

Pre-lecture brain teaser

Prove that 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

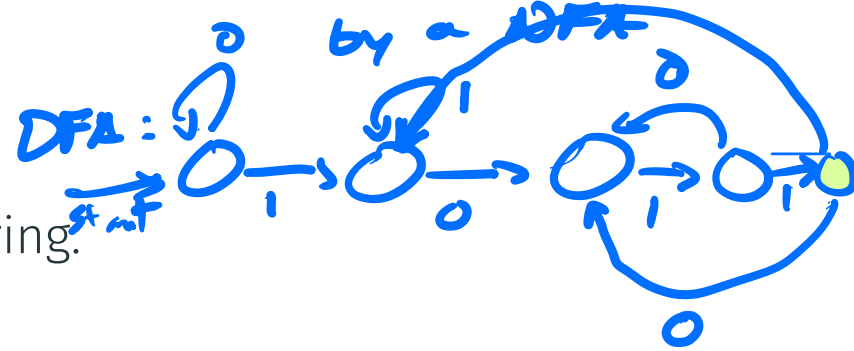
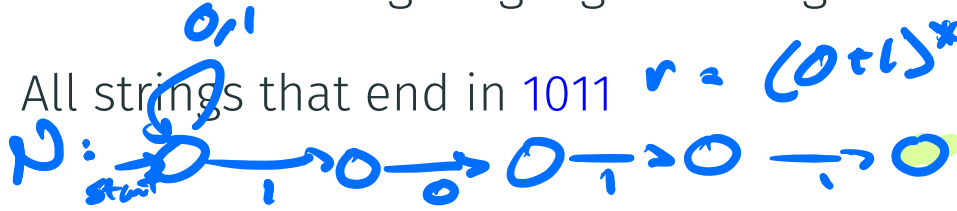
University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Prove that the following languages are regular:

representable by a regexp
* 1011 in a DFA
0 by a ~~DFA~~

- 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.

Pre-lecture brain teaser

Prove that the following languages are regular:

representable by a regexp
in a DFA
by a NFA

- All strings that end in 1011

~~✗~~ All strings that contain 101 or 010 as a substring.

NFA :

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

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

Pre-lecture brain teaser

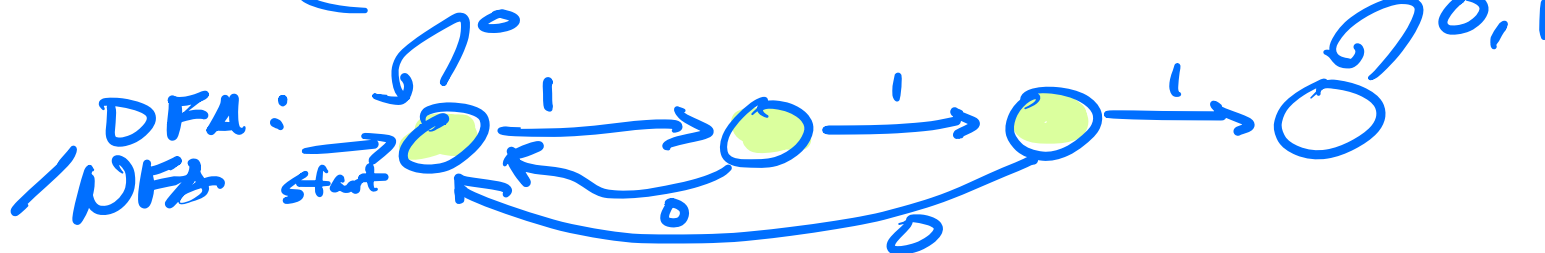
Prove that the following languages are regular:

representable by a regexp
in a DFA
by a NFA

- 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.

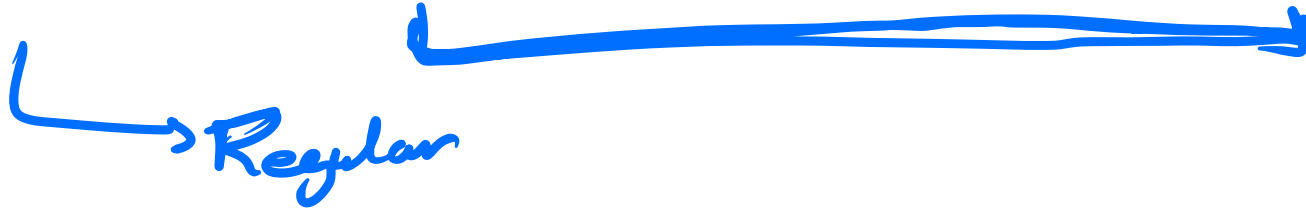
$$r = (0 + 10 + 110)^* (\epsilon + 1 + 11)$$



Regular Languages, DFAs, NFAs

Theorem *representable*

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



Regular Languages, DFAs, NFAs

Theorem

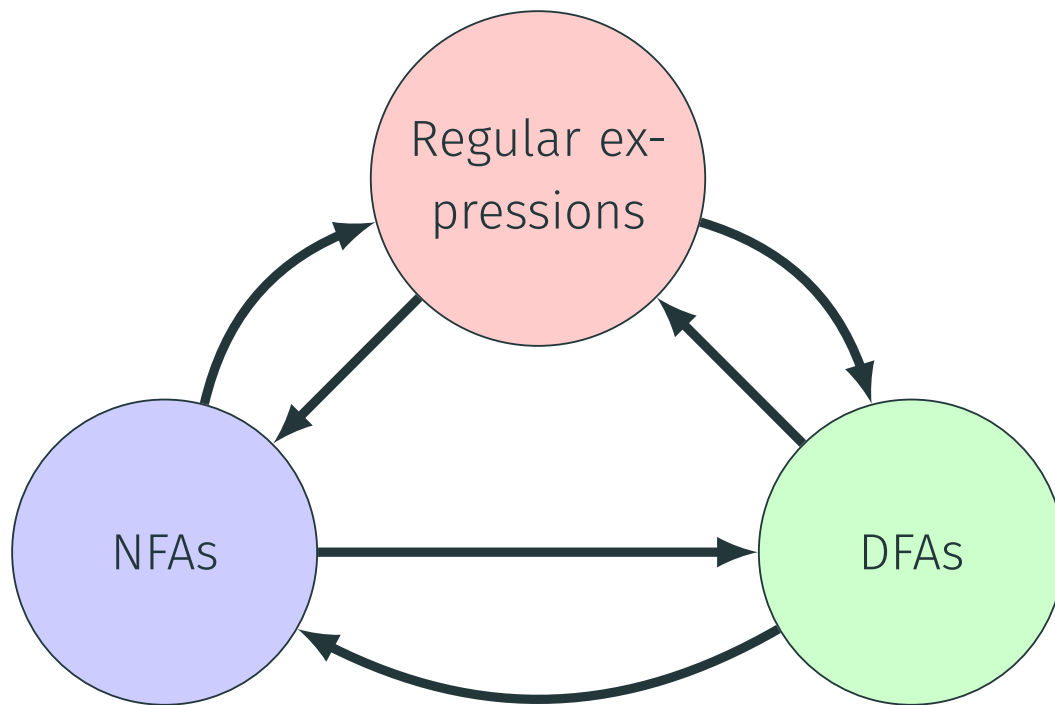
Languages accepted by DFAs, NFAs, and regular expressions are the same.

- 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

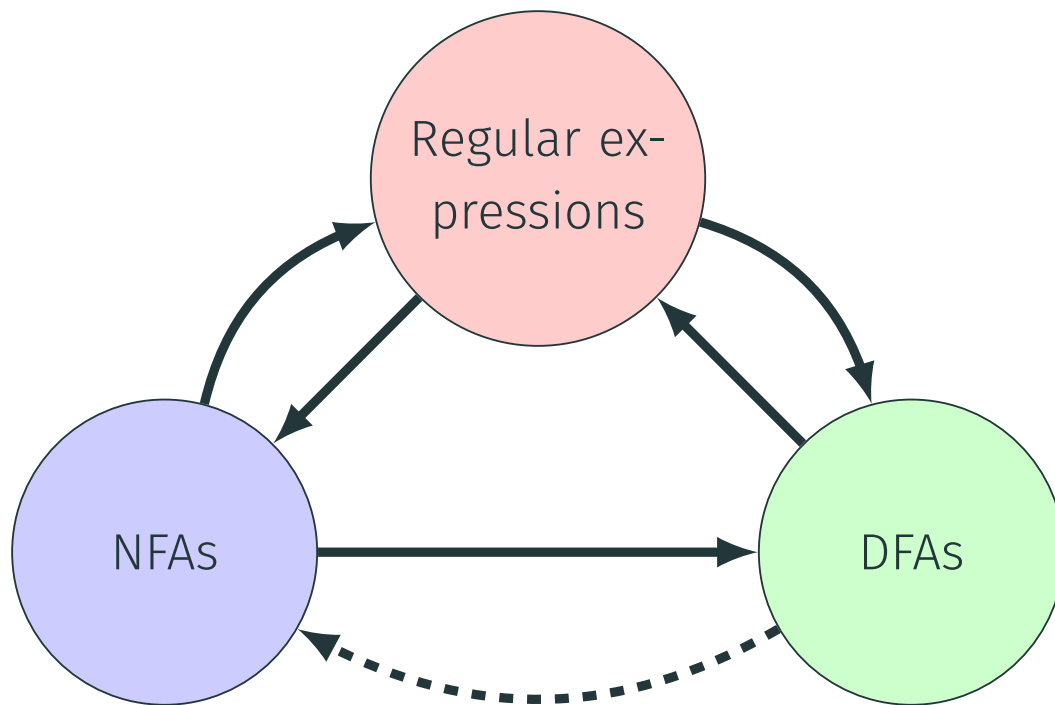
*Languages accepted by **DFAs**, **NFAs**, and regular expressions are the same.*



Regular Languages, DFAs, NFAs

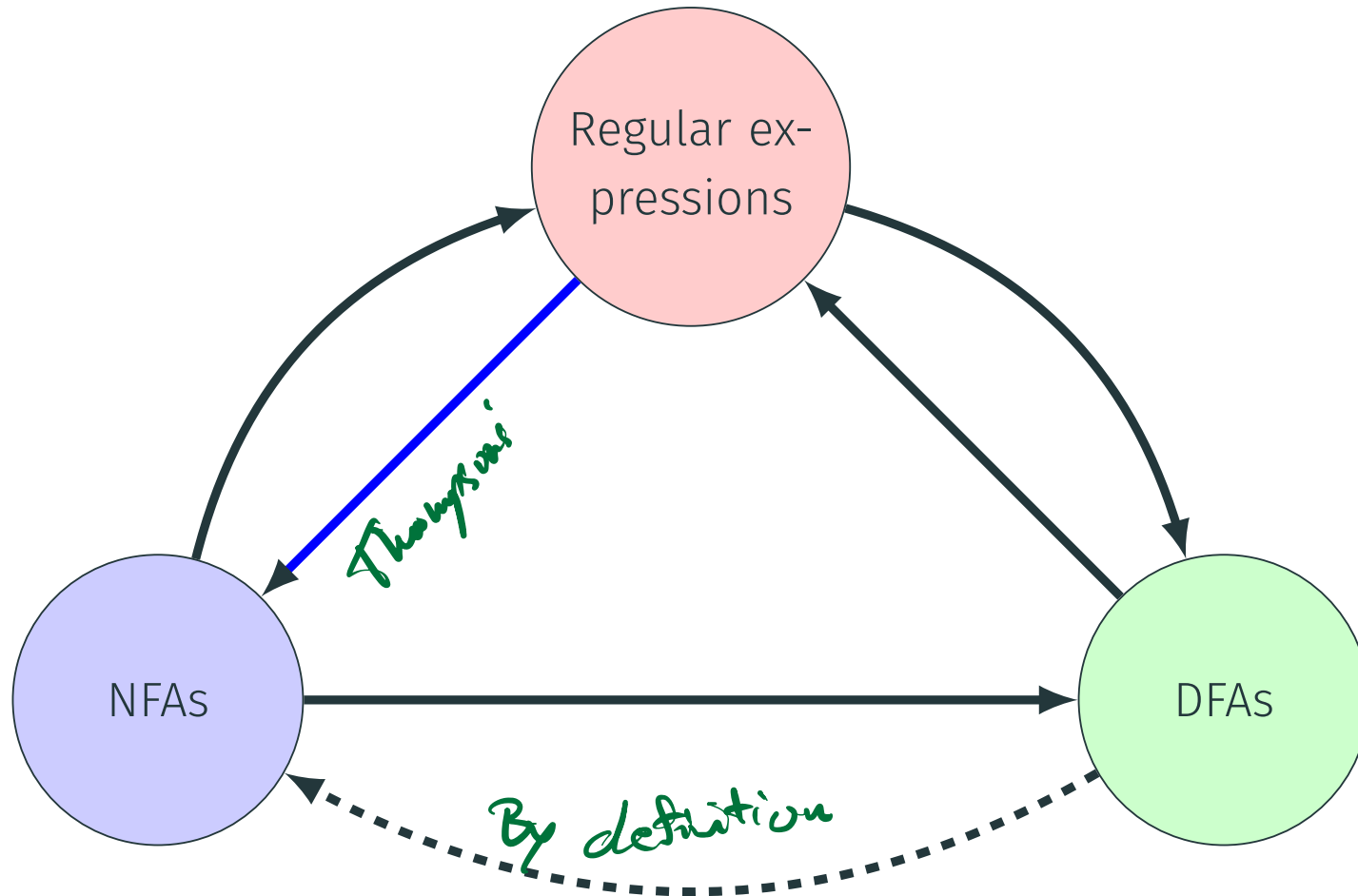
Theorem

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



Regular Expression to NFA

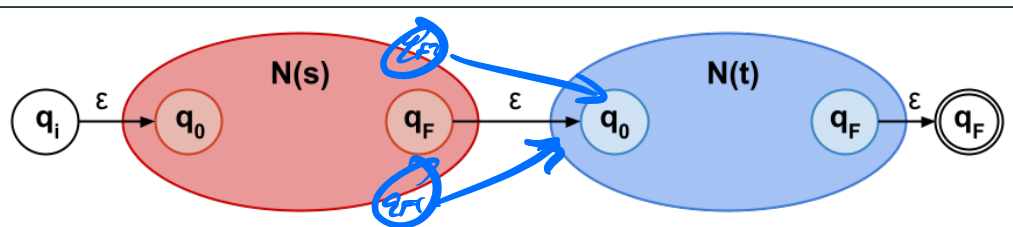
Proving equivalence



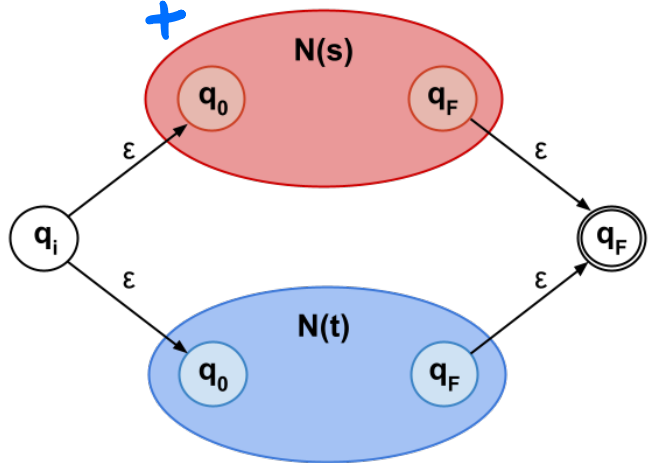
Thompson's algorithm

Given two NFAs s and t :

