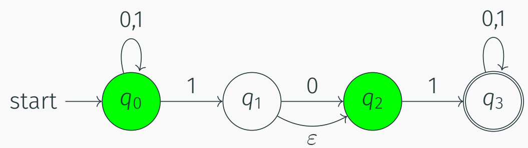
0



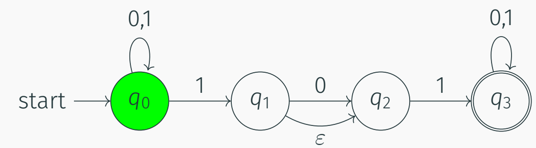
1



0

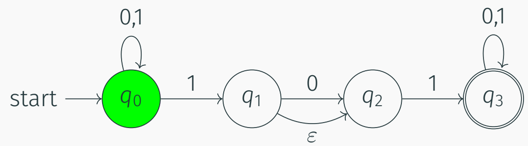


Another Way to look at NFAs

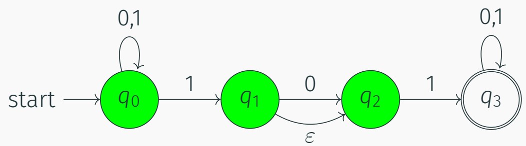


Is 010110 accepted?

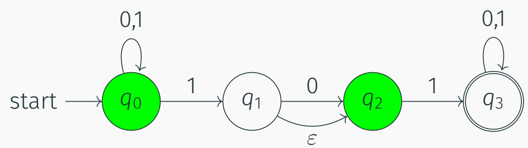
0



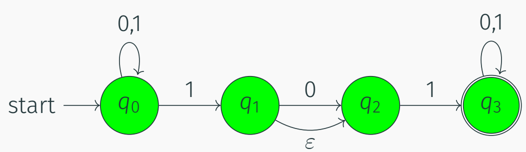
1



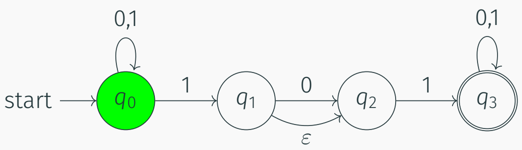
0



1

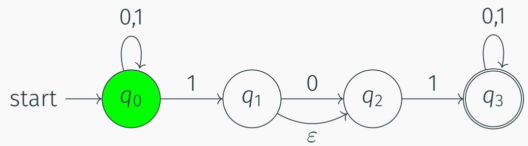


Another Way to look at NFAs

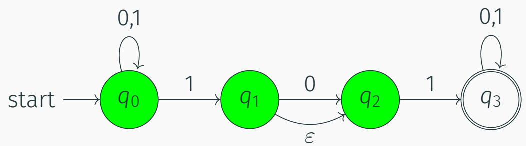


Is 010110 accepted?

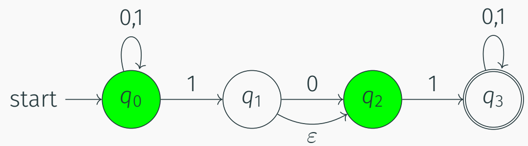
0



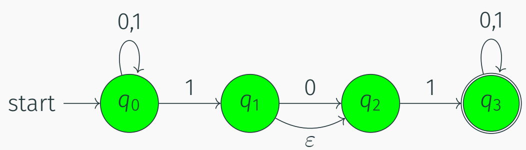
1



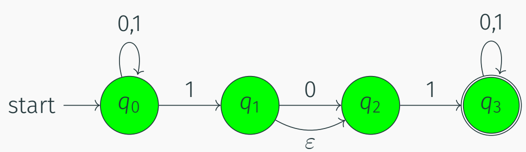
0



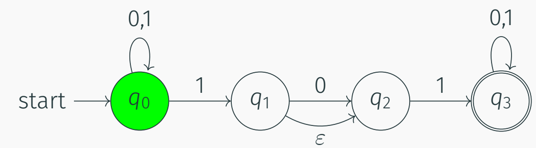
1



1

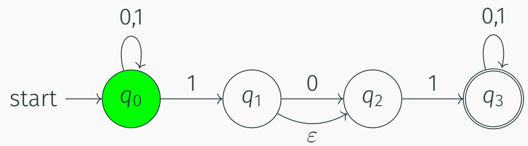


Another Way to look at NFAs

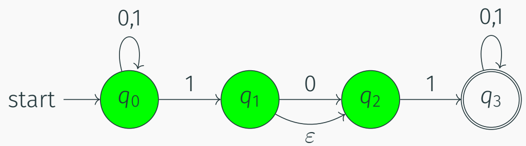


Is 010110 accepted?

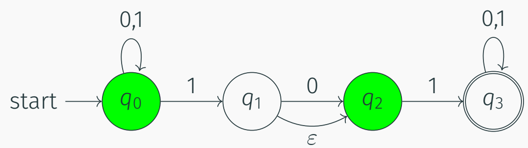
0



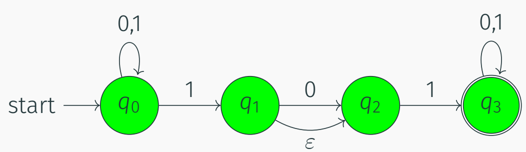
1



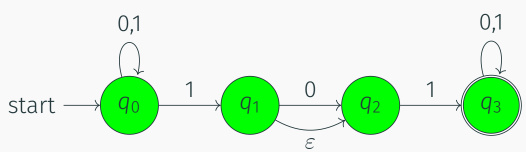
0



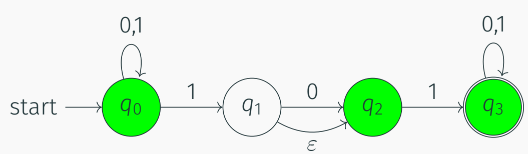
1



1

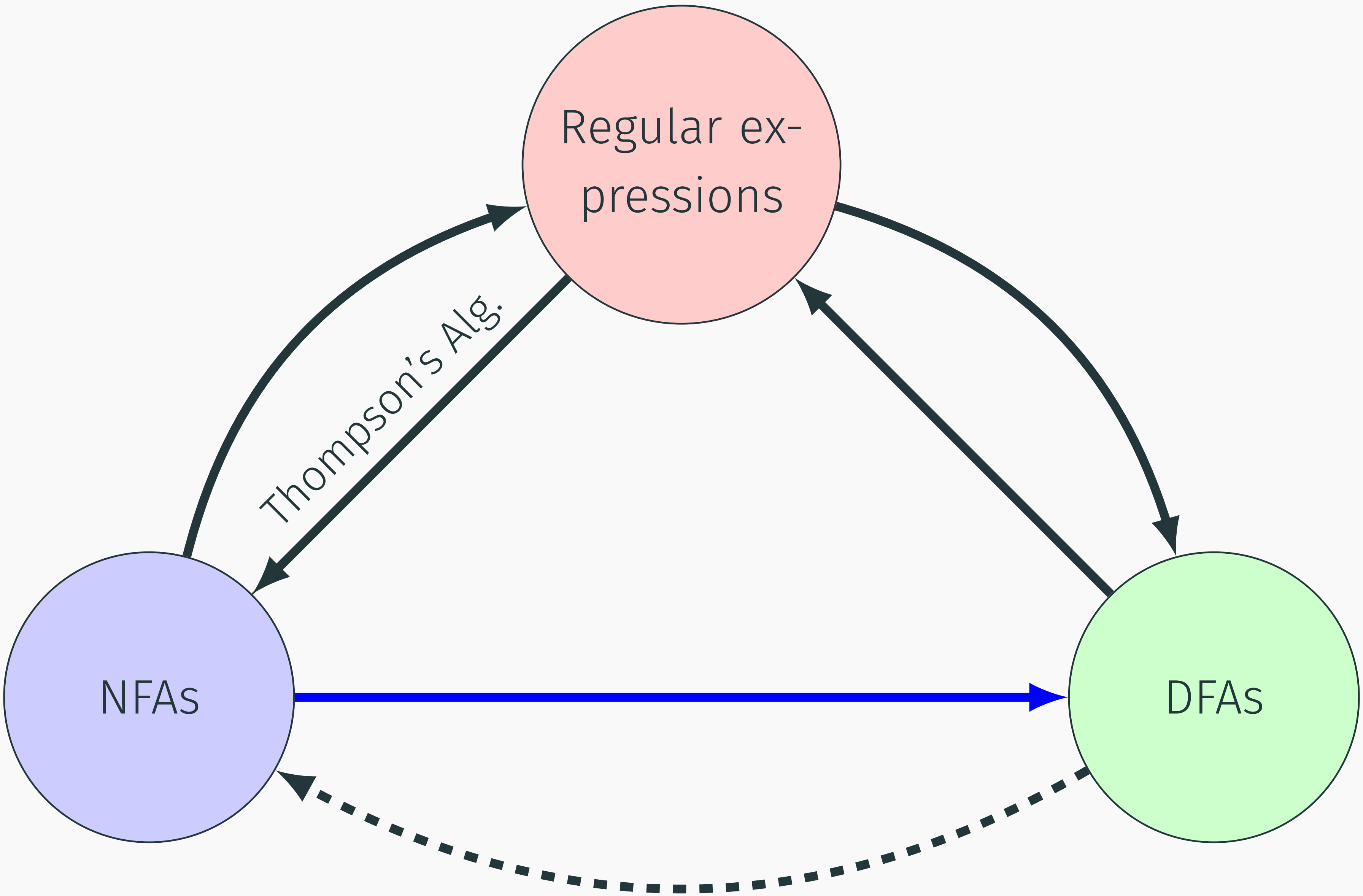


0



Conversion of NFA to DFA

Proving equivalence



Equivalence of NFAs and DFAs

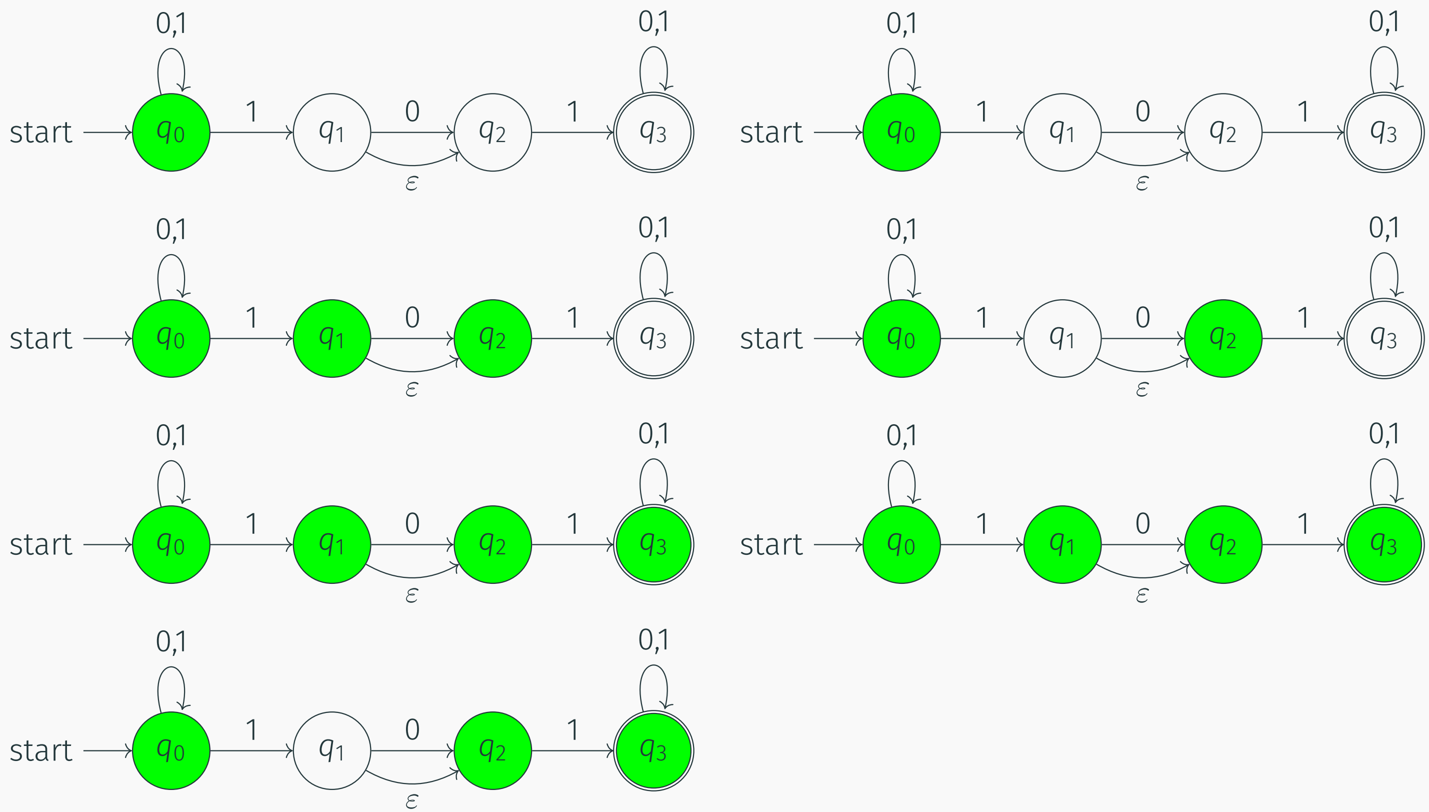
Theorem
For every NFA N there is a DFA M such that $L(M) = L(N)$.