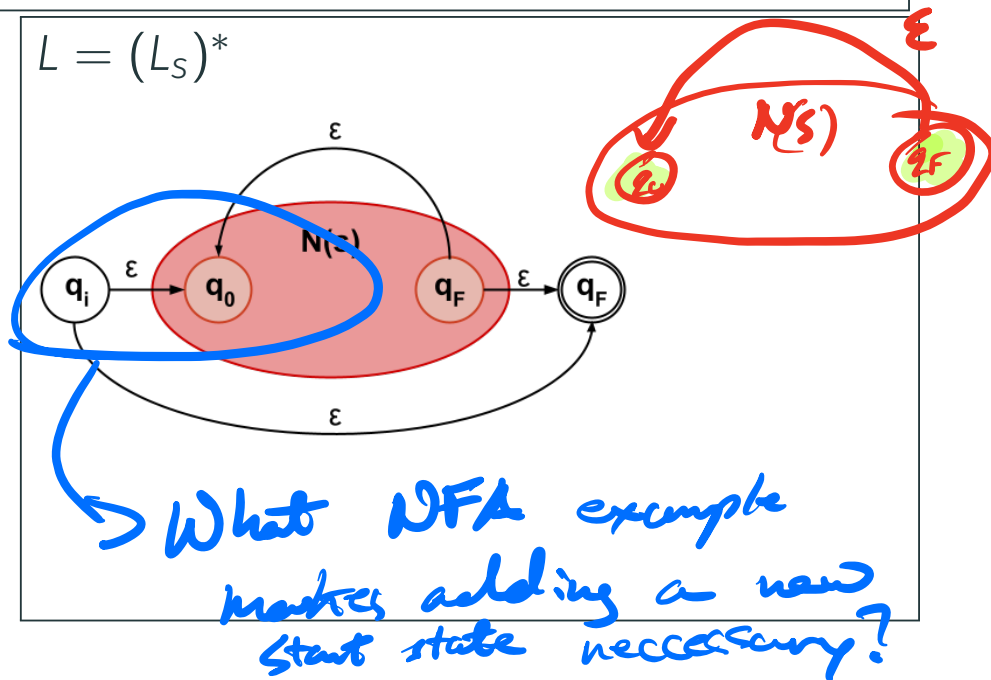
$$L = L_s \cdot L_t$$



$$L = L_s \cup L_t$$



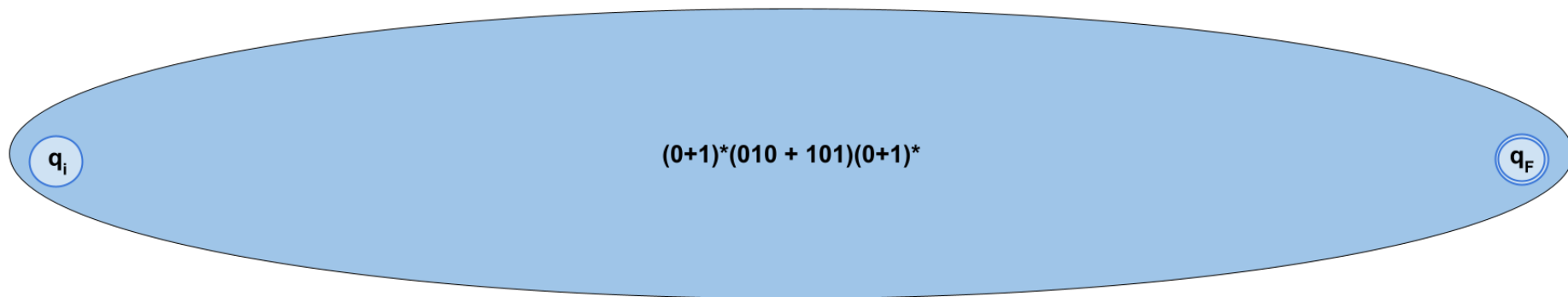
$$L = (L_s)^*$$



Regular expression to NFA example

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

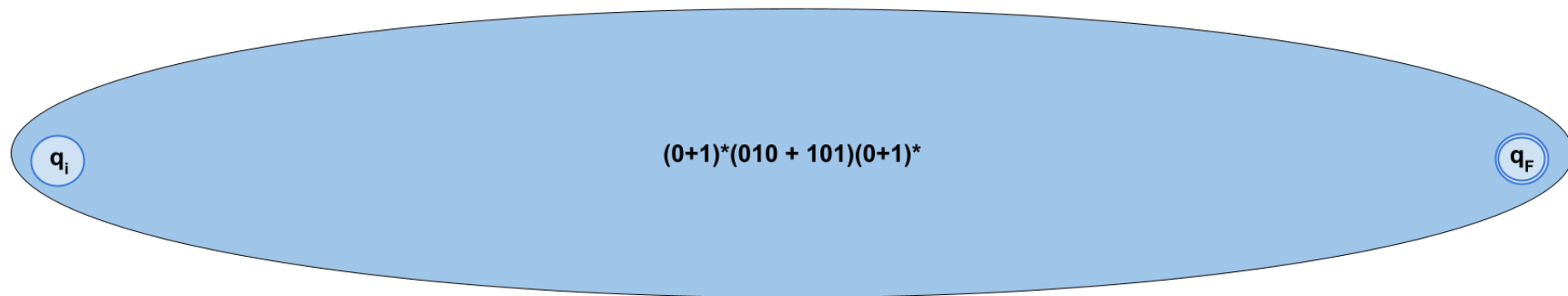
Example: $(0 + 1)^*(101 + 010)(0 + 1)^*$



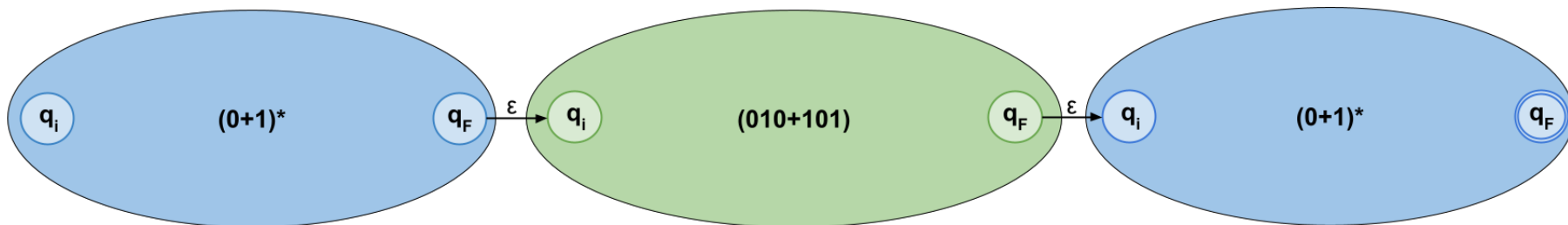
Regular expression to NFA example

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

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

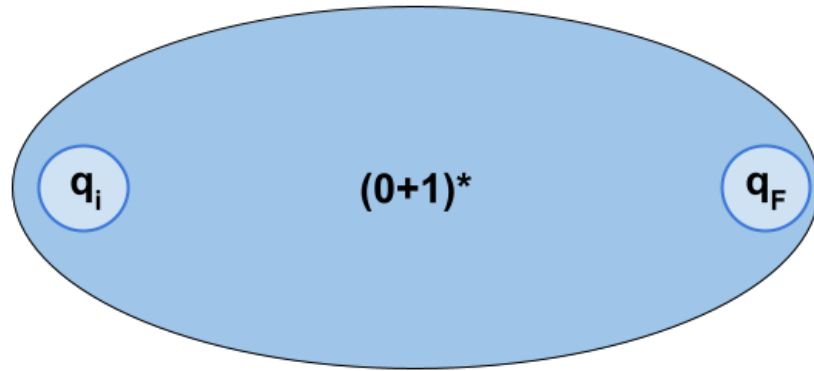


Using the concatenation rule:



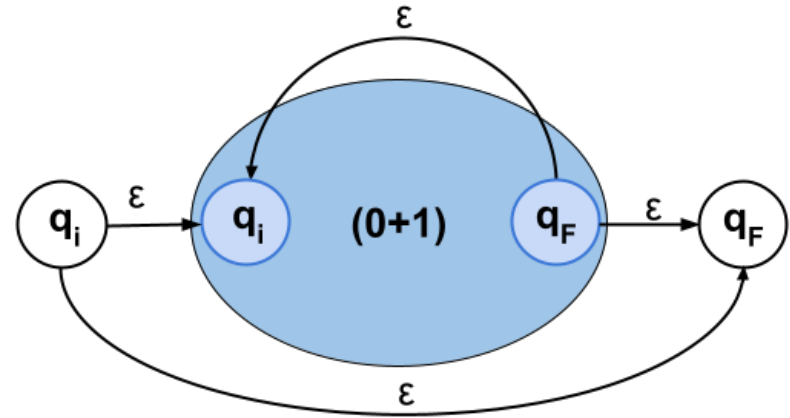
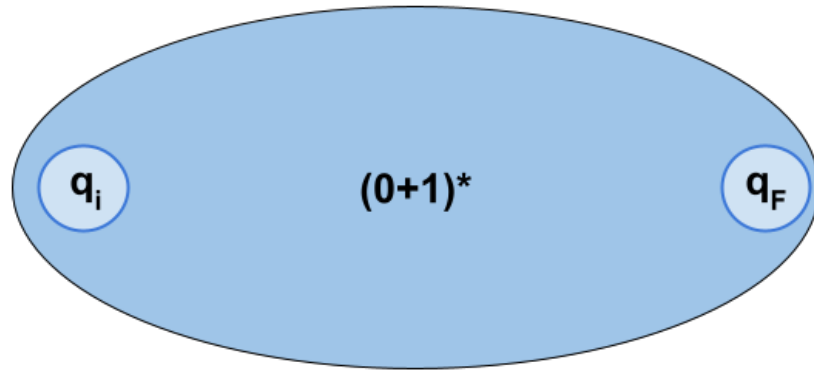
Regular expression to NFA example

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



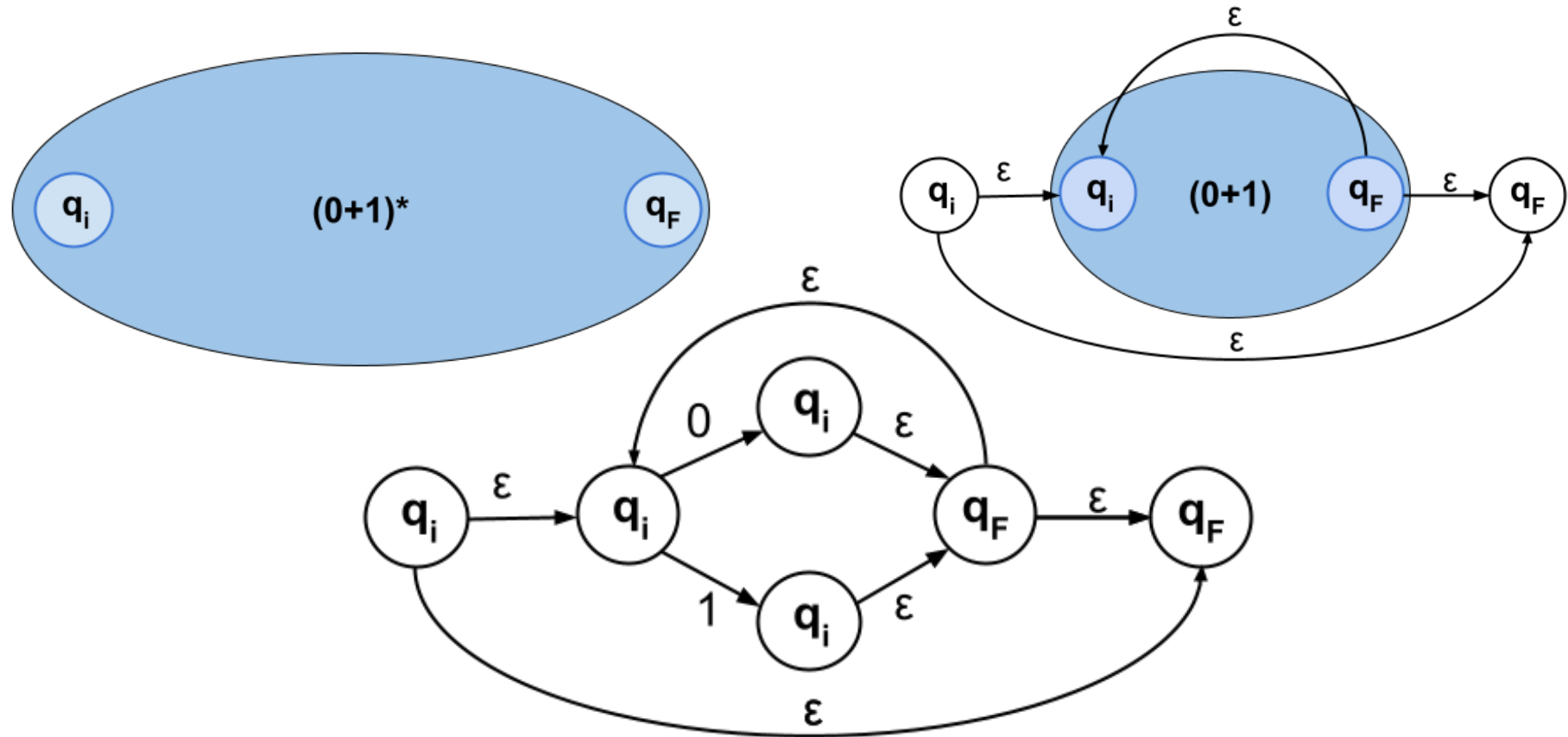
Regular expression to NFA example

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



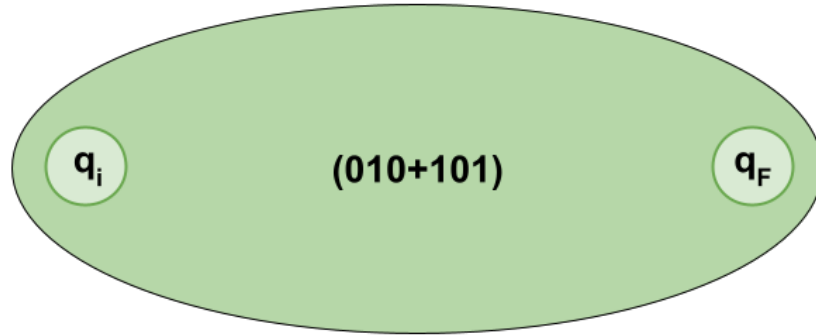
Regular expression to NFA example

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



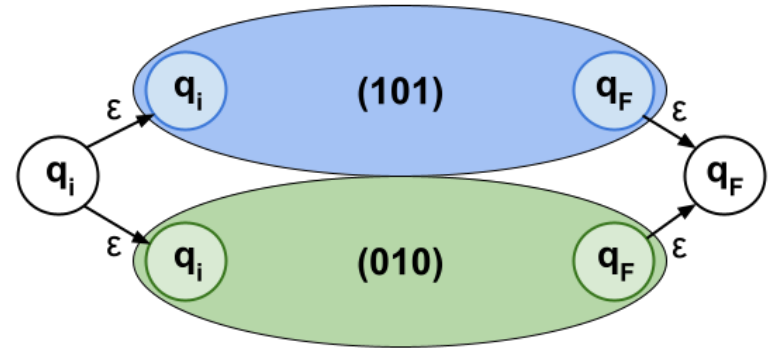
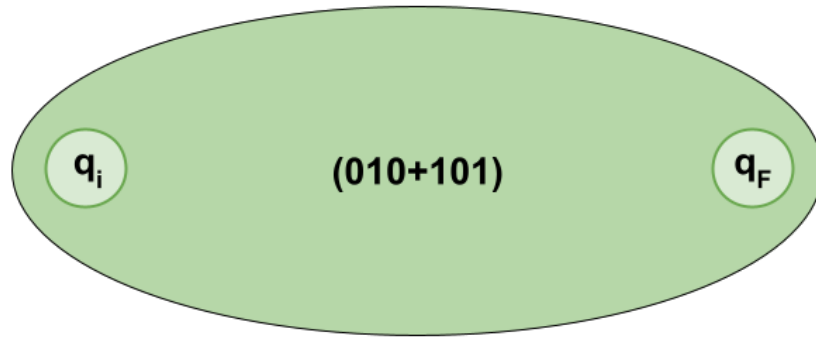
Regular expression to NFA example