DFAs are memoryless...

- DFA knows only its current state.
- The state is the memory.
- To design a DFA, answer the question:
  What minimal info needed to solve problem.

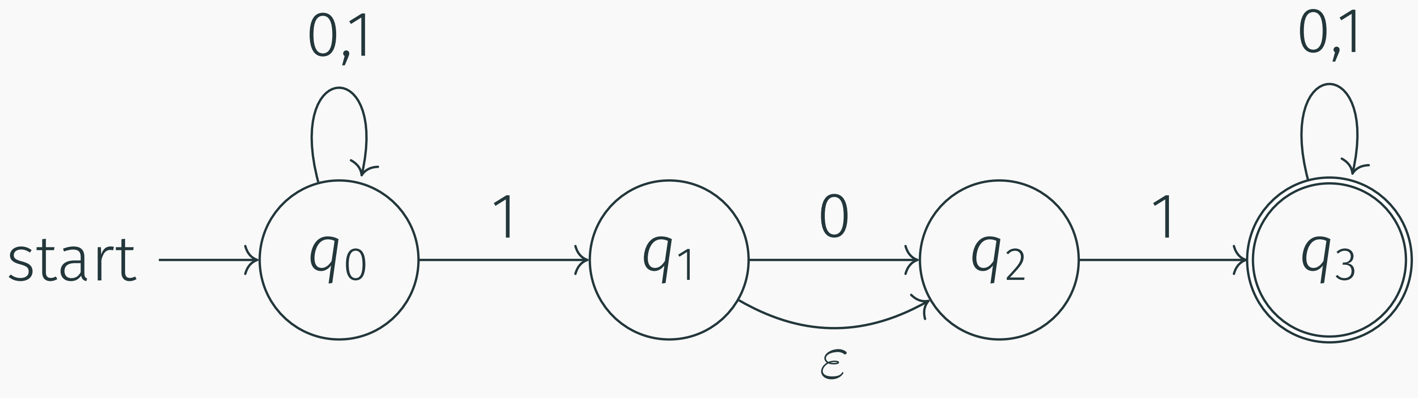
Simulating NFA

NFAs know many states at once on input 010110.



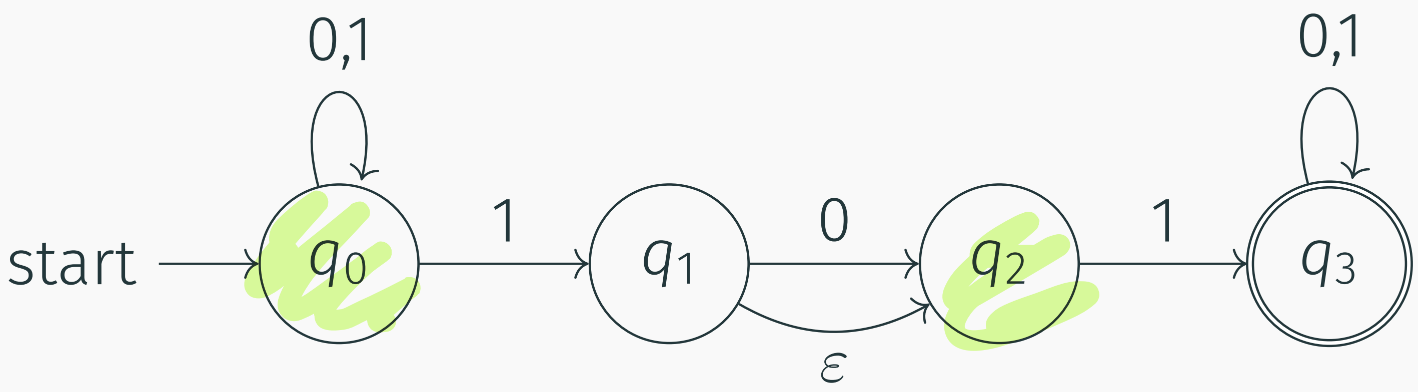
The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.



The state of the NFA

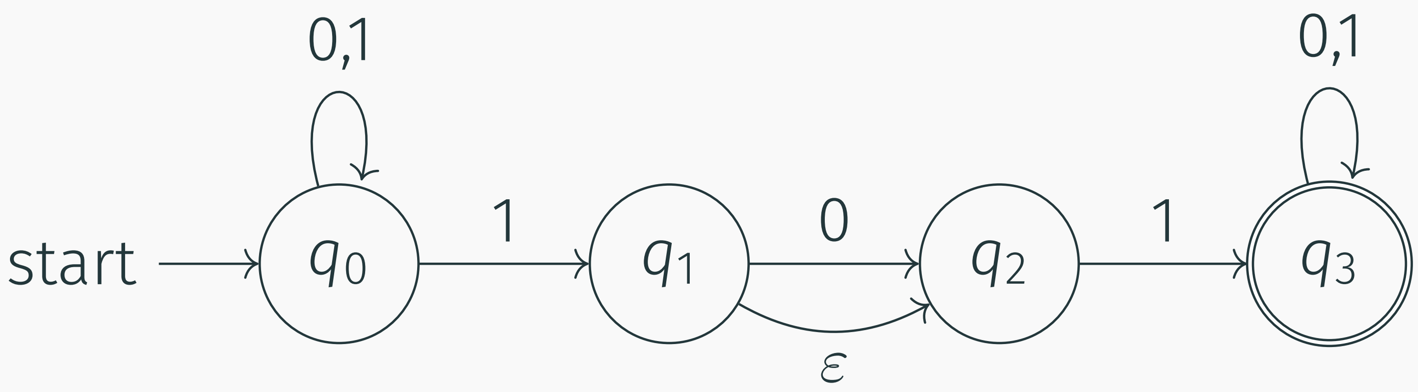
It is easy to state that the state of the automata is the states that it might be situated at.



configuration: A set of states the automata might be in.

The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

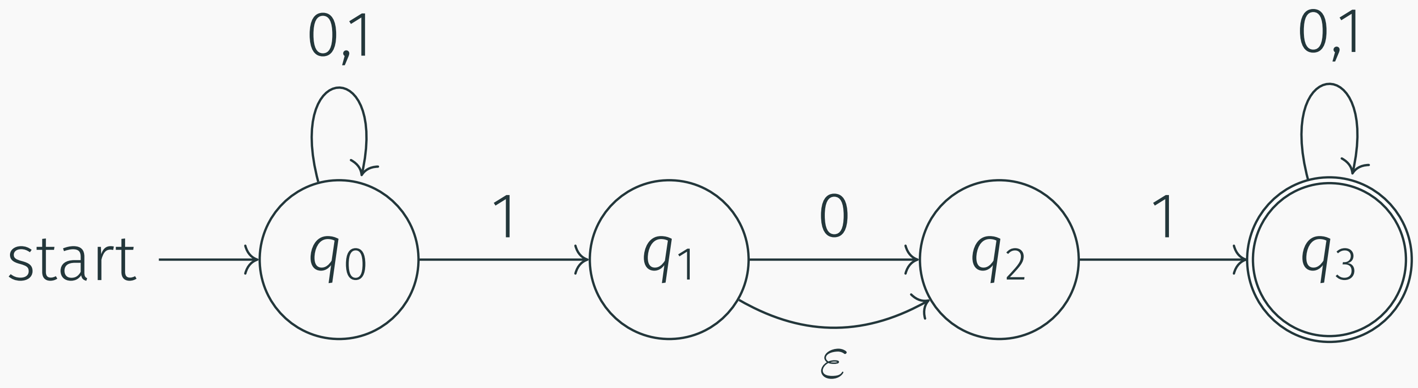


configuration: A set of states the automata might be in.

Possible configurations: $\mathcal{P}(q) = \emptyset$, $\{q_0\}$, $\{q_0, q_1\}$...

The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

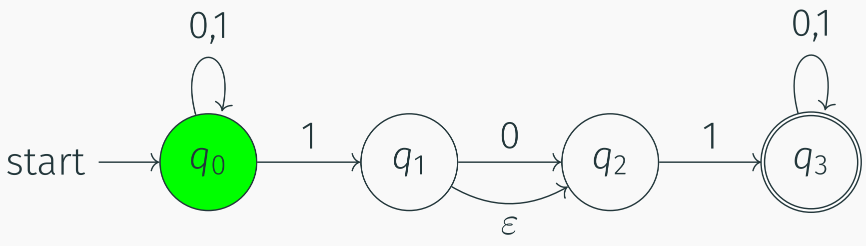


configuration: A set of states the automata might be in.

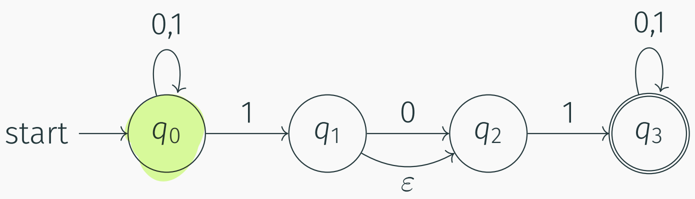
Possible configurations: $\mathcal{P}(q) = \emptyset$, $\{q_0\}$, $\{q_0, q_1\}$...

Big idea: Build a DFA on the configurations.

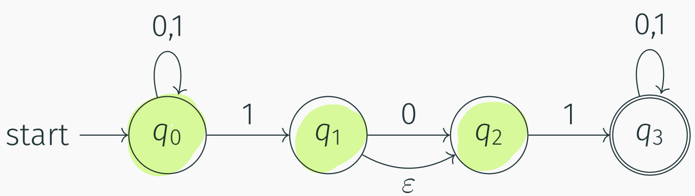
Example



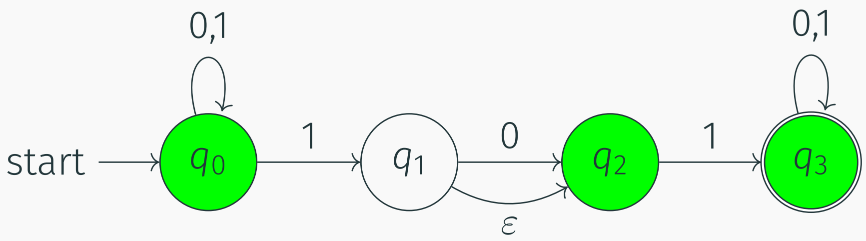
If receives 0 :



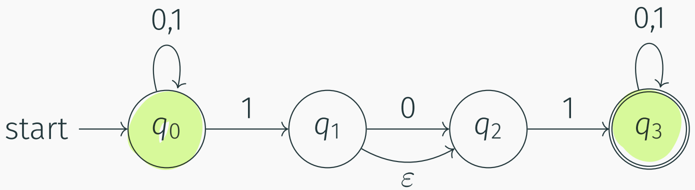
If receives 1 :



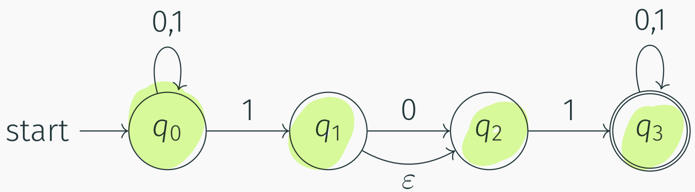
Example



If receives 0 :



If receives 1 :



Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient?