Find NFA for $(101 + 010)$



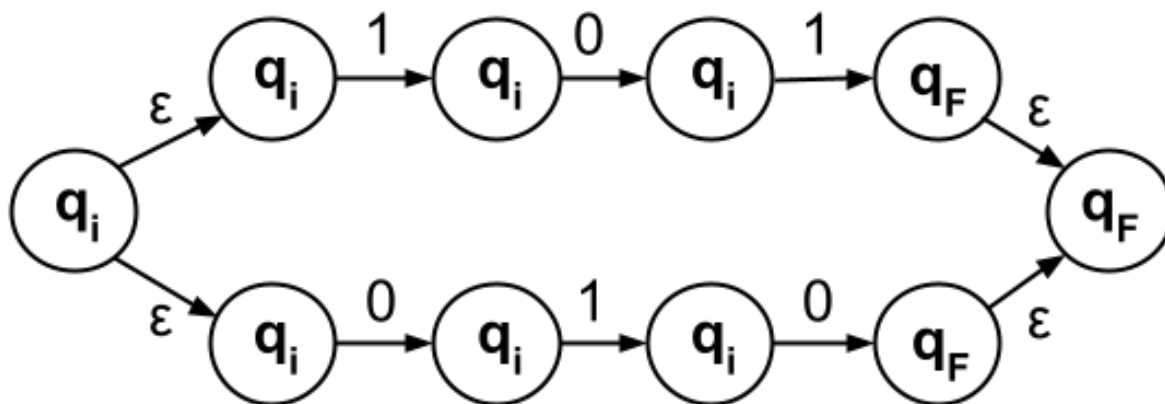
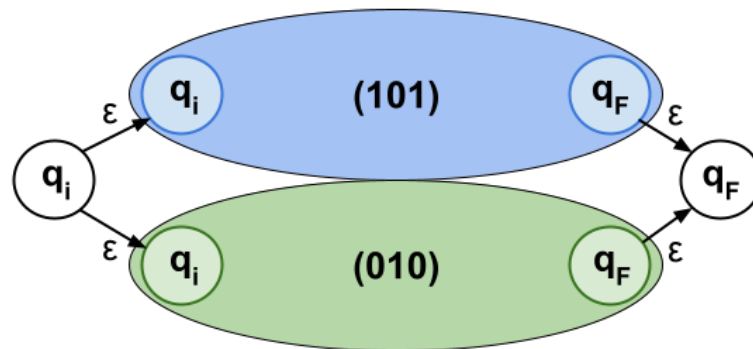
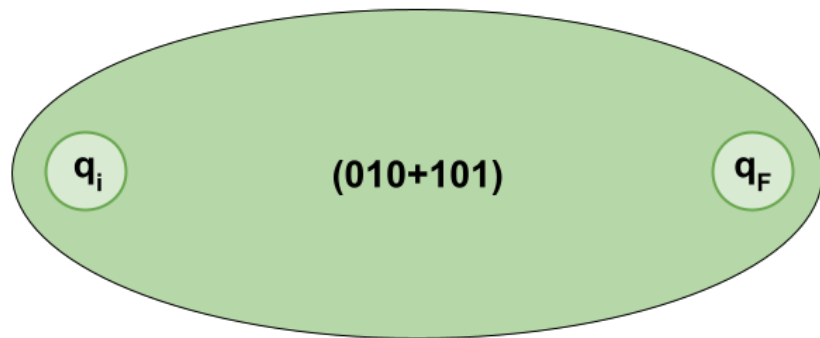
Regular expression to NFA example

Find NFA for $(101 + 010)$



Regular expression to NFA example

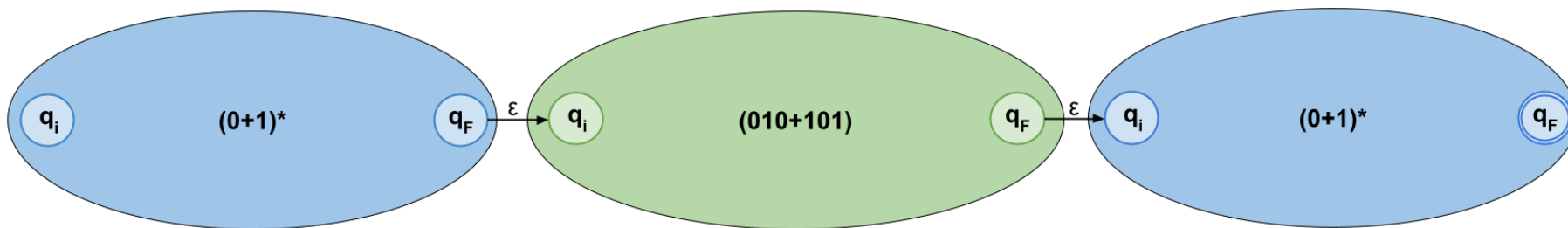
Find NFA for $(101 + 010)$



Regular expression to NFA example

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

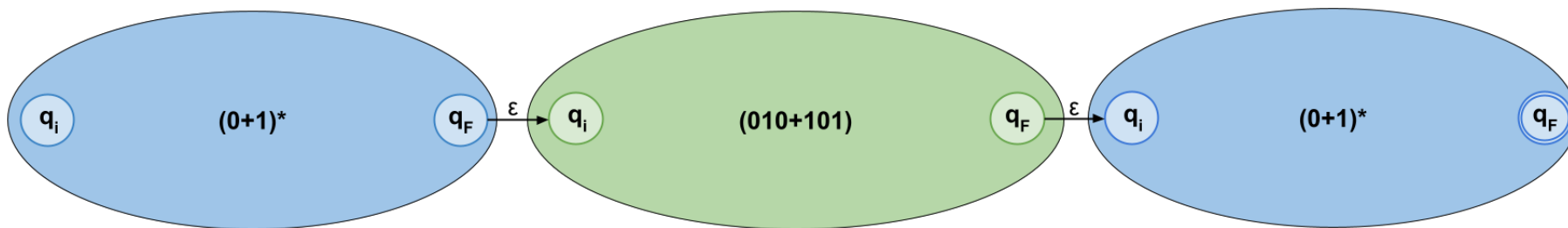
Example: $(0 + 1)^*(101 + 010)(0 + 1)^*$



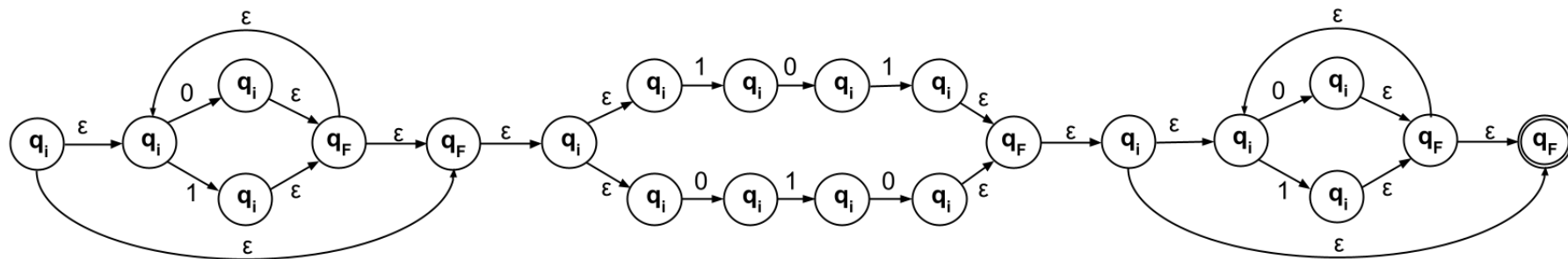
Regular expression to NFA example

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

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



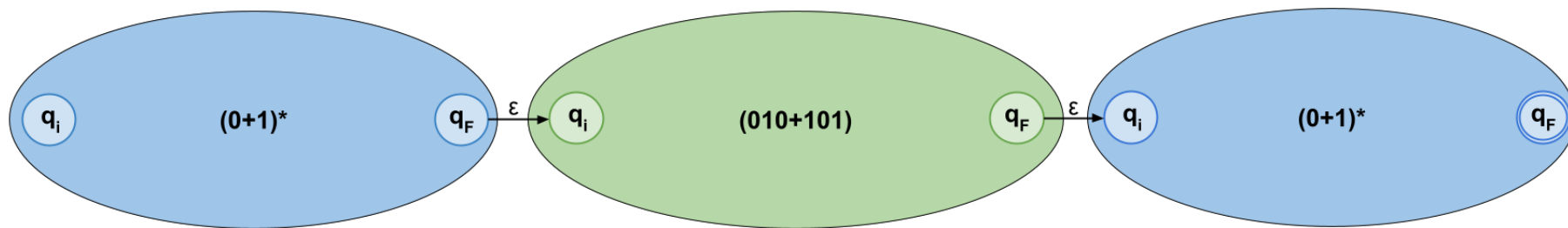
Using the concatenation rule:



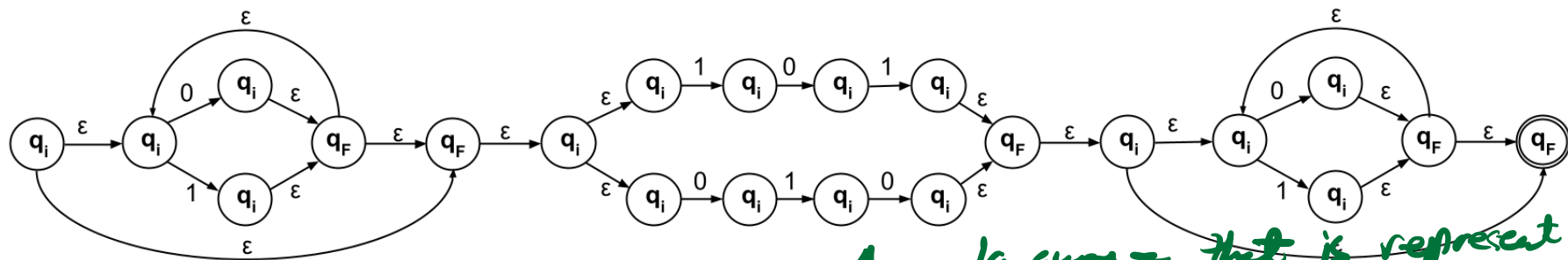
Regular expression to NFA example

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

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



Using the concatenation rule:

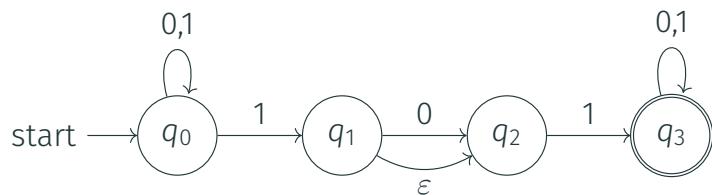


What does Thompson's algorithm mean?!

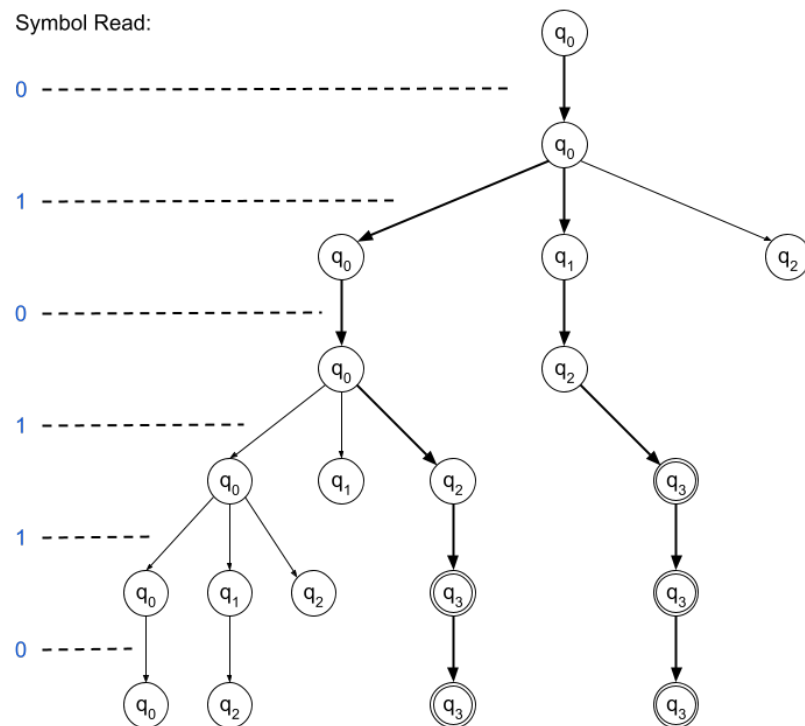
Any language that is representable by a regex is also representable by a NFA.

Equivalence of NFAs and DFAs

Another Way to look at NFAs

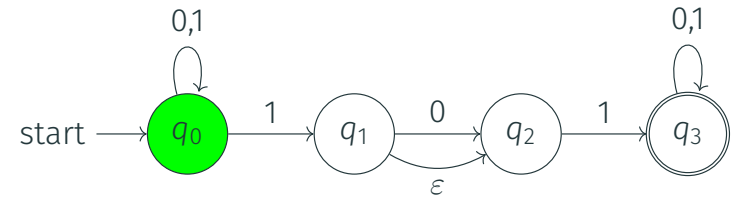


Is 010110 accepted?



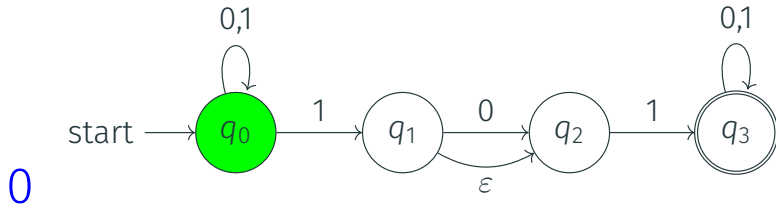
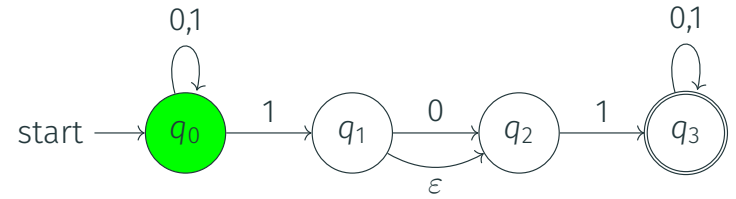
Another Way to look at NFAs

Is **010110** accepted?



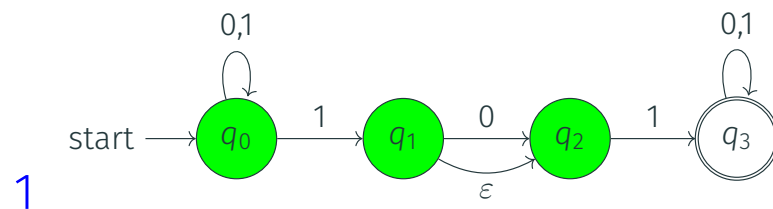
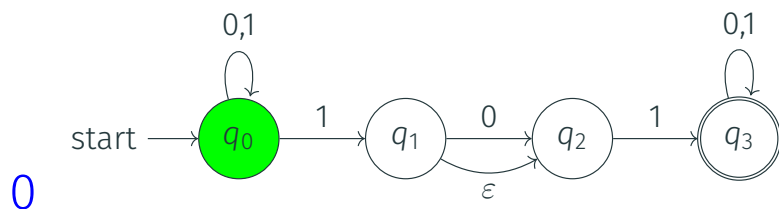
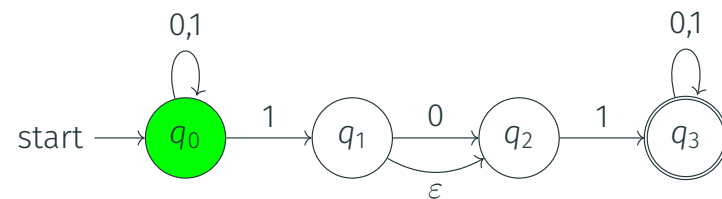
Another Way to look at NFAs

Is 010110 accepted?