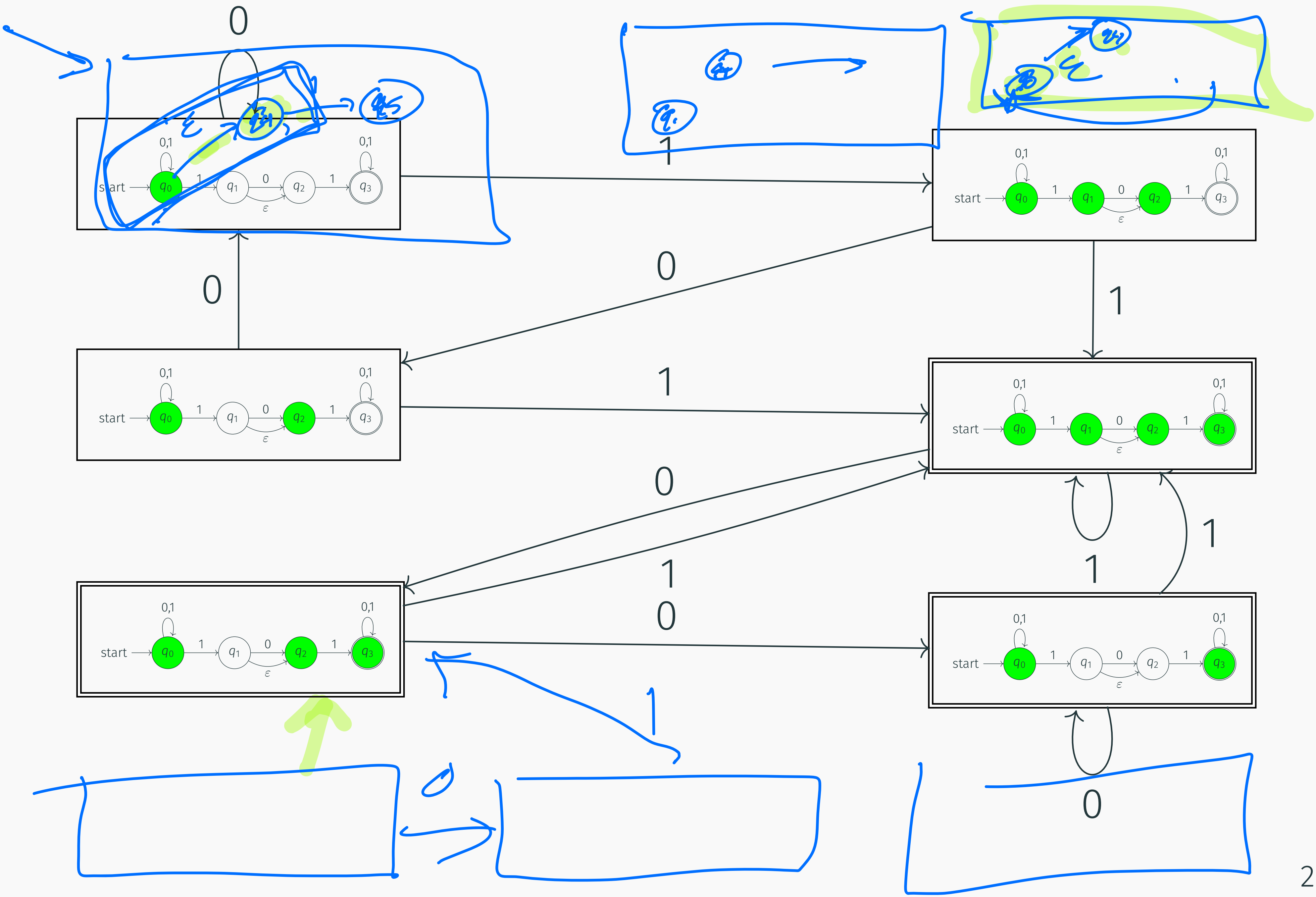
Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol a in the input.
- When should the program accept a string w? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: DFA M simulating N should know current configuration of N.

State space of the DFA is $\mathcal{P}(Q)$.

DFA from NFA



Formal Tuple Notation for NFA

Definition
A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

$\delta(q, a)$ for $a \in \Sigma \cup \{\epsilon\}$ is a subset of Q — a set of states.

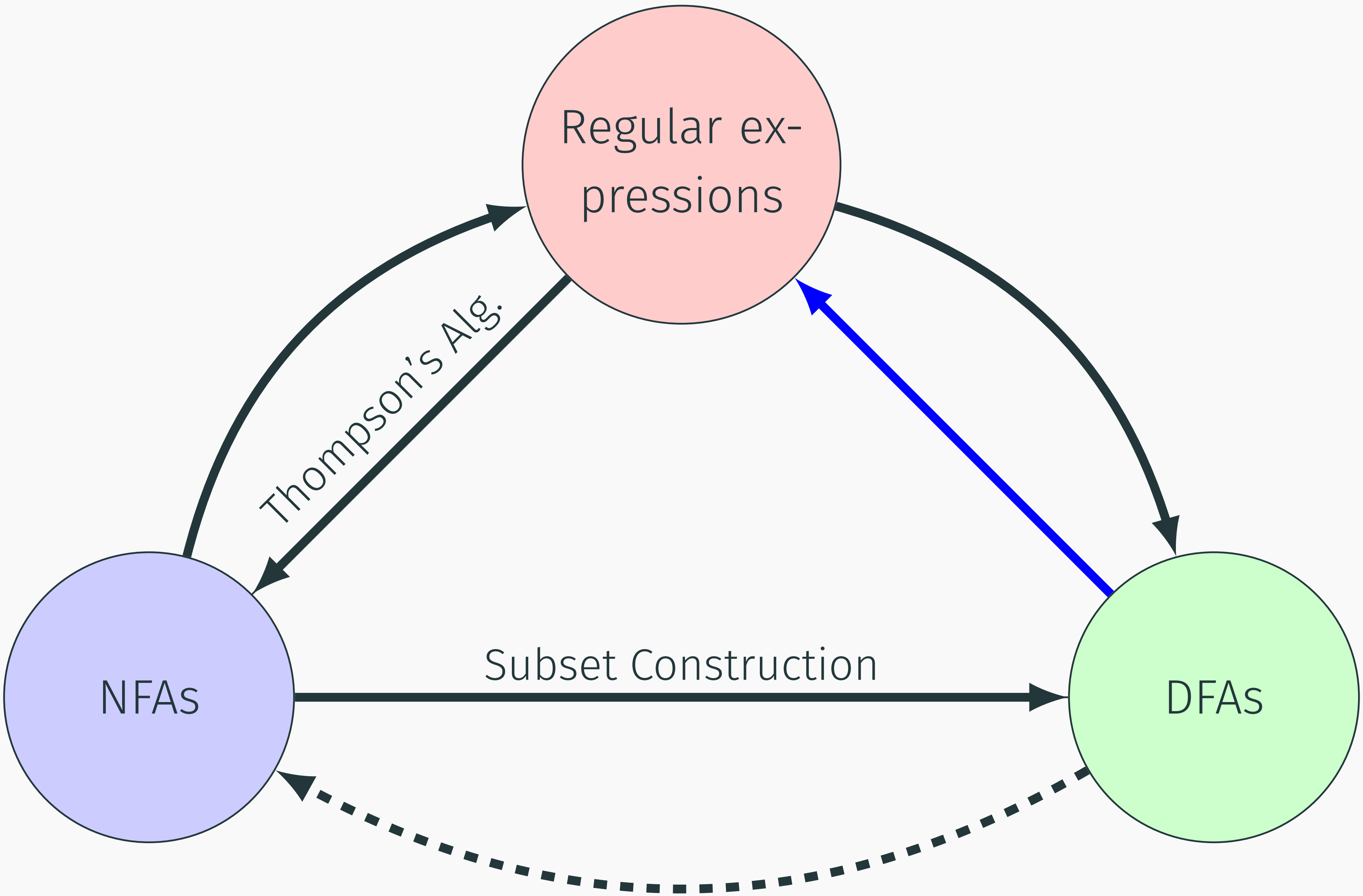
Subset State Construction

NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon reach(s)$
- $A' = \{X \subseteq Q \mid X \cap A \neq \emptyset\}$
- $\delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)$ for every $X \subseteq Q$, $a \in \Sigma$

DFAs to Regular expressions

Proving equivalence

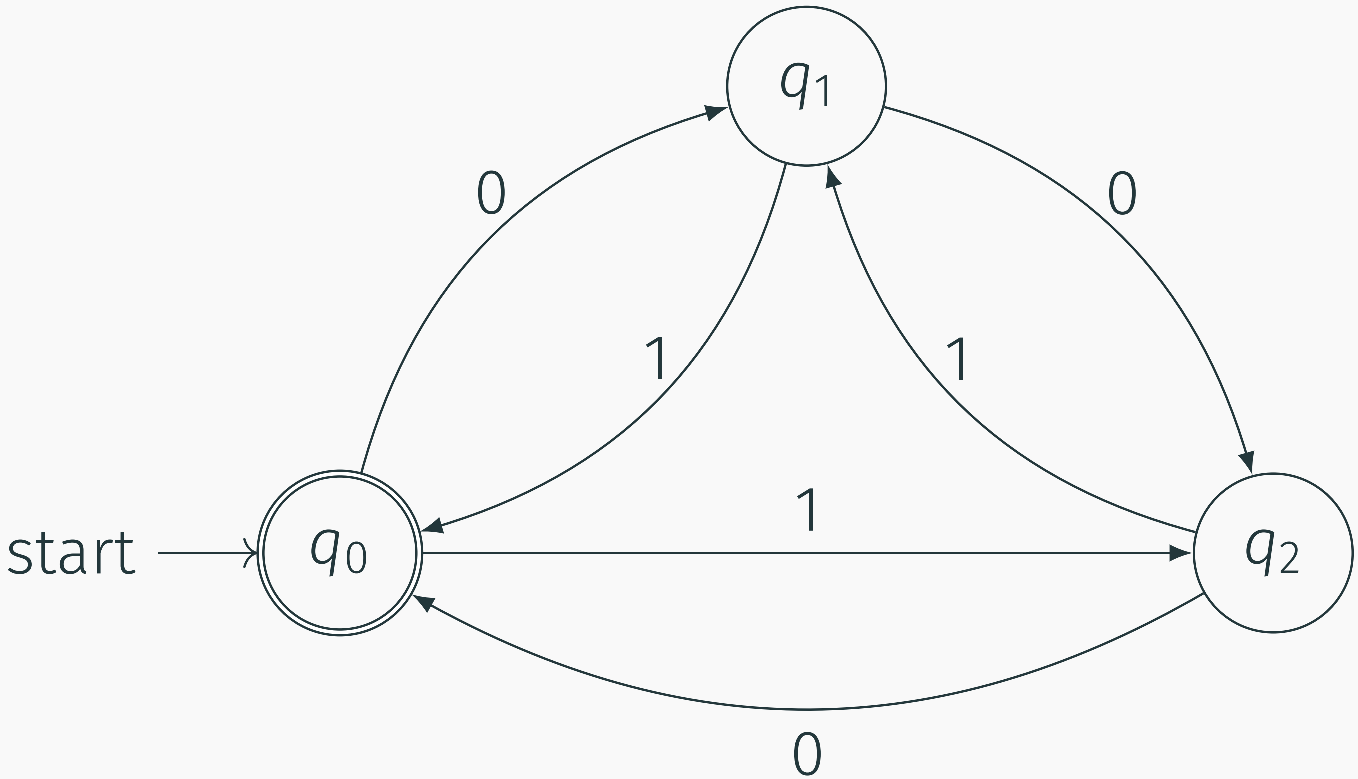


State Removal method

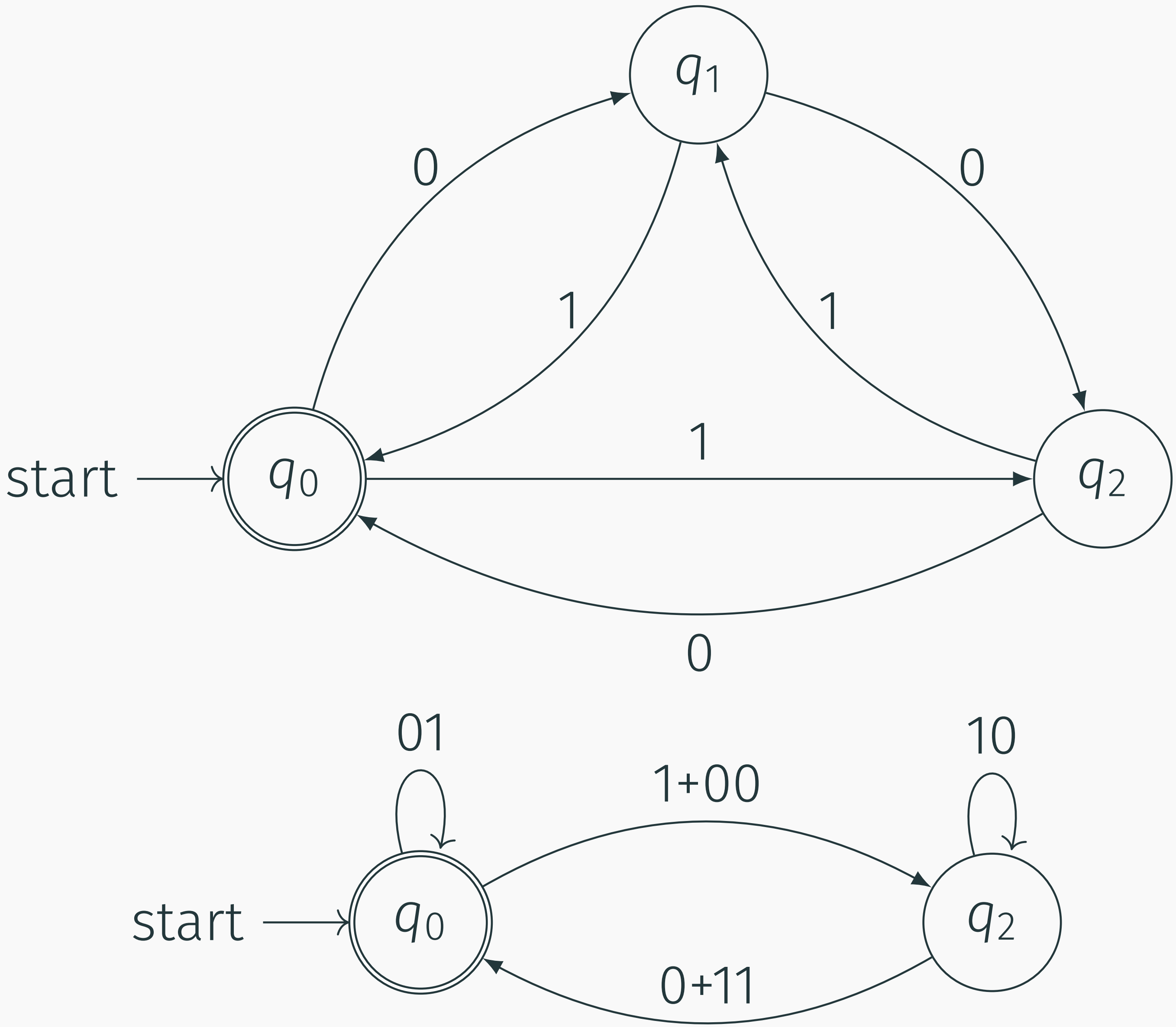
If $q_1 = \delta(q_0, x)$ and $q_2 = \delta(q_1, y)$

then $q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta^{*}(q_0, xy)$

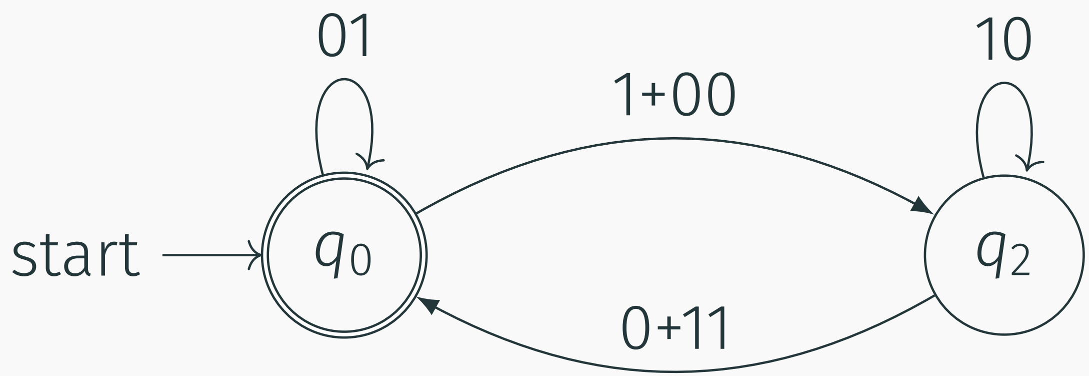
State Removal method - Example



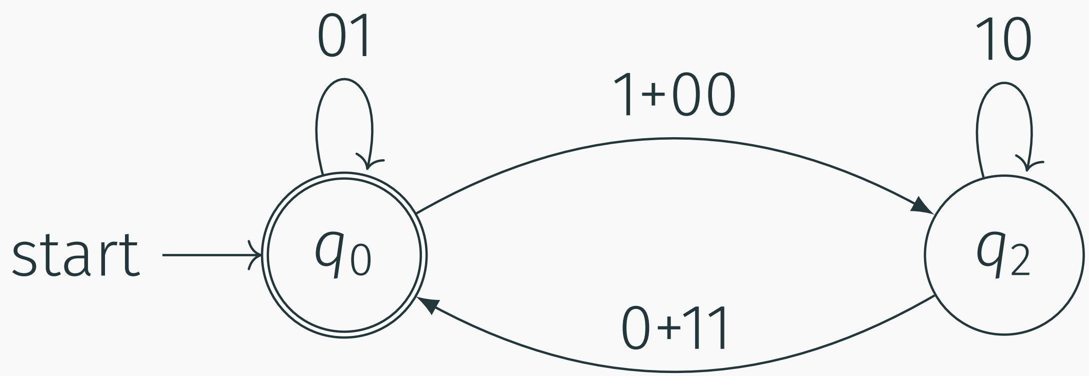
State Removal method - Example



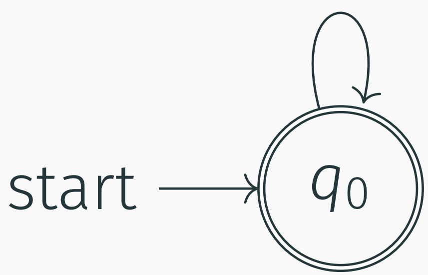
State Removal method - Example



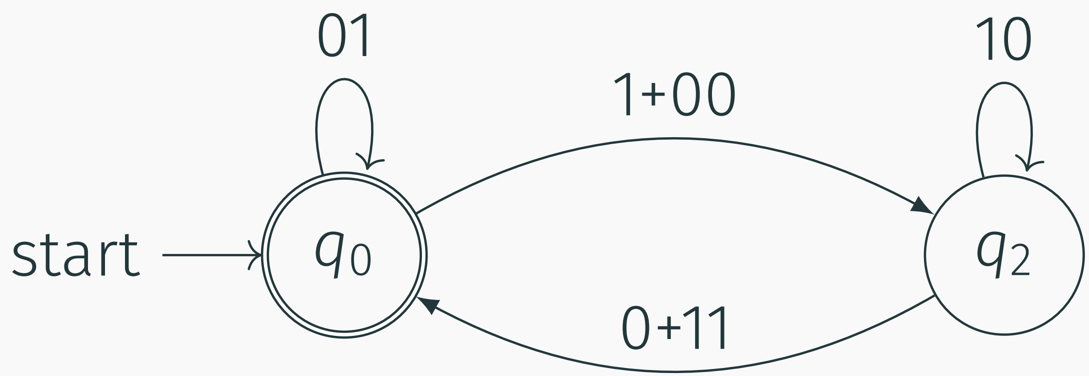
State Removal method - Example



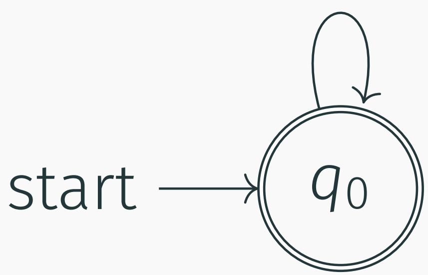
$$01 + (1 + 00)(10)^*(0 + 11)$$

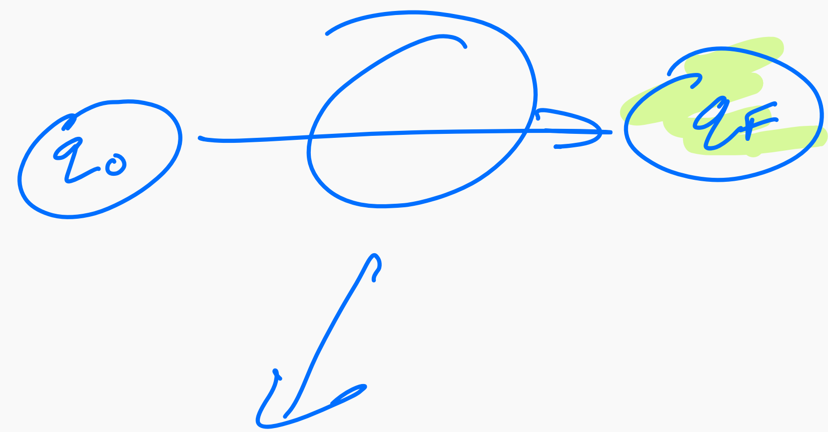


State Removal method - Example



$$01 + (1 + 00)(10)^*(0 + 11)$$





$$(01^* + (1 + 00)(10)^*(0 + 11))^*$$

Algebraic method

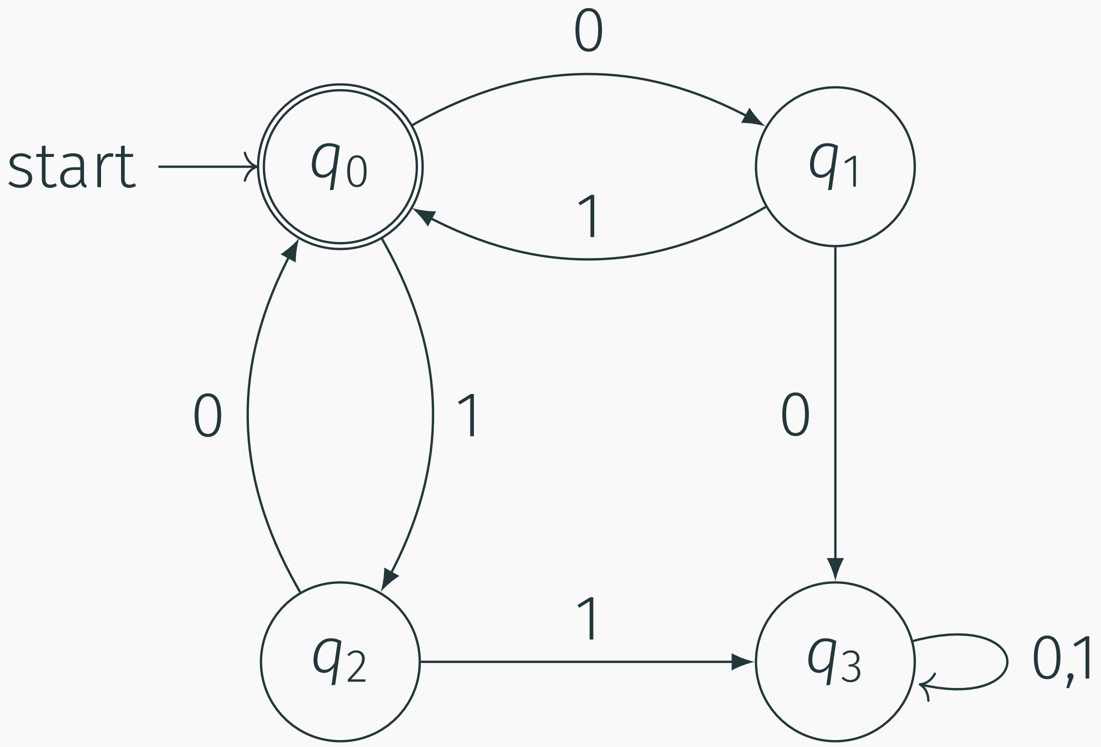
Transition functions are themselves algebraic expressions!