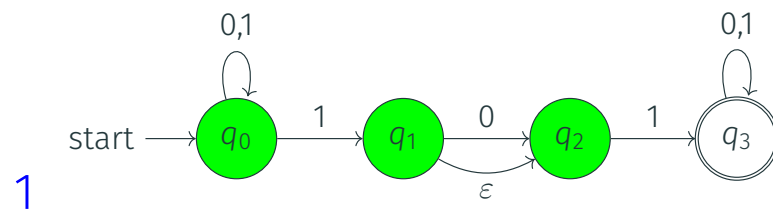
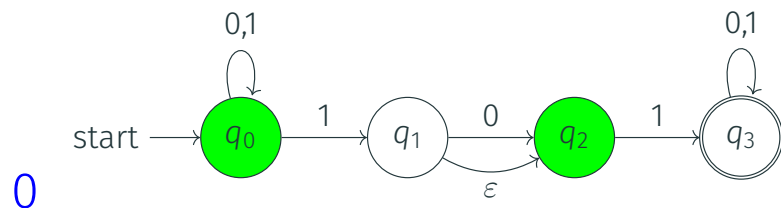
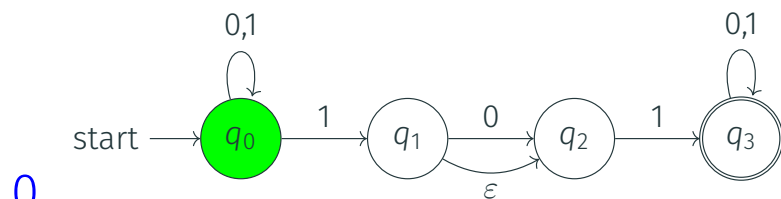
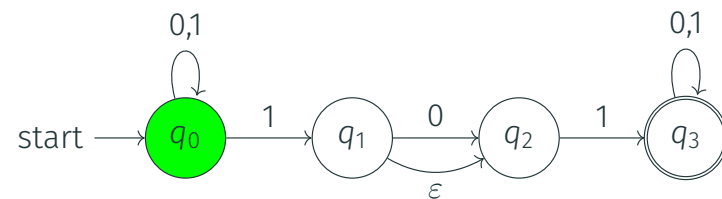
Another Way to look at NFAs

Is **010110** accepted?



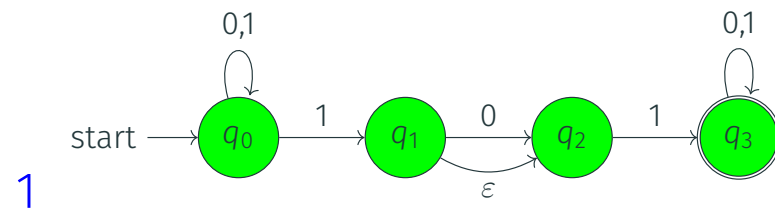
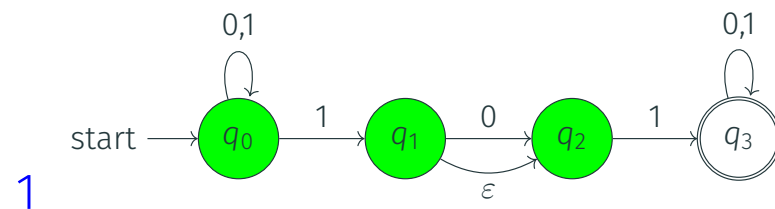
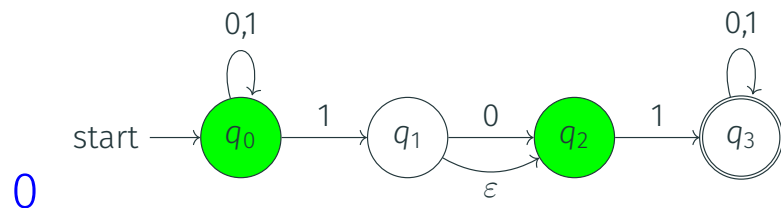
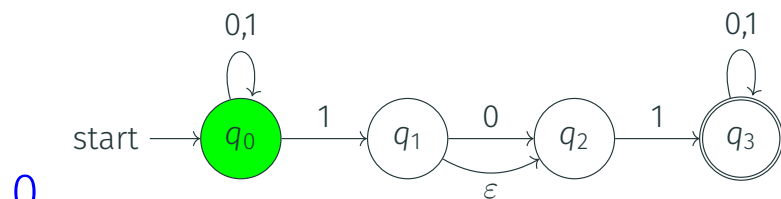
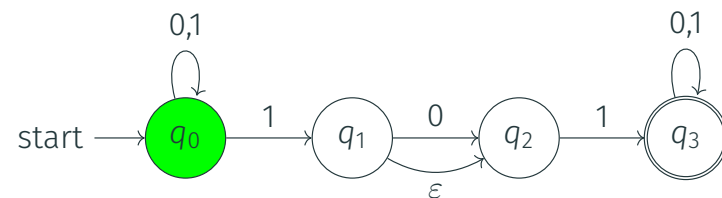
Another Way to look at NFAs

Is 010110 accepted?



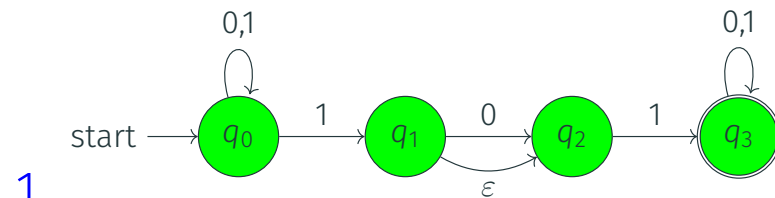
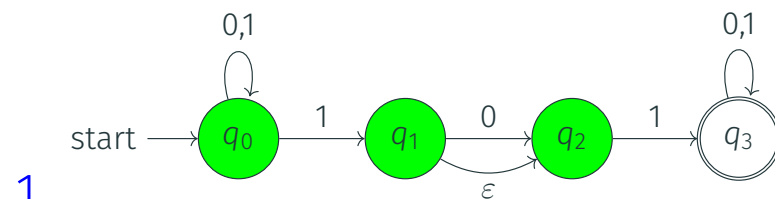
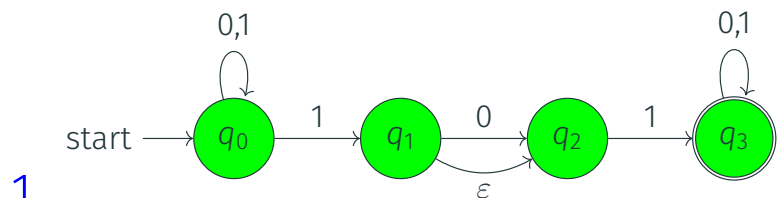
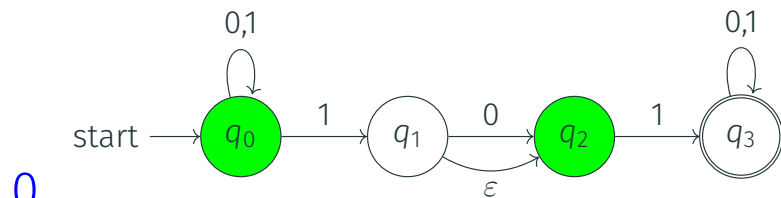
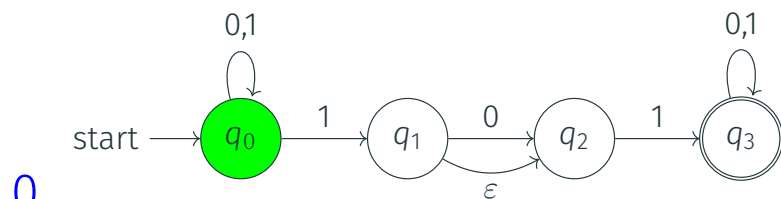
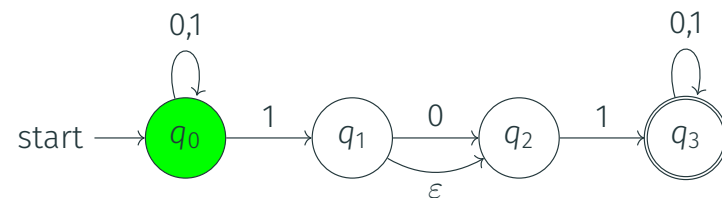
Another Way to look at NFAs

Is 010110 accepted?



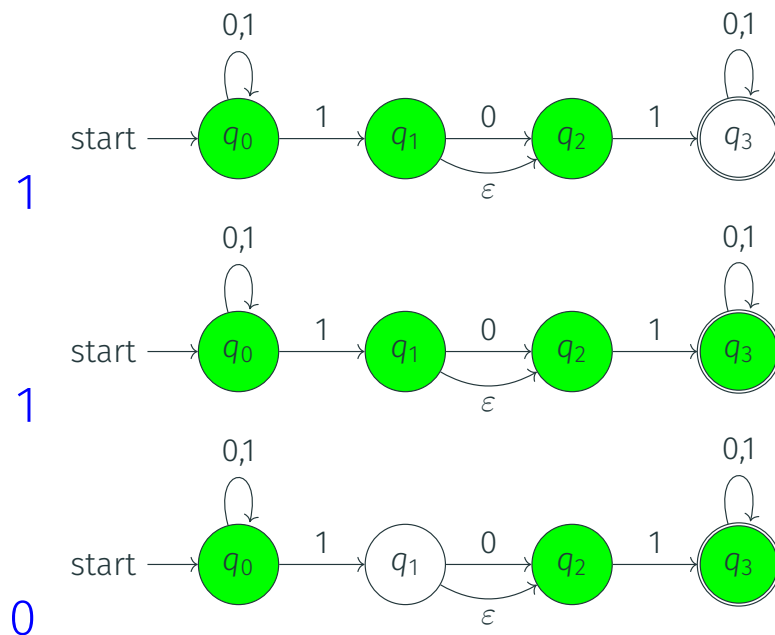
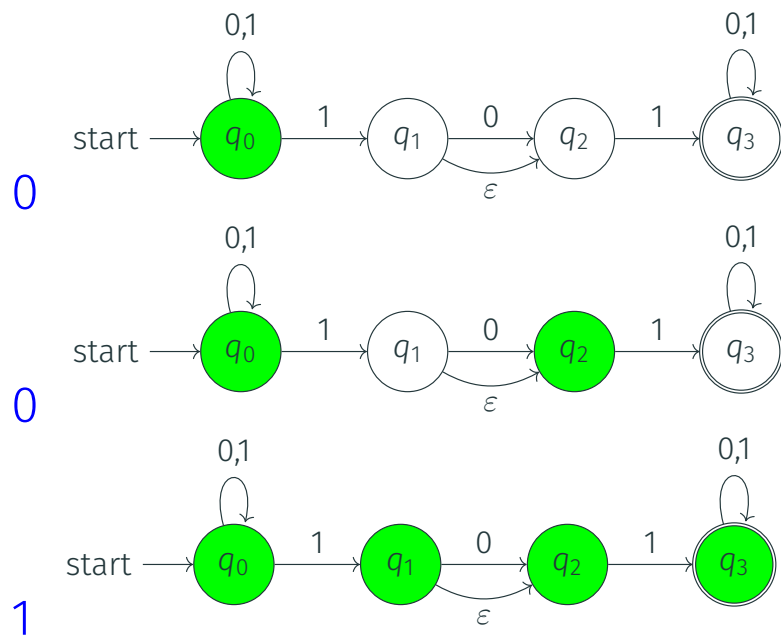
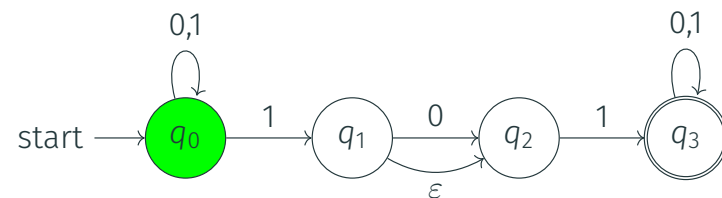
Another Way to look at NFAs

Is 010110 accepted?



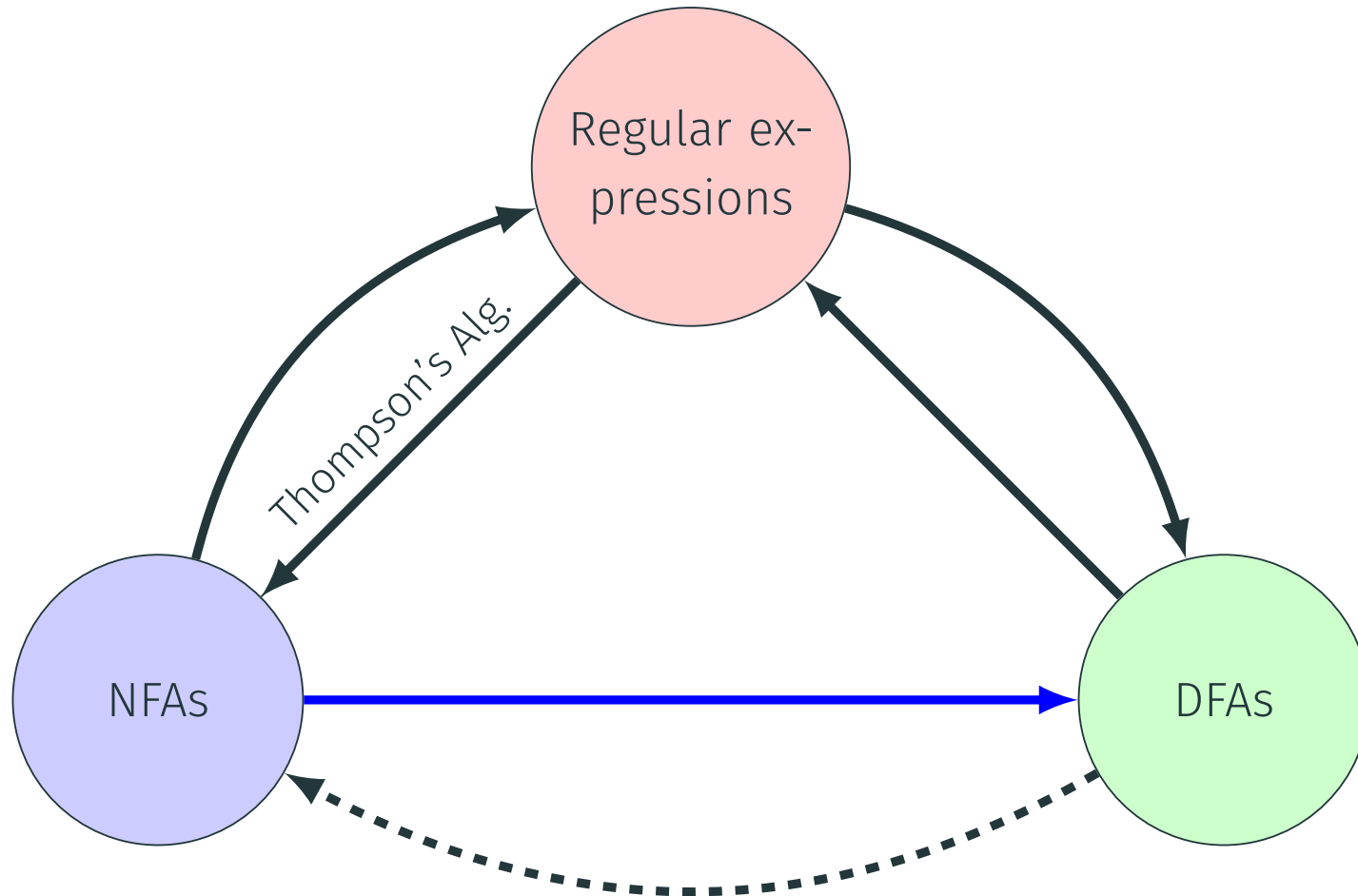
Another Way to look at NFAs

Is 010110 accepted?