Demarcate states as variables.

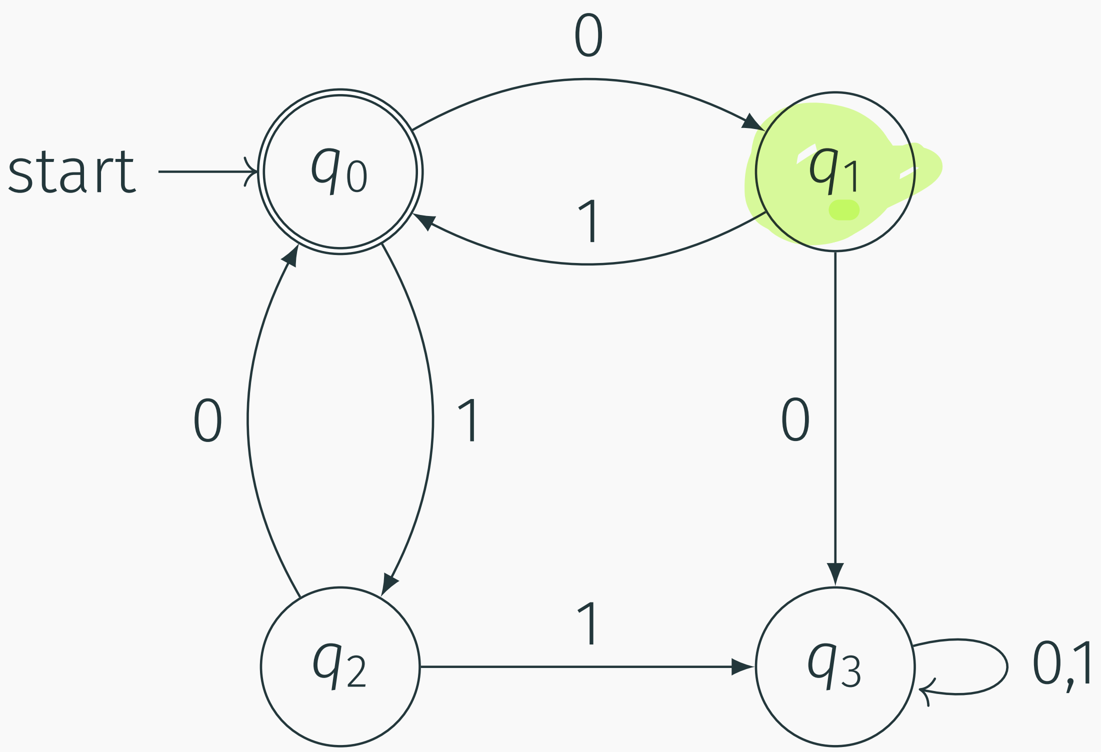
Can rewrite $q_1 = \delta(q_0, x)$ as $q_1 = q_0\mathrm{x}$

Solve for accepting state.

Algebraic method - Example



Algebraic method - Example



- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Now we simple solve the system of equations for q_0:

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$

Theorem (Arden's Theorem)
$R = Q + RP = QP^*$

Algebraic method - Example

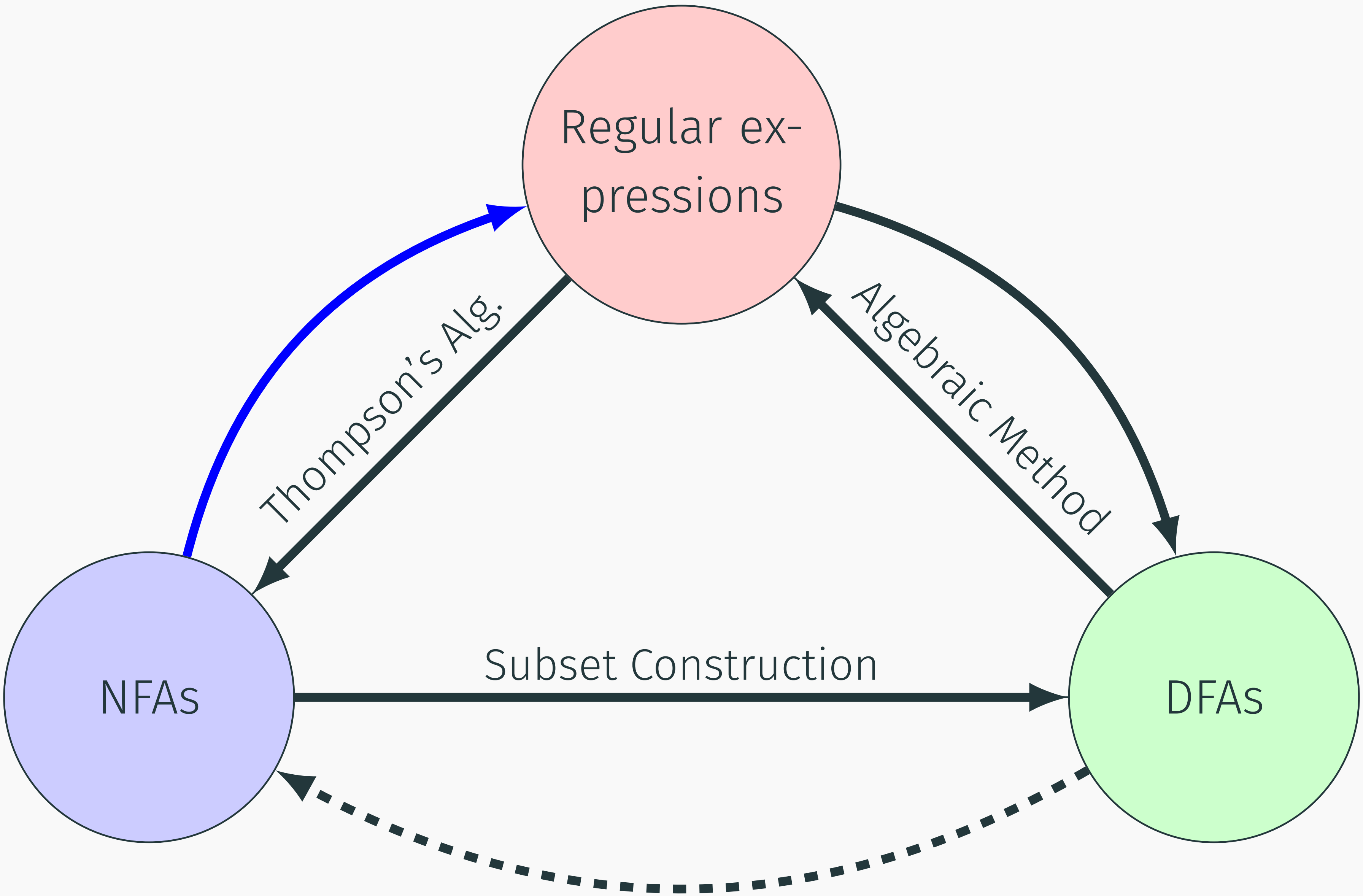
- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3(0 + 1)$

Now we simple solve the system of equations for q_0:

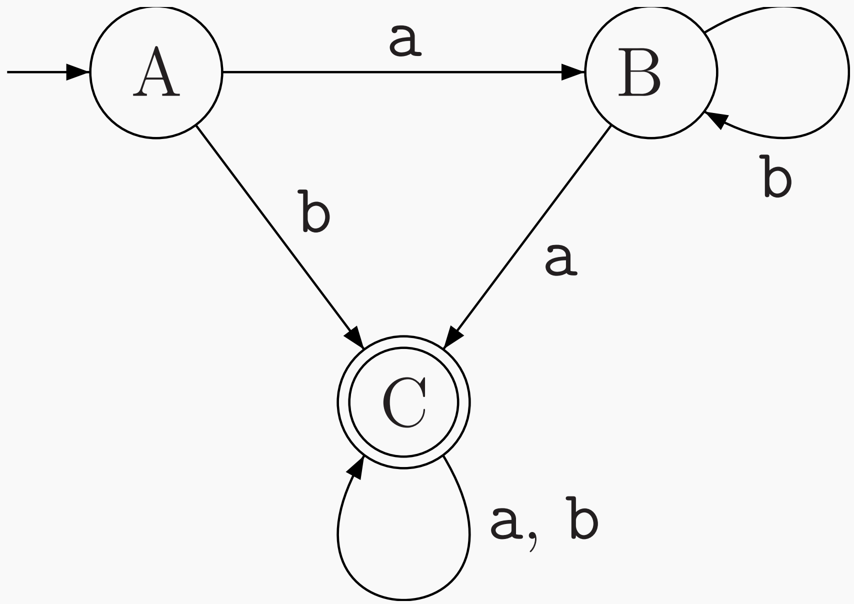
- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0(01 + 10)$
- $q_0 = \epsilon(01 + 10)^* = (01 + 10)^*$