Conversion of NFA to DFA

Proving equivalence



Equivalence of NFAs and DFAs

Theorem

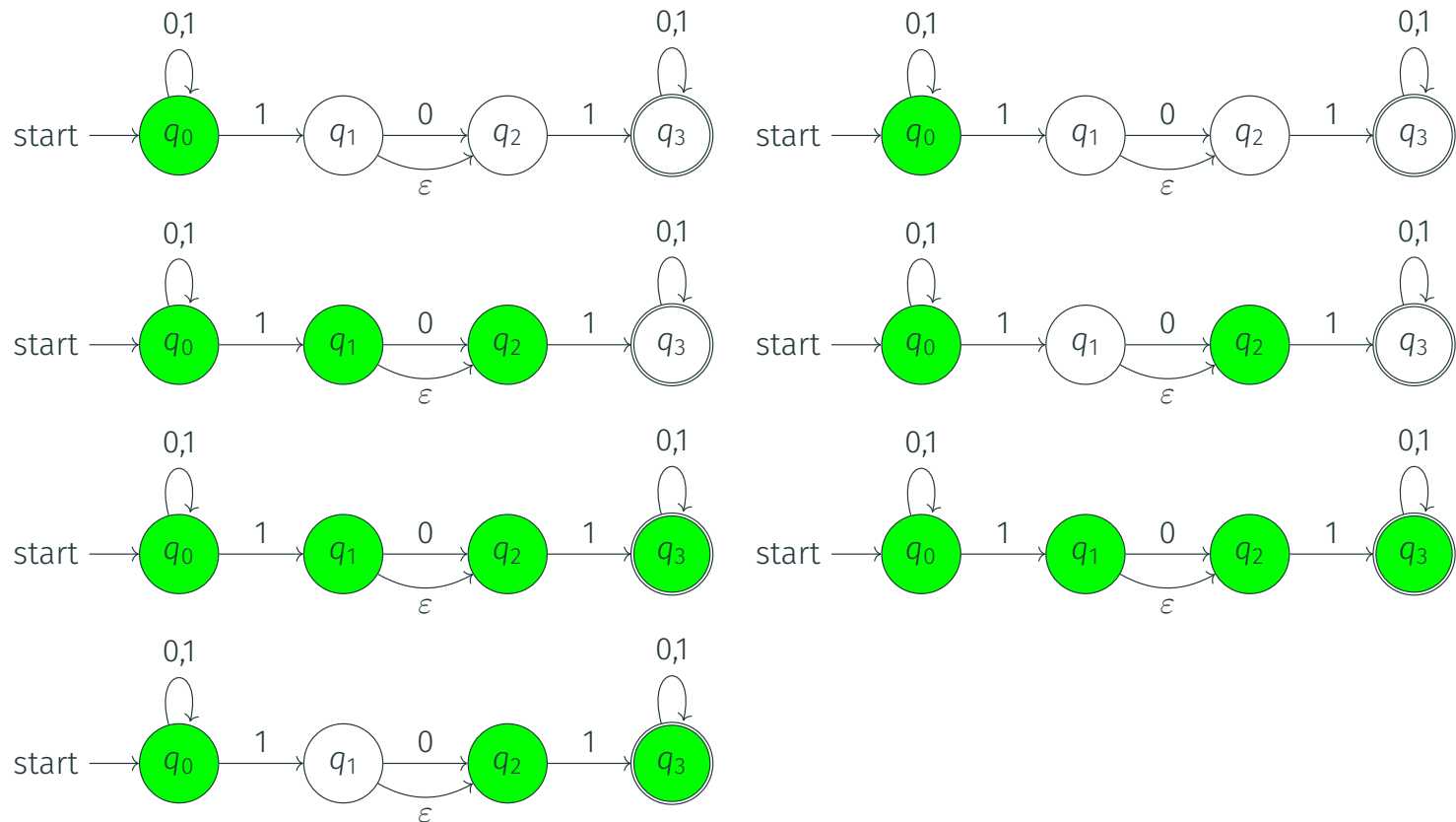
For every NFA N there is a DFA M such that $L(M) = L(N)$.

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

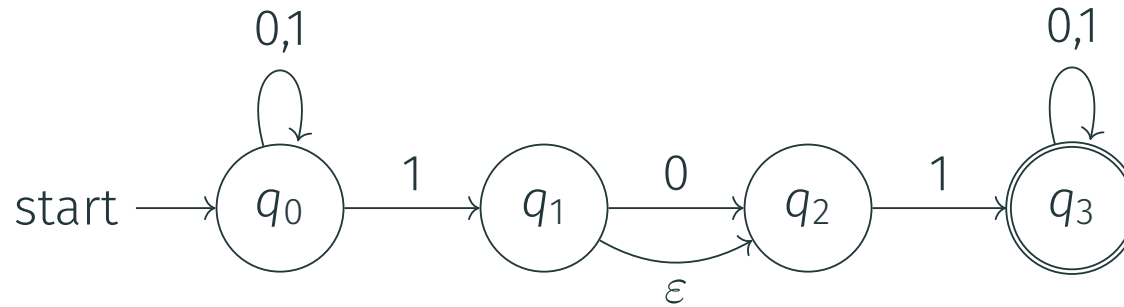
NFAs know many states at once on input 010110.



The state of the NFA

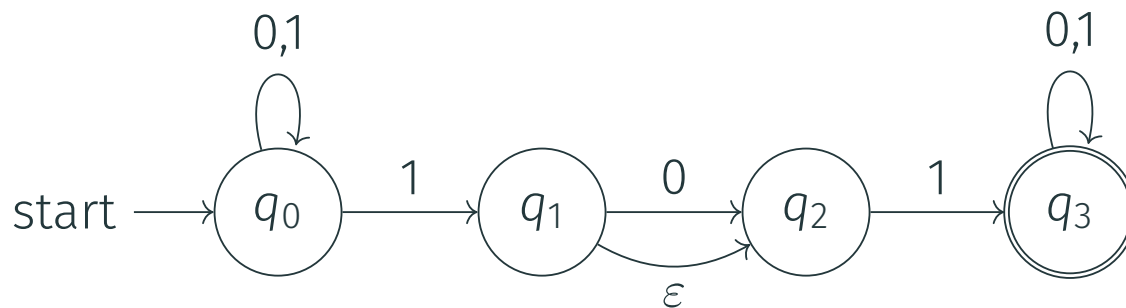
It is easy to state that the state of the automata is the states that it might be

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

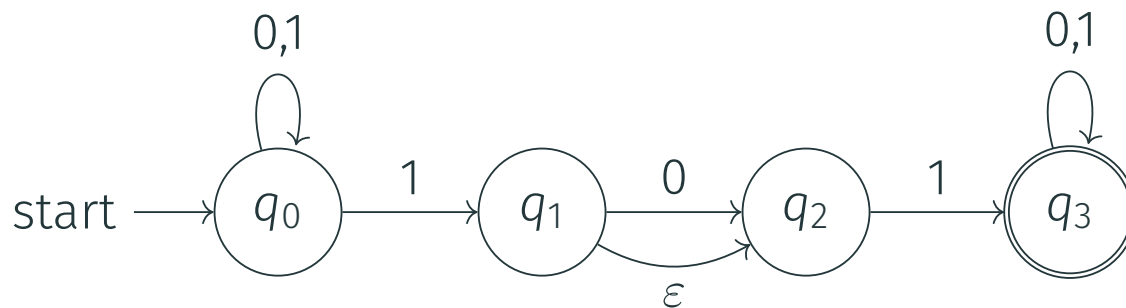


situated at.

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.

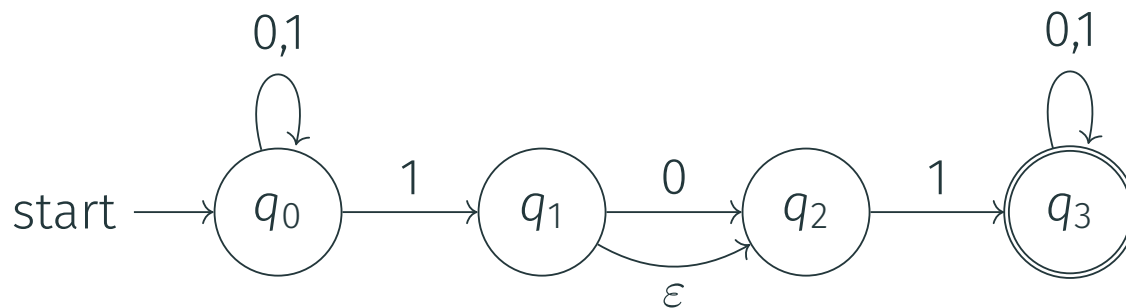
configuration: A set of states the automata might be in.

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

$$2^{10} = 1024$$

The state of the NFA

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



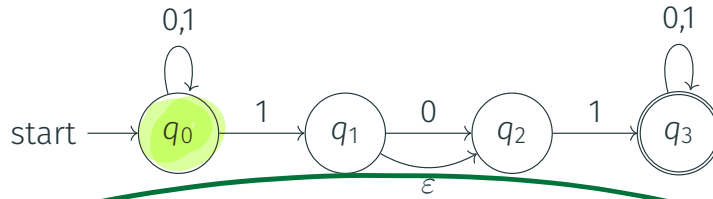
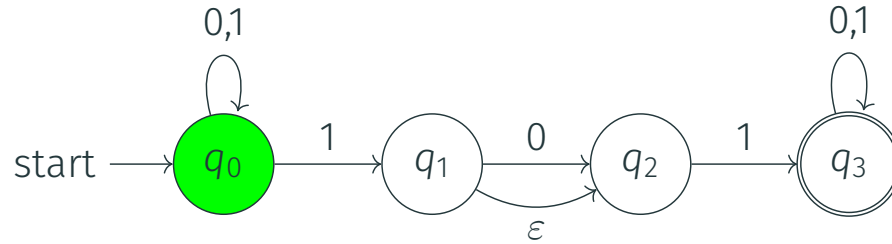
situated at.

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

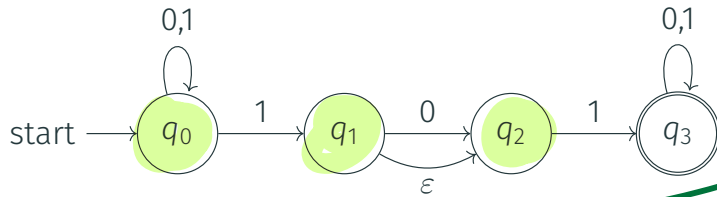
Possible configurations: $\mathcal{P}(q) = \emptyset, \{q_0\}, \{q_0, q_1\} \dots$

Big idea: Build a **DFA** on the configurations.

Example

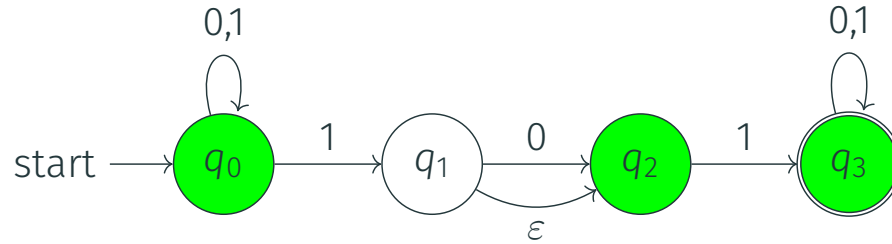


If receives 0 :

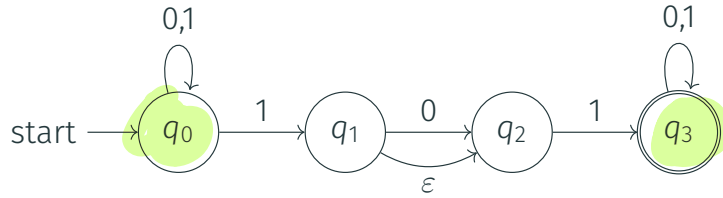


If receives 1 :

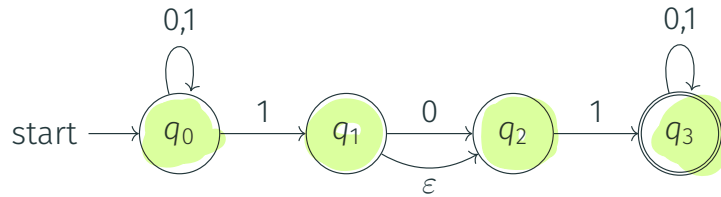
Example



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?

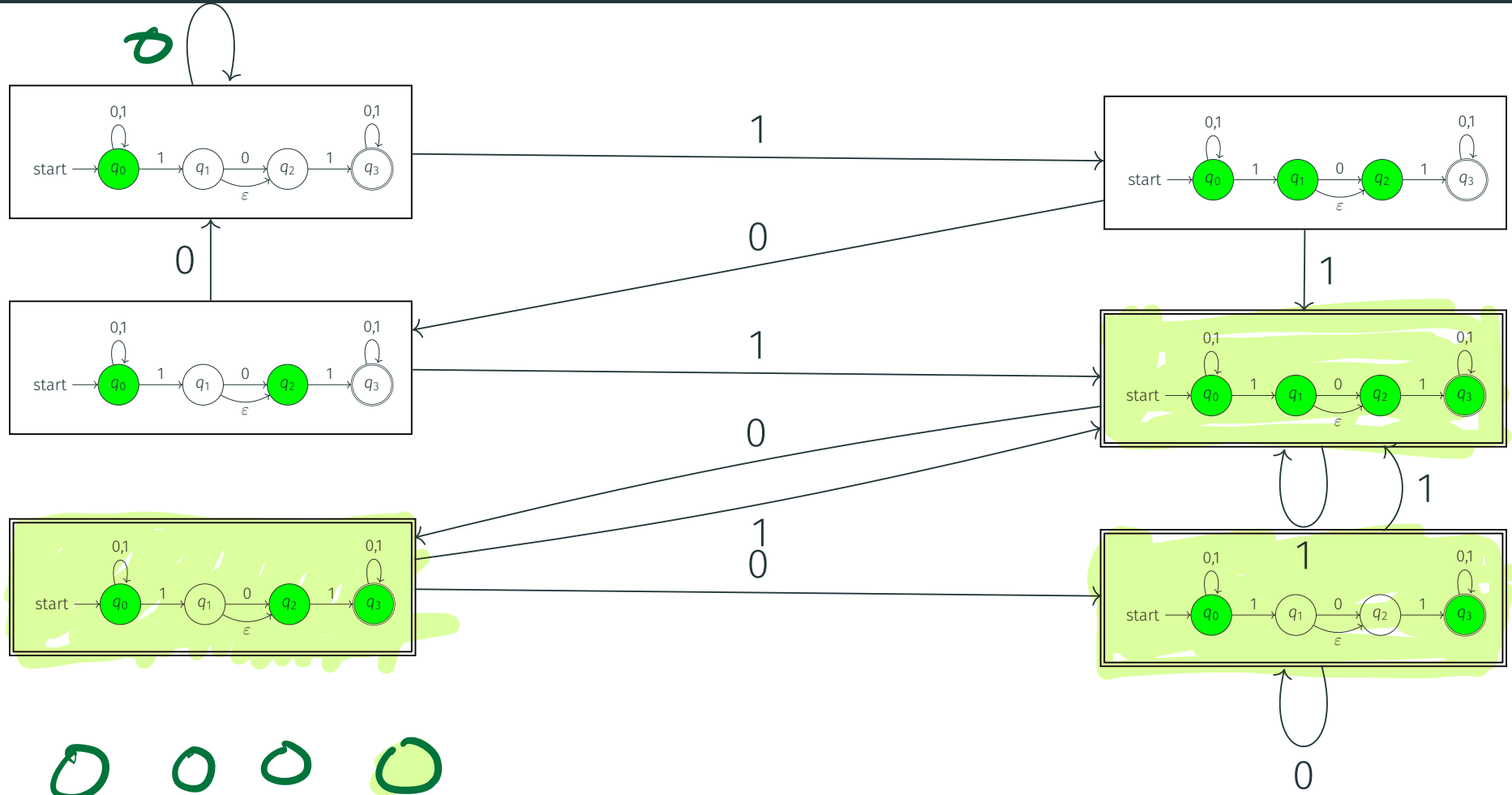
Simulating an NFA by a DFA

- 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 $\delta^*(s, w) \cap A \neq \emptyset$.