Converting NFAs to Regular Expression

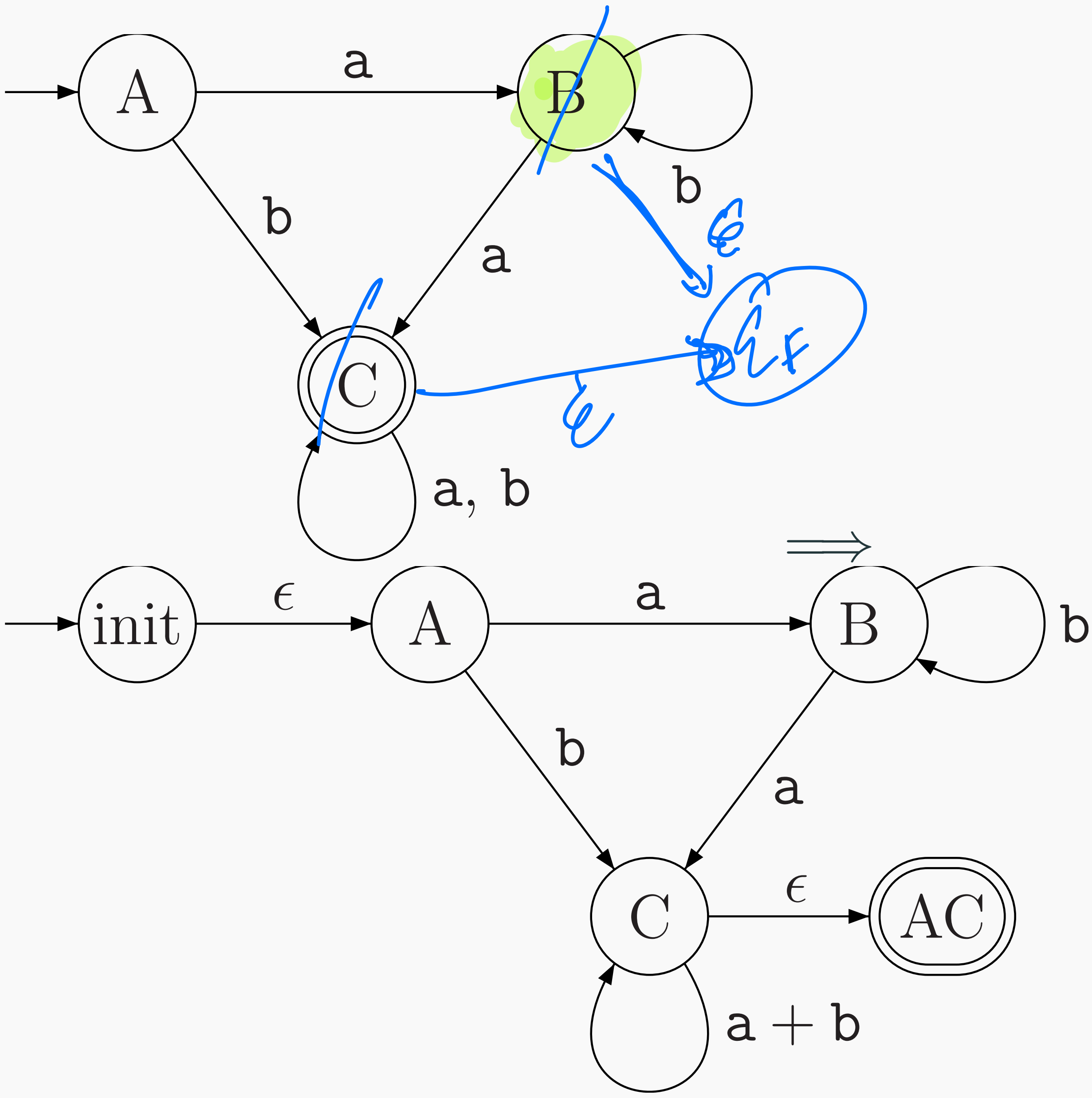
Proving equivalence



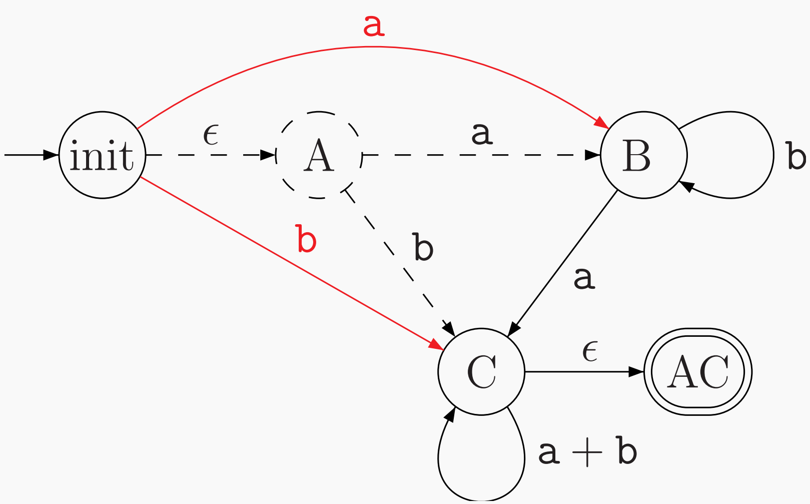
Stage 0: Input



Stage 1: Normalizing

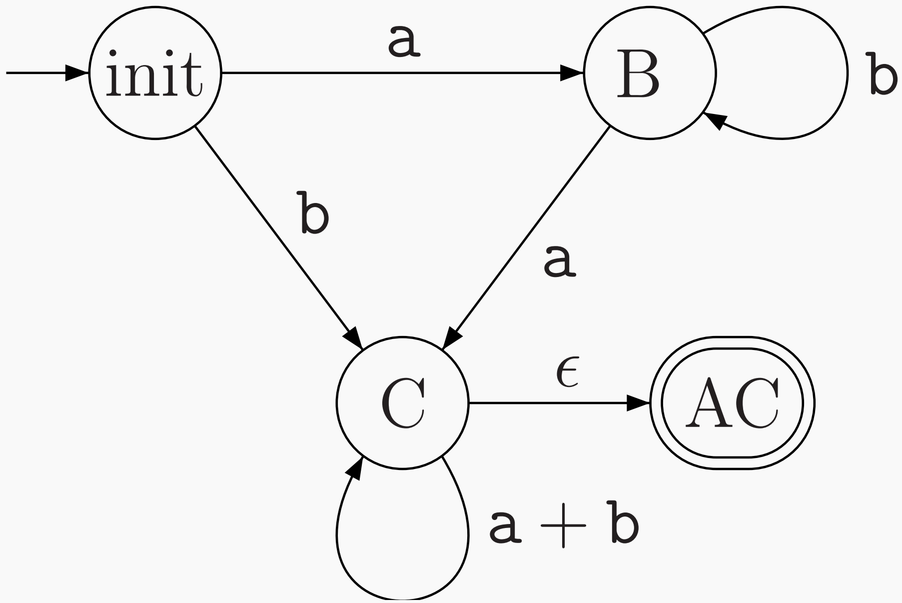


Stage 2: Remove state A



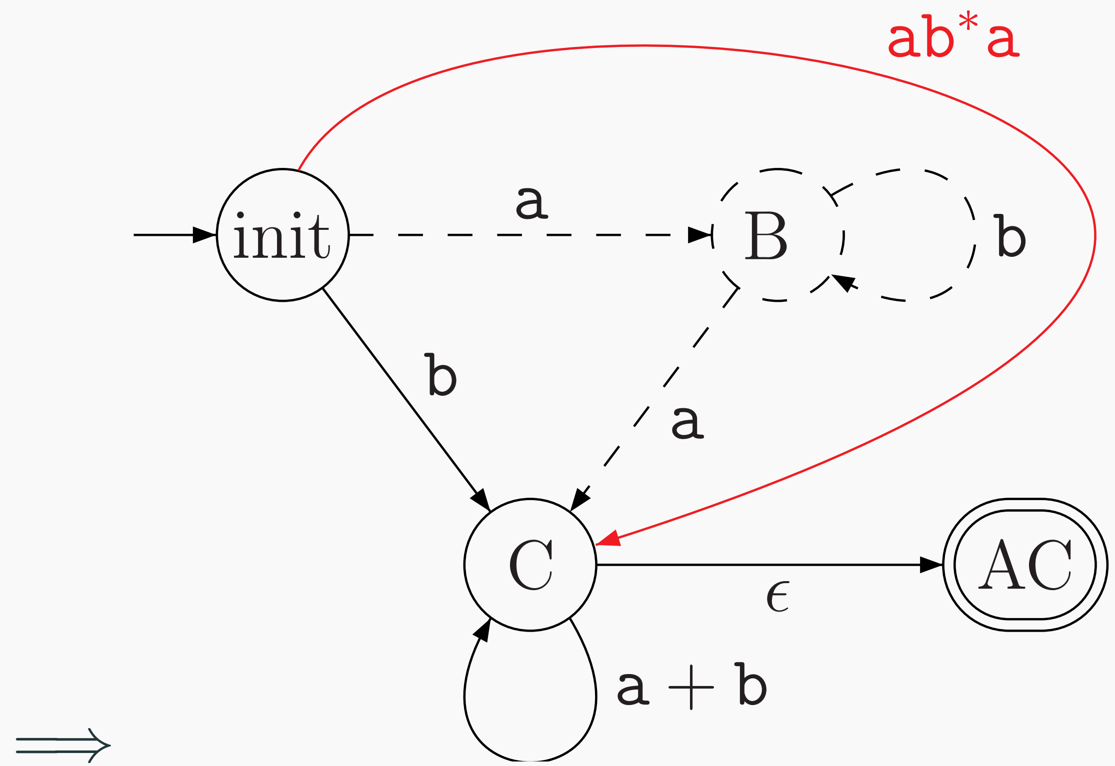
$\Longrightarrow$

Stage 4: Redrawn without old edges



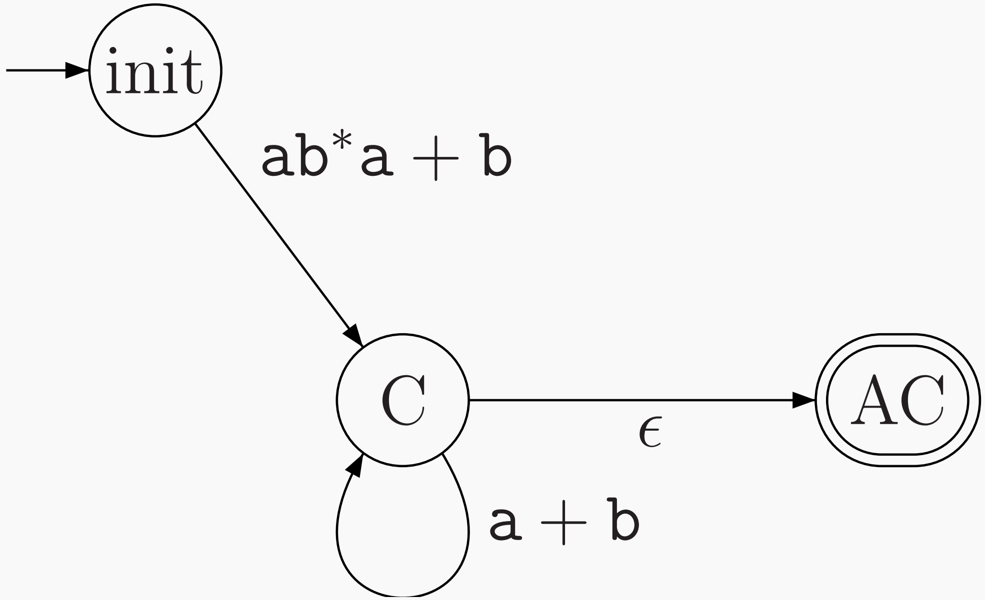
$\Longrightarrow$

Stage 4: Removing B

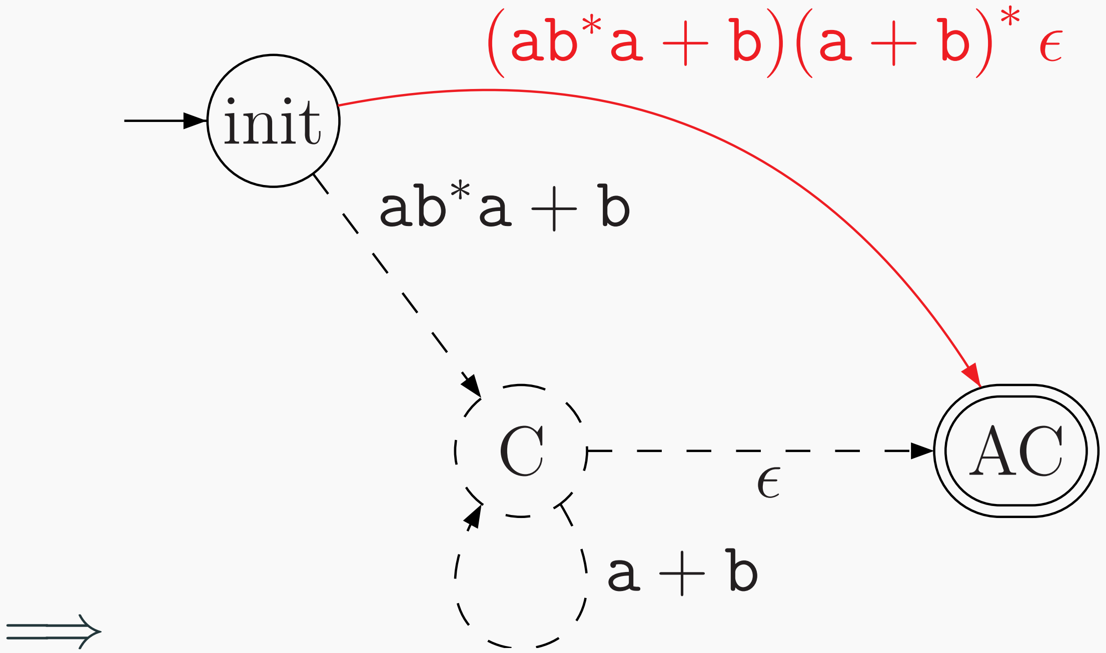


Stage 5: Redraw

$\Longrightarrow$

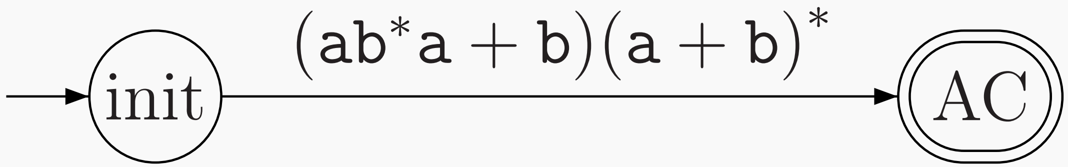


Stage 6: Removing C

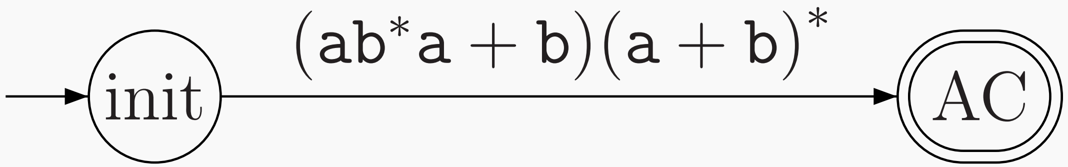


Stage 7: Redraw

$\Longrightarrow$



Stage 8: Extract regular expression

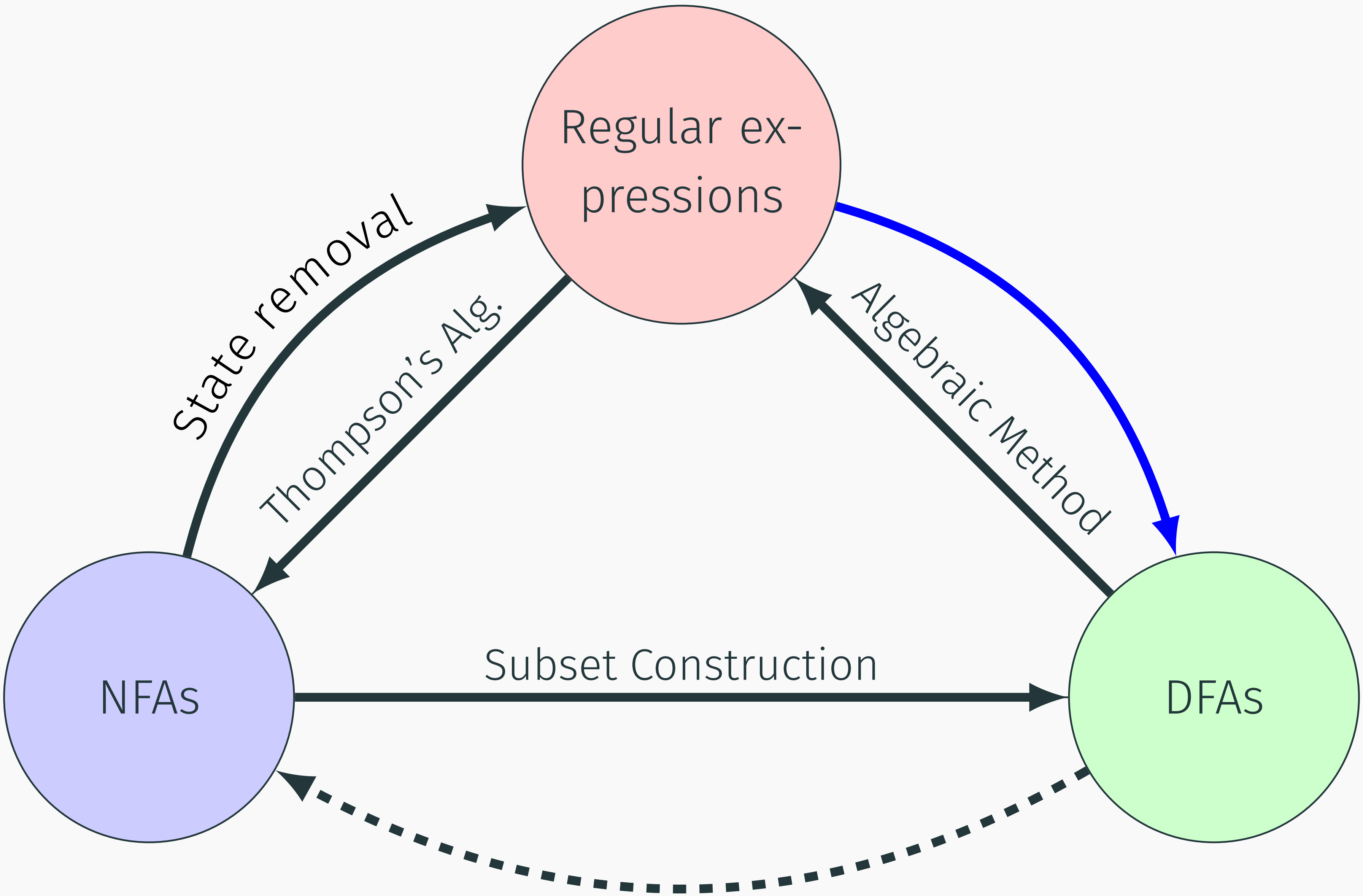


Thus, this automata is equivalent to the regular expression

$$\boxed{(ab^*a + b)(a + b)^*.}$$

Regular expressions to DFAs

Proving equivalence



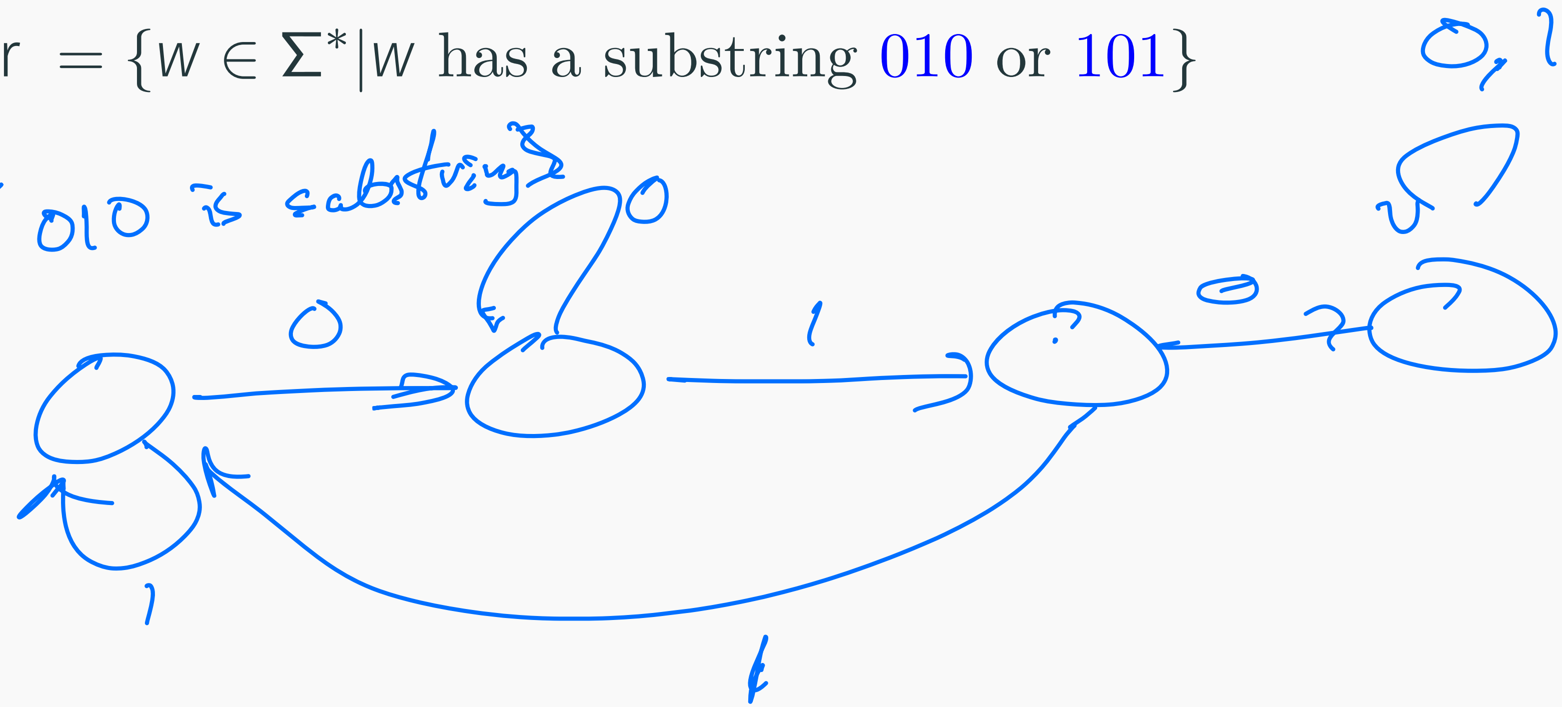
Difficulty going from RegEx's to DFAs

Lemma
Many regular expressions cannot be easily converted to DFAs.

Lemma
Many regular expressions cannot be easily converted to DFAs.

Consider $= \{w \in \Sigma^* \mid w \text{ has a substring } 010 \text{ or } 101\}$

L = { 010 is substring }



Difficulty going from RegEx's to DFAs

Lemma
Many regular expressions cannot be easily converted to DFAs.

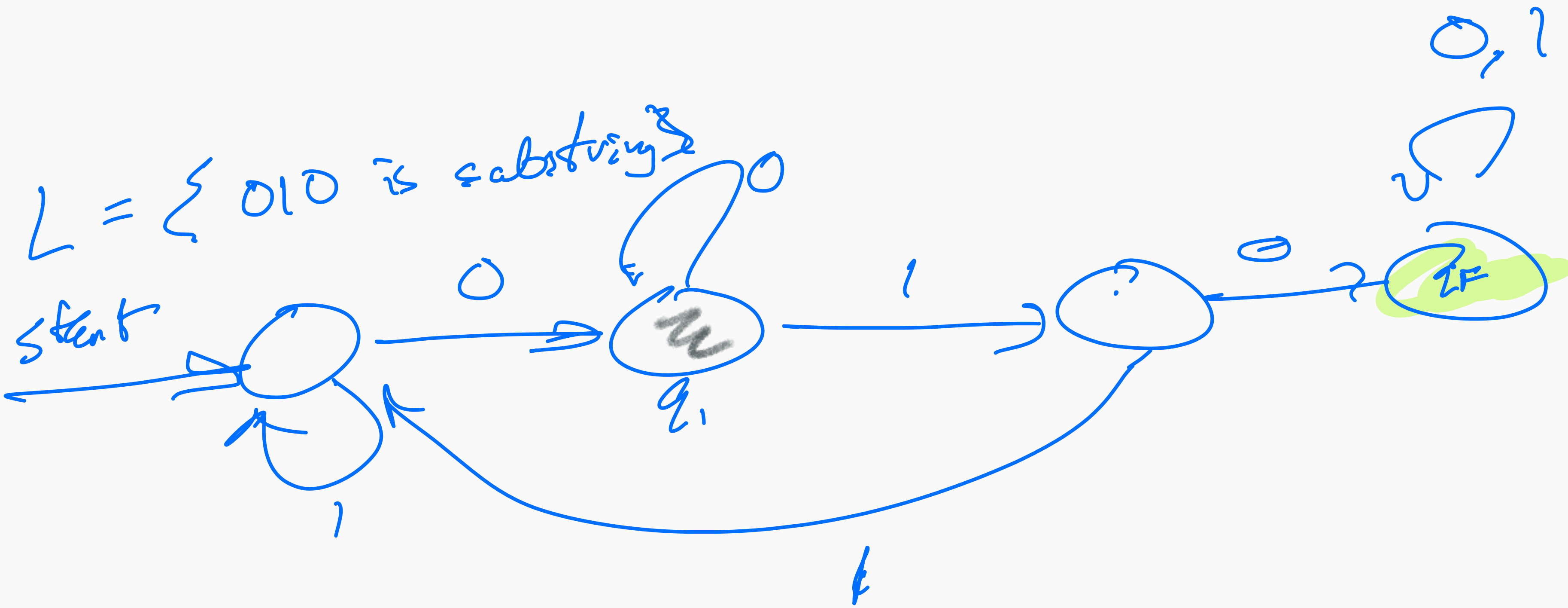
Consider $= \{w \in \Sigma^* | w \text{ has a substring } 010 \text{ or } 101\}$

- Is possible using Brzozowski[1] algorithm. Not needed for this course.