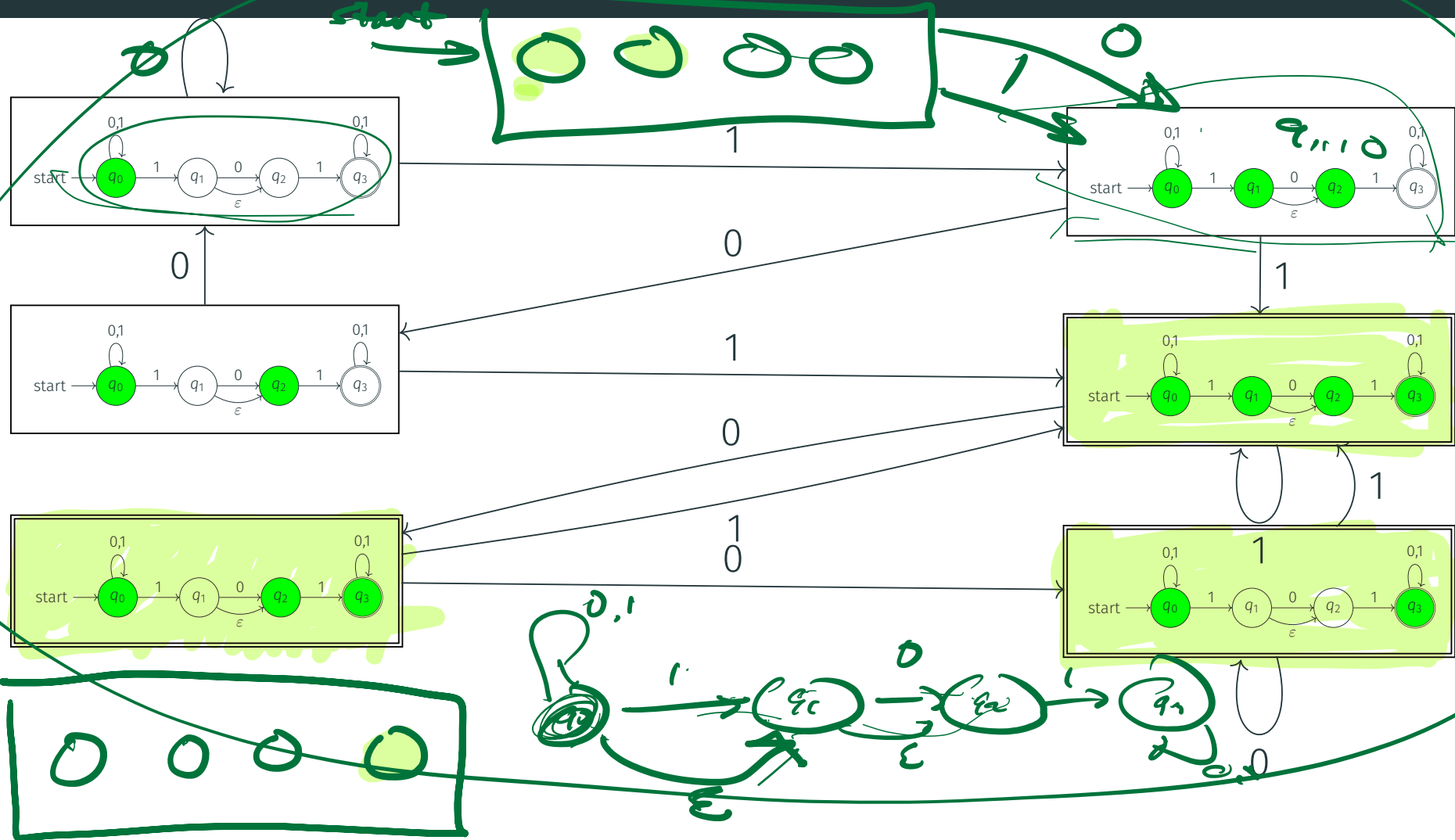
Key Observation: DFA M simulating N should know current configuration of N .

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

DFA from NFA



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

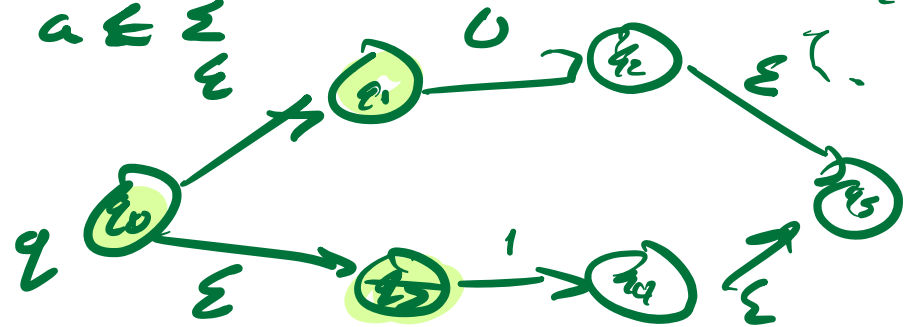
- Q is a finite set whose elements are called **states**,
- Σ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the **start state**,
- $A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

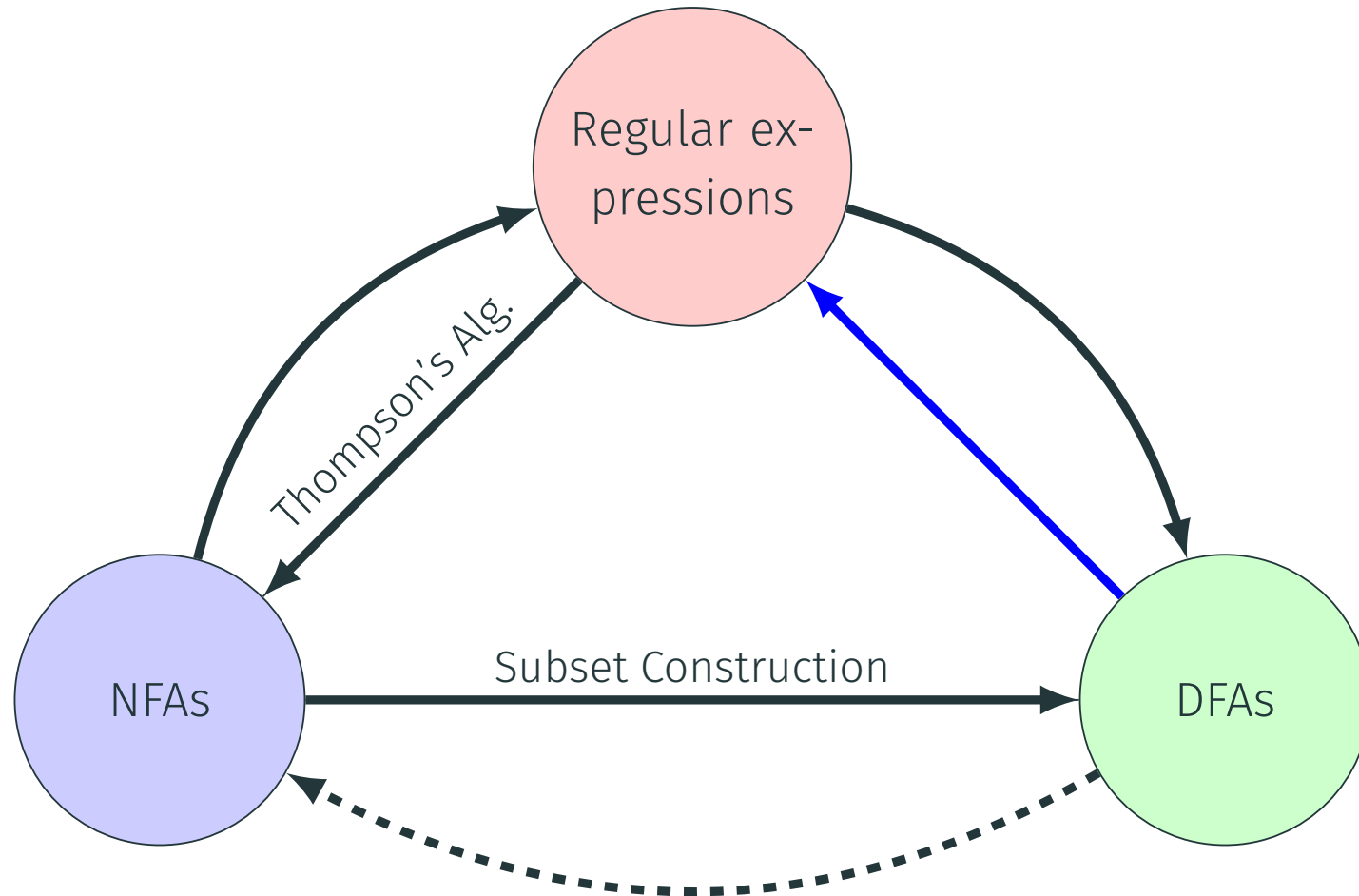
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a **DFA** $D = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' =$ one state representing each subset of Q : $P(Q)$
- $s' = \text{Evens}(s) \equiv \delta^*(s, \epsilon)$
- $A' = \{x \subset Q \mid x \cap A \neq \emptyset\}$
- $\delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)$ for each $X \subset Q$
 $a \in \Sigma$
- $\Sigma' = \Sigma$



DFAs to Regular expressions

Proving equivalence

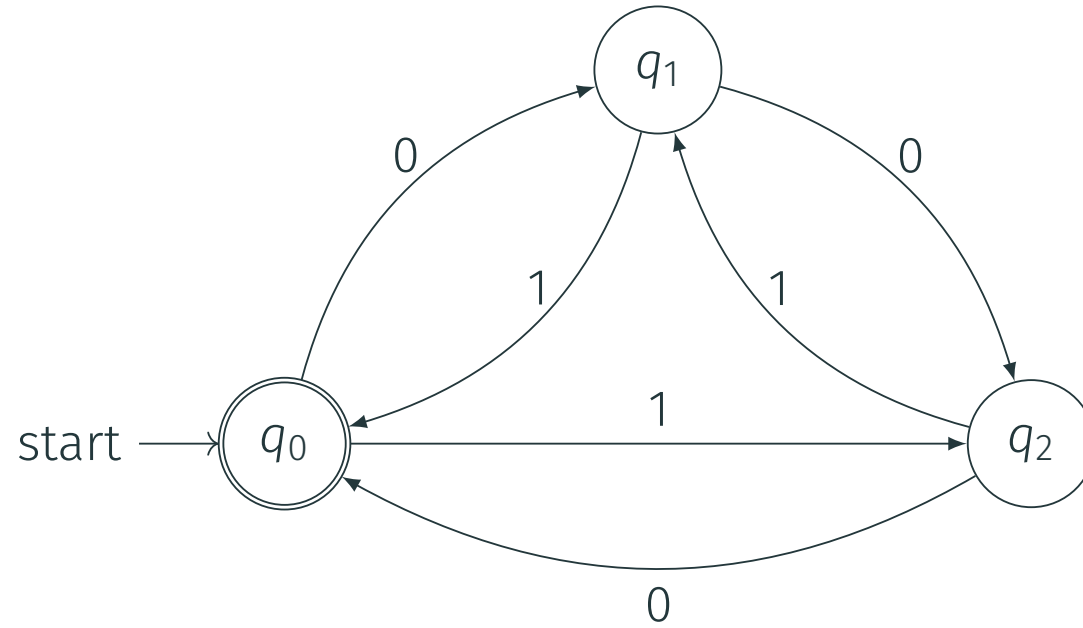


State Removal method

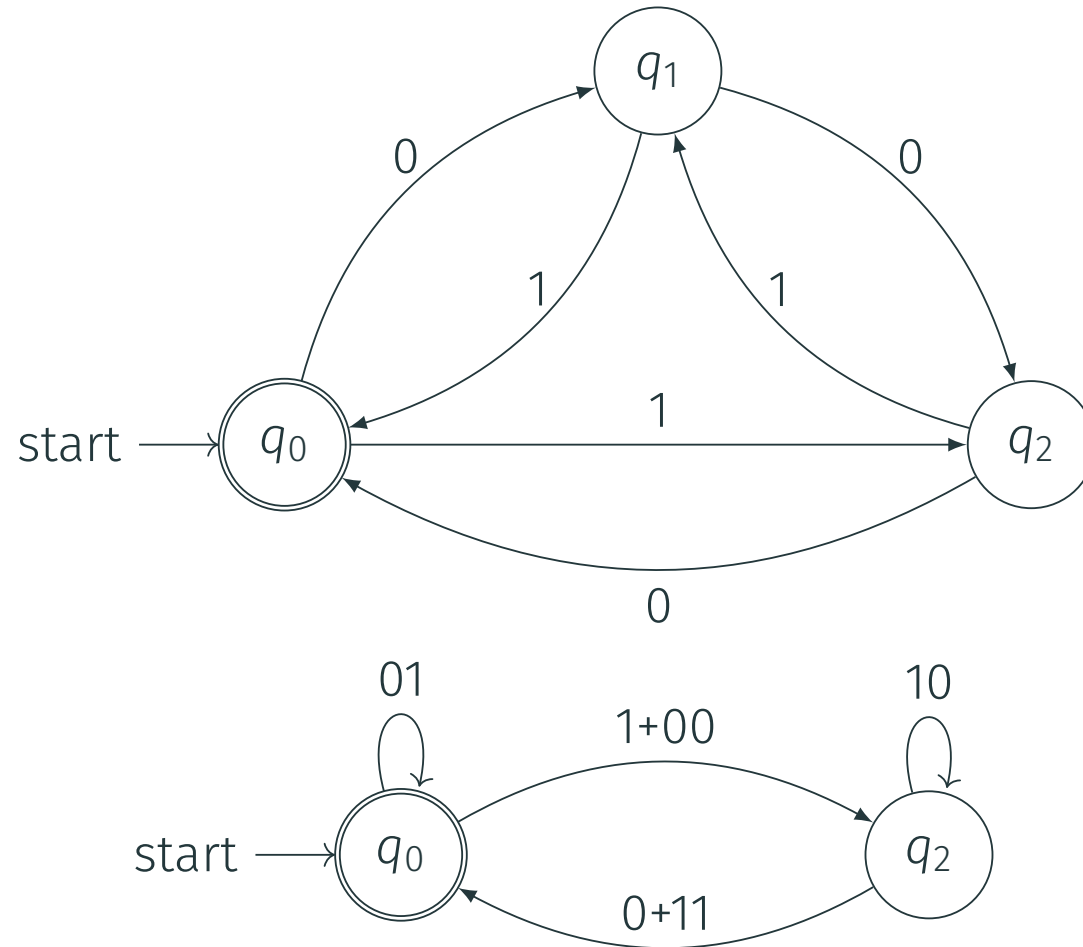
If $q_1 = \delta(q_0, x)$ and $q_2 = \delta(q_1, y)$

then $q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta(q_0, xy)$

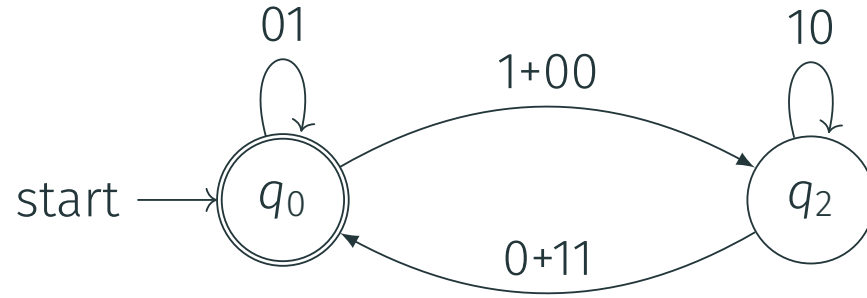
State Removal method - Example



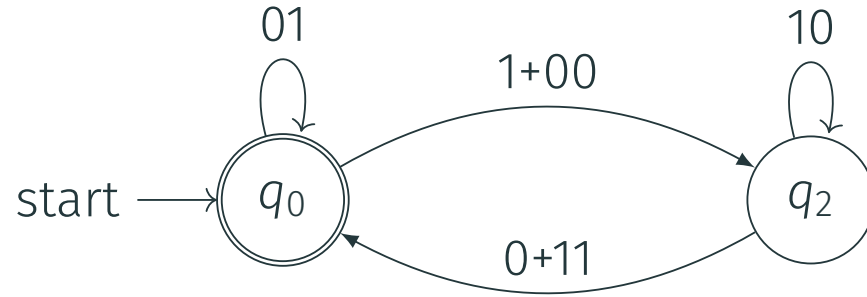
State Removal method - Example



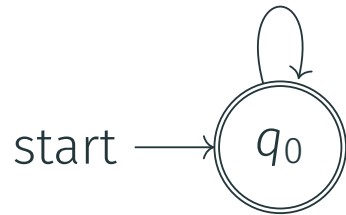
State Removal method - Example



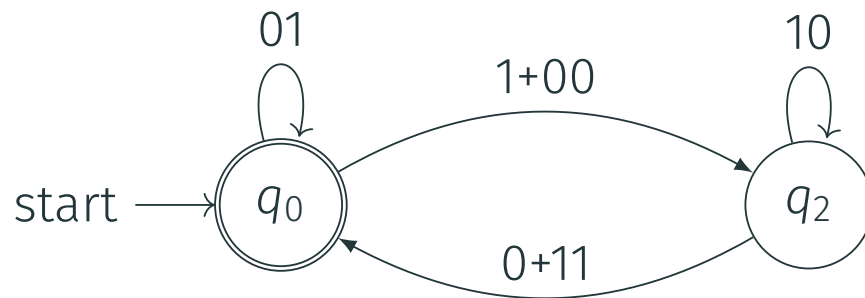
State Removal method - Example



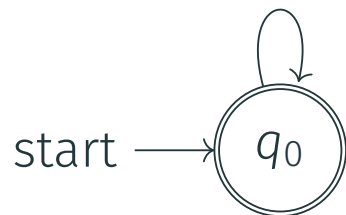
$$01 + (1 + 00)(10)^*(0 + 11)$$



State Removal method - Example



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



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

Algebraic method

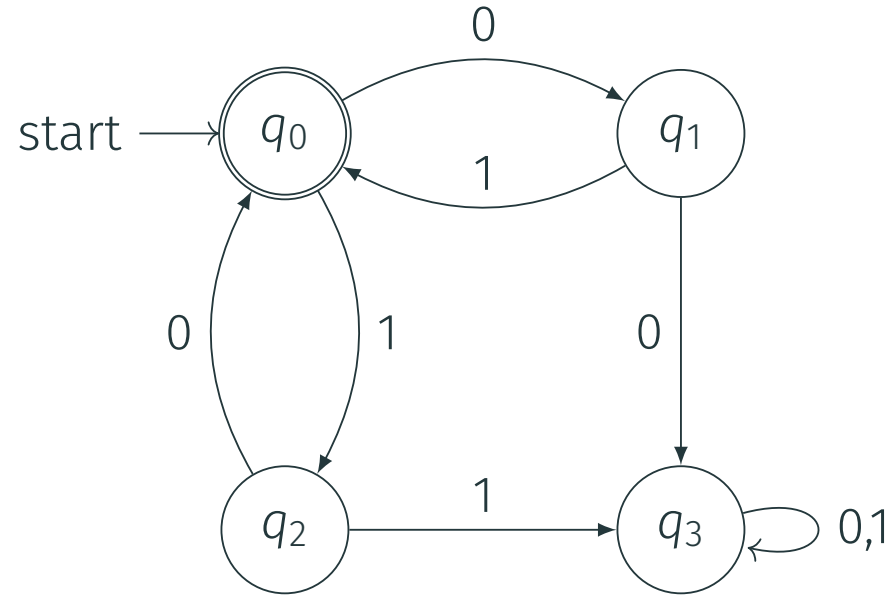
Transition functions are themselves algebraic expressions!

Demarcate states as variables.

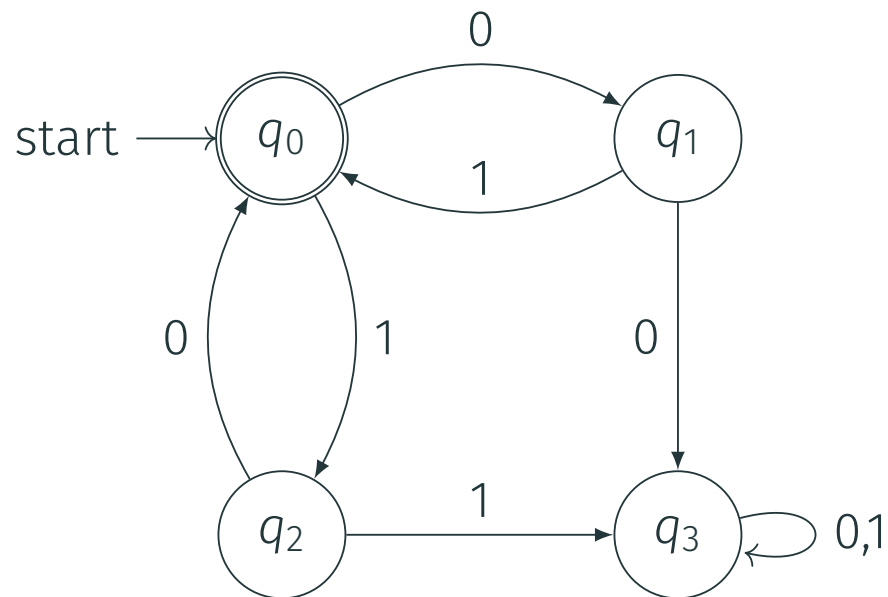
Can rewrite $q_1 = \delta(q_0, x)$ as $q_1 = q_0x$

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 simply 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

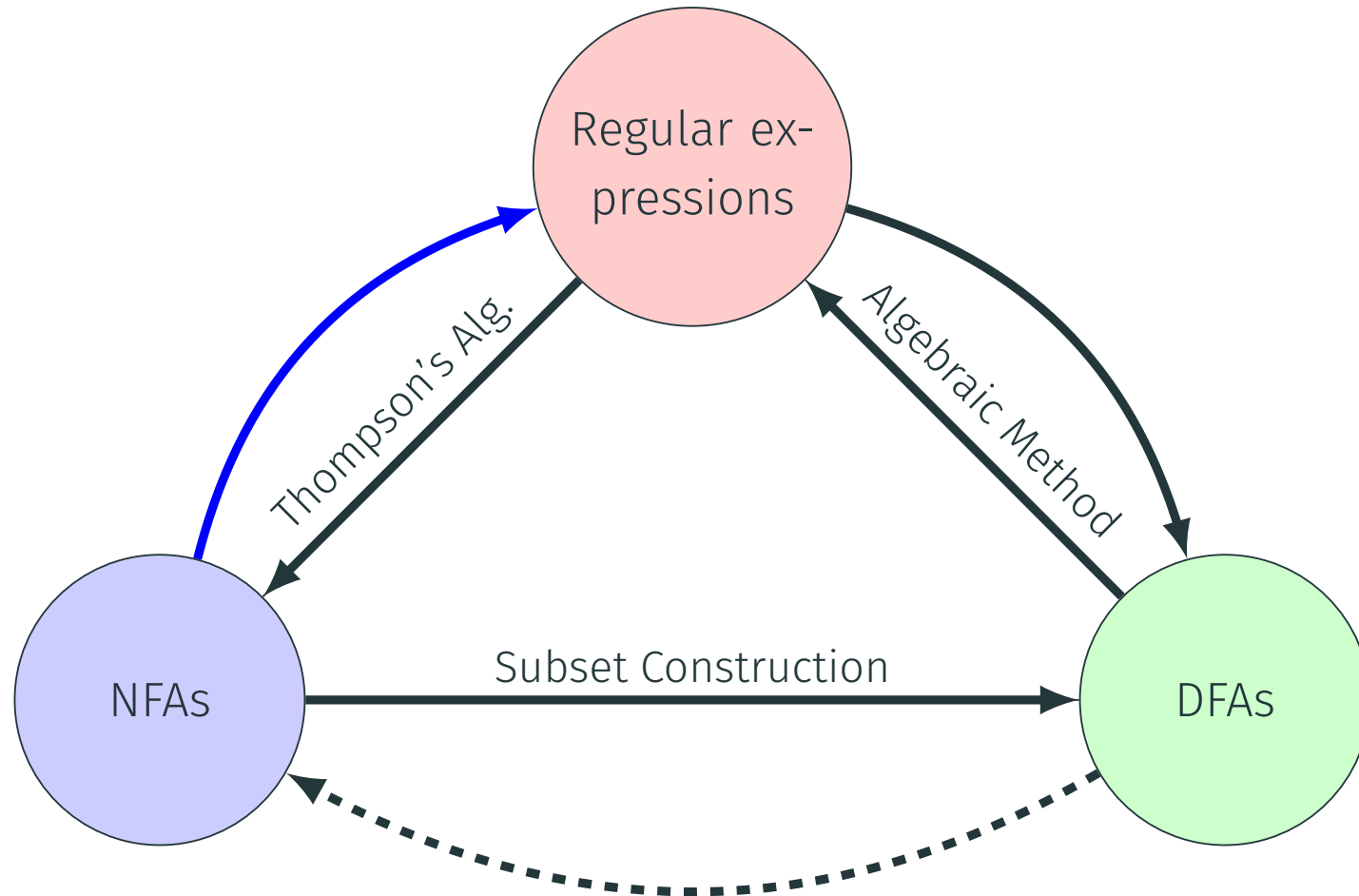
- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3(0 + 1)$

Now we simply solve the system of equations for q_0 :

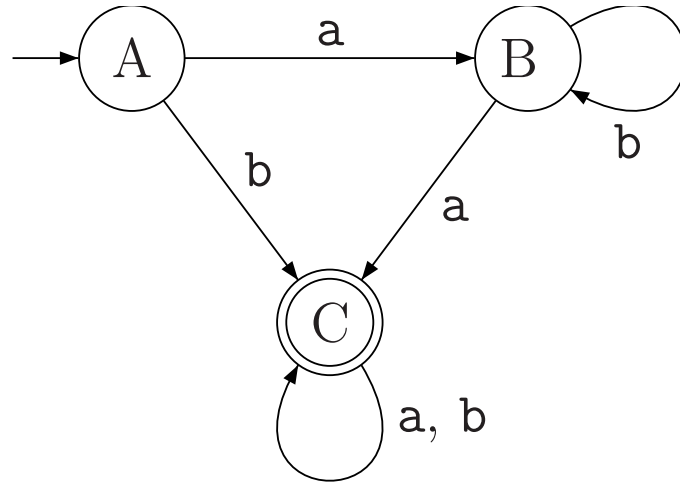
- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0(01 + 10)$
- $q_0 = \epsilon(01 + 10)^* = (01 + 10)^*$