But here's the idea anyway....

Draw the DFA for $= \{w \in \Sigma^* | w \text{ has a substring } 010\}$. What does each state represent?

$L = \{ 010 \text{ is substring} \}$



$(1+0)^*(010)(1+0)^*$

$(10)(0+1)^*$

$q_F = 10 q_1$

Brzozowski Method

Brings us to the **Brzozowski derivative** where $(u^{-1}S)$ of a set S of strings and a string u is the set of strings obtainable from a string in S by cutting of the prefixing u.

Consider the language $R = (ab + c)^*$

q_F

Brzozowski Method

Brings us to the **Brzozowski derivative** where $(u^{-1}S)$ of a set S of strings and a string u is the set of strings obtainable from a string in S by cutting of the prefixing u.

Consider the language $R = (ab + c)^*$

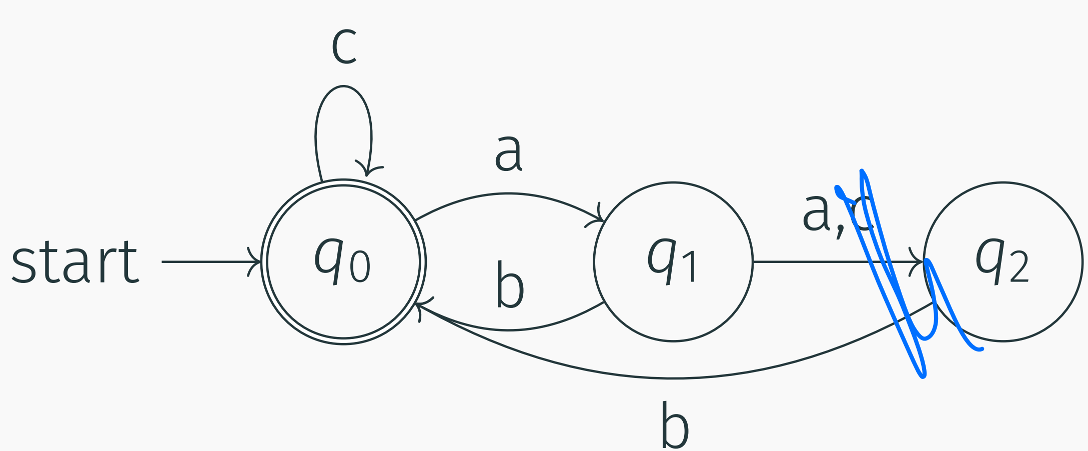
R	$a^{-1}R$	$b^{-1}R$	$c^{-1}R$
$q_0 = \varepsilon^{-1}R = (ab + c)^*$	$b\,(ab + c)^*$	$\emptyset$	$\boxed{(ab + c)^*}$
$q_1 = b\,(ab + c)^*$	$\emptyset$	$(ab + c)^*$	$\emptyset$
$q_2 = \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Brzozowski Method

Brings us to the **Brzozowski derivative** where $(u^{-1}S)$ of a set S of strings and a string u is the set of strings obtainable from a string in S by cutting of the prefixing u.

Consider the language $R = (ab + c)^*$

R	$a^{-1}R$	$b^{-1}R$	$c^{-1}R$
$q_0 = \varepsilon^{-1}R = (ab + c)^*$	$b\,(ab + c)^*$	$\emptyset$	$(ab + c)^*$
$q_1 = b\,(ab + c)^*$	$\emptyset$	$(ab + c)^*$	$\emptyset$
$q_2 = \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Difficulty going from RegEx's to DFAs

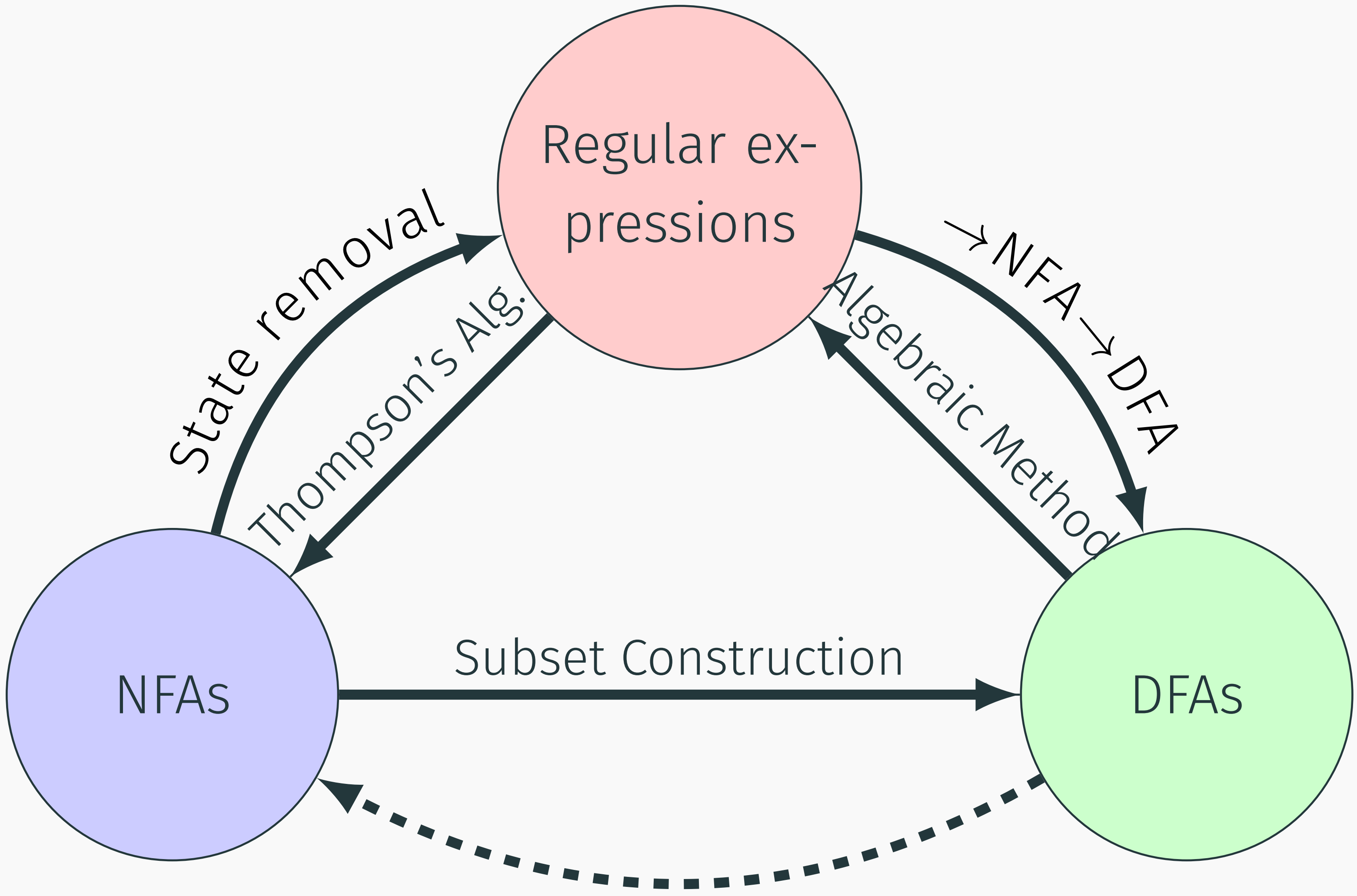
Lemma
Many regular expressions cannot be easily converted to DFAs.

Consider $= \{w \in \Sigma^* | w \text{ has a substring } 010 \text{ or } 010\}$

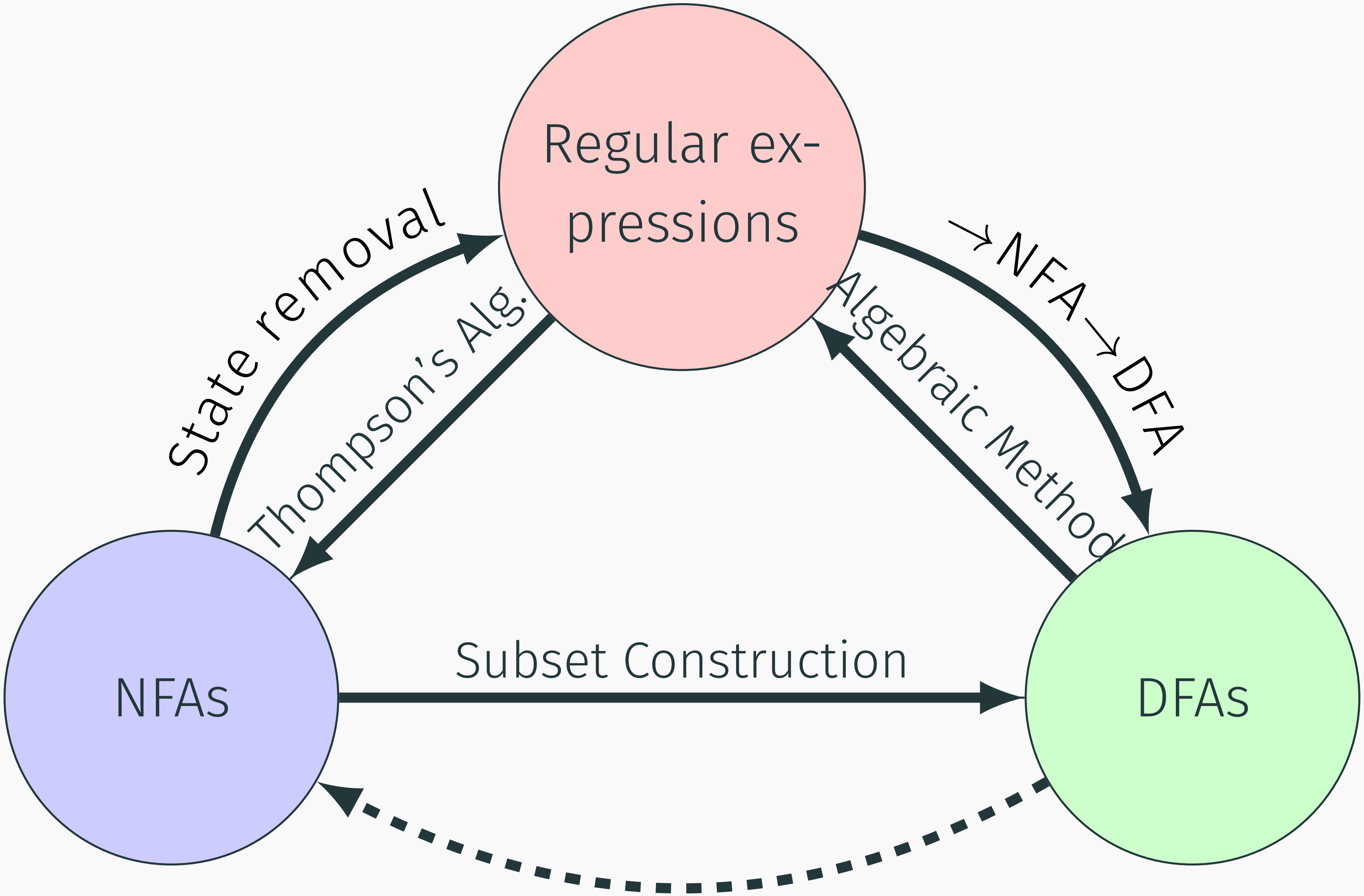
- Is possible using Brzozowski[2] algorithm. Not needed for this course.
- **Easier to just convert RegEx $\rightarrow$ NFA $\rightarrow$ DFA.**

Conclusion

Proving equivalence



Proving equivalence



But what about the expressions at aren't regular?! See on Thursday