Converting NFAs to Regular Expression

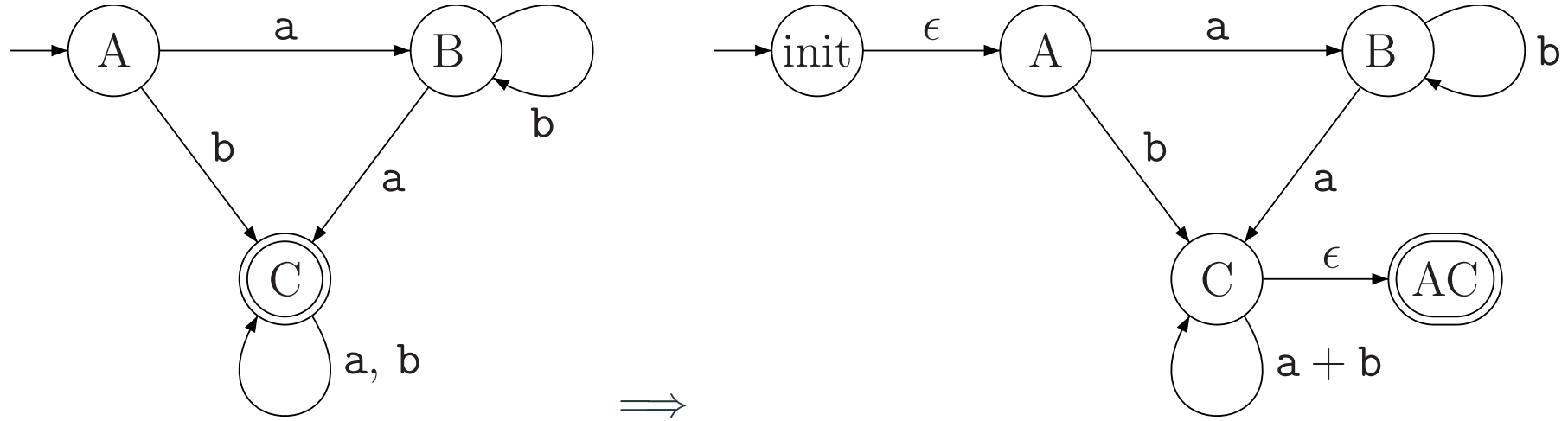
Proving equivalence



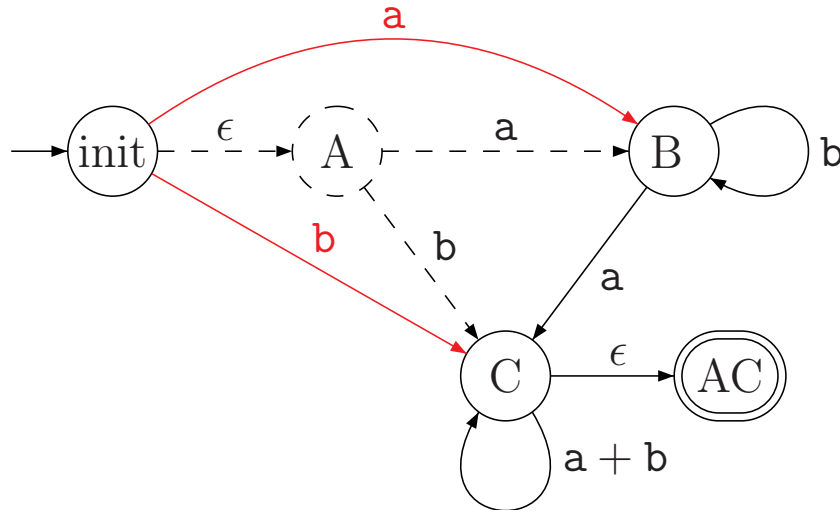
Stage 0: Input



Stage 1: Normalizing

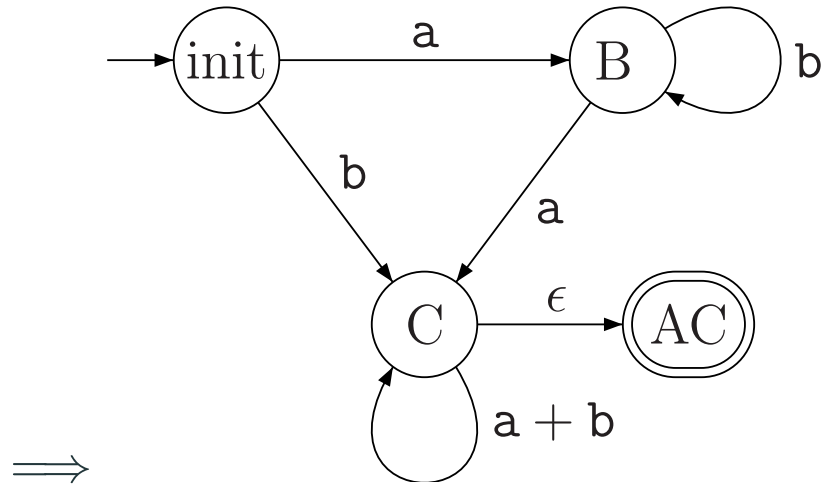


Stage 2: Remove state A

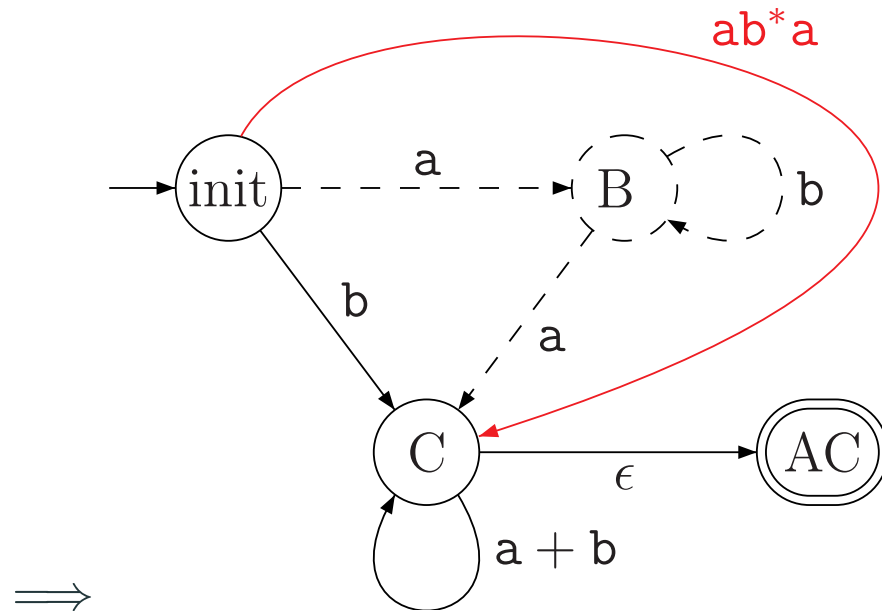


\Rightarrow

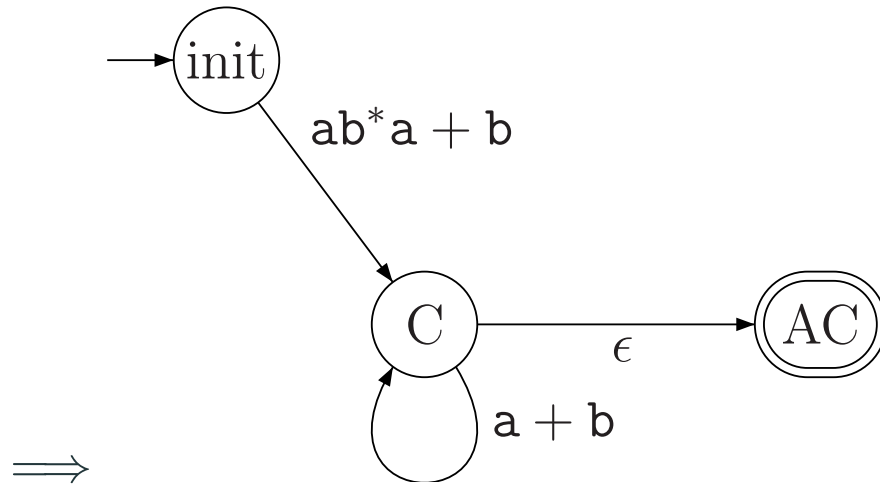
Stage 4: Redrawn without old edges



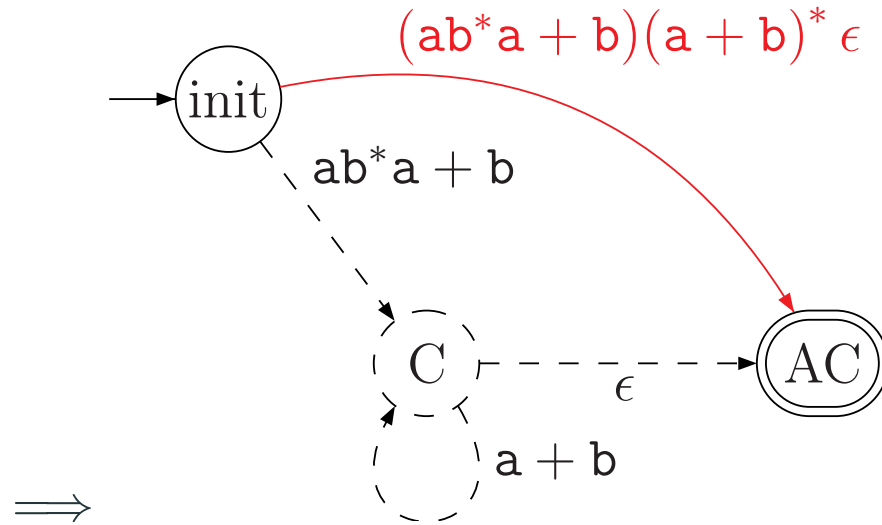
Stage 4: Removing B



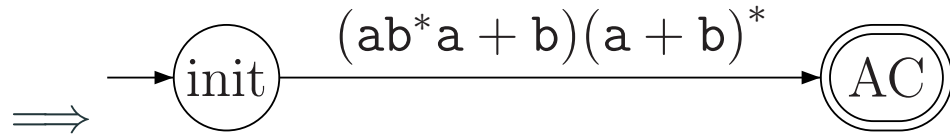
Stage 5: Redraw



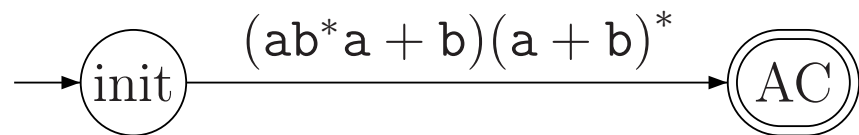
Stage 6: Removing C



Stage 7: Redraw



Stage 8: Extract regular expression

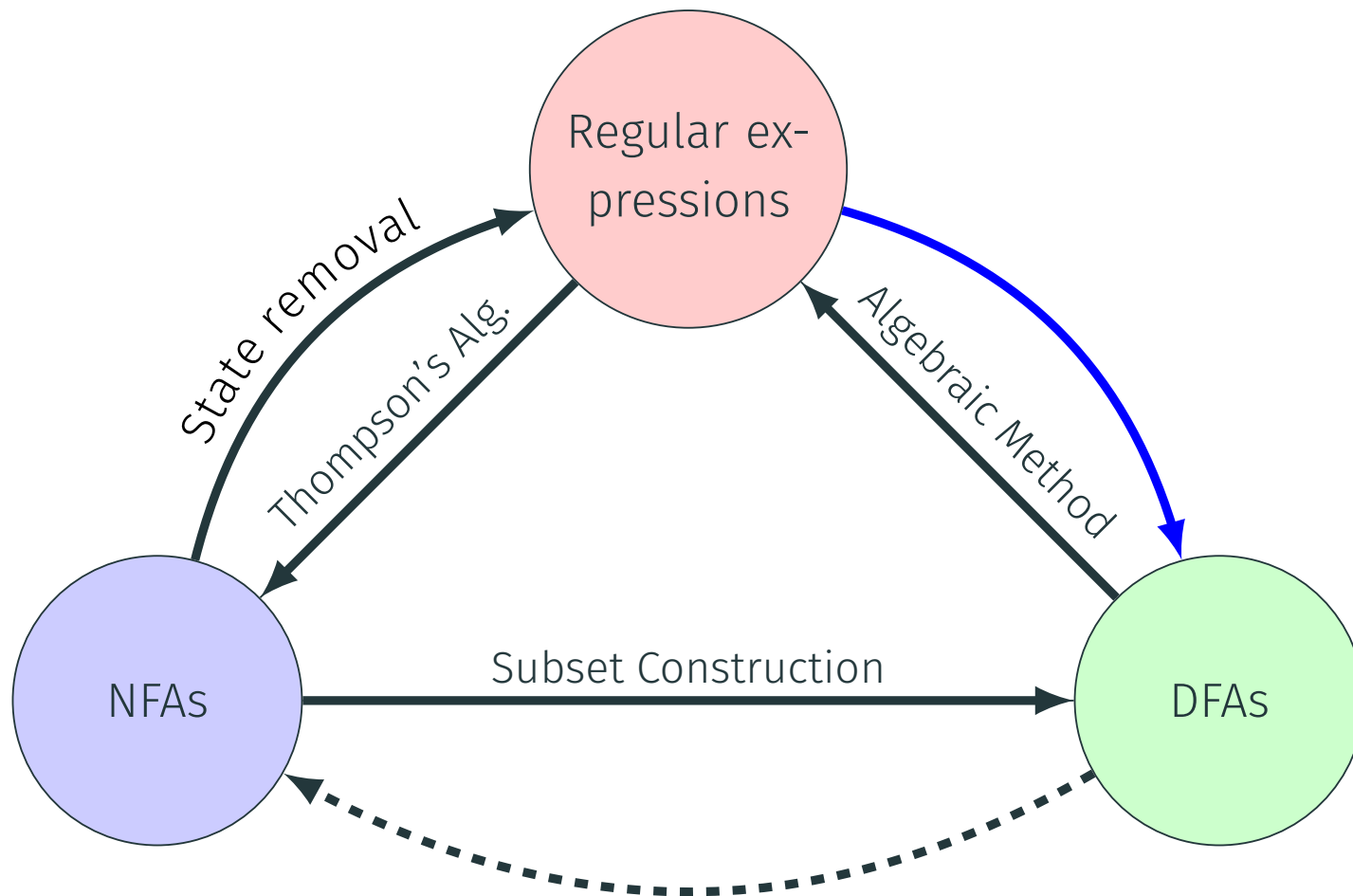


Thus, this automata is equivalent to the regular expression

$$(ab^*a + b)(a + b)^*.$$

Regular expressions to DFAs

Proving equivalence



Difficulty going from RegEx's to DFAs

Lemma

Many regular expressions cannot be easily converted to DFAs.

Difficulty going from RegEx's to DFAs

Lemma

Many regular expressions cannot be easily converted to DFAs.

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

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 Brzozowski¹ algorithm. Not needed for this course.

But here's the idea anyway....

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

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 $R = (ab + c)^*$

Brzowski Method

Brings us to the **Brzowski 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 $R = (ab + c)^*$

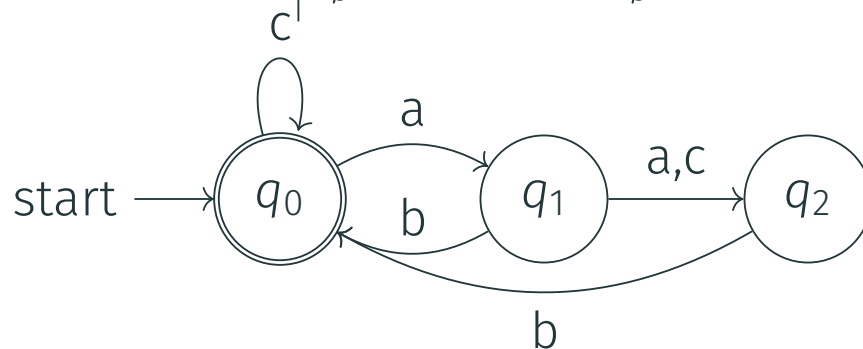
R	$a^{-1}R$	$b^{-1}R$	$c^{-1}R$
$q_0 = \varepsilon^{-1}R = (ab + c)^*$	$b(ab + c)^*$	\emptyset	$(ab + c)^*$
$q_1 = b(ab + c)^*$	\emptyset	$(ab + c)^*$	\emptyset
$q_2 = \emptyset$	\emptyset	\emptyset	\emptyset

Brzowski Method

Brings us to the **Brzowski 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.

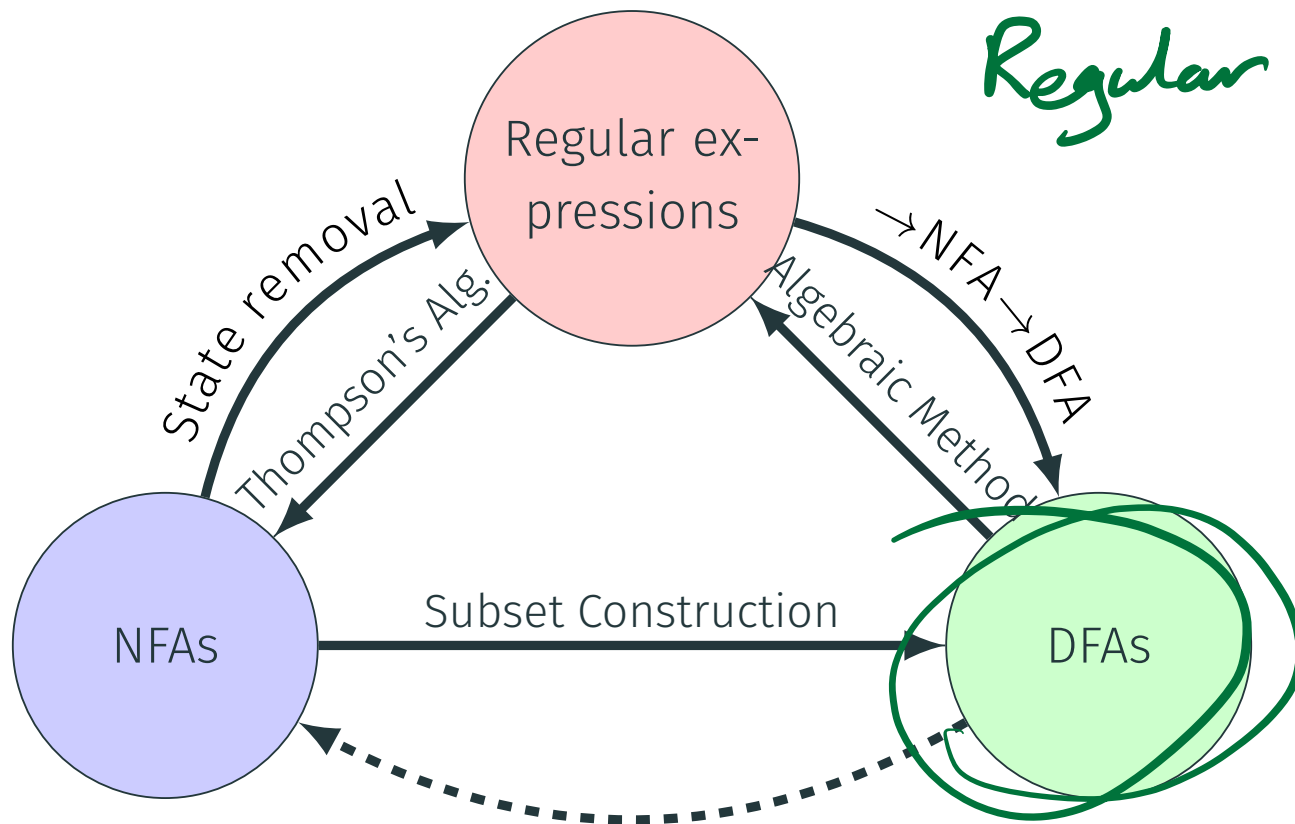
Consider $= \{w \in \Sigma^* | w \text{ has a substring } 010 \text{ or } 010\}$

- Is possible using Brzozowski² algorithm. Not needed for this course.
- Easier to just convert RegEx \rightarrow NFA \rightarrow DFA.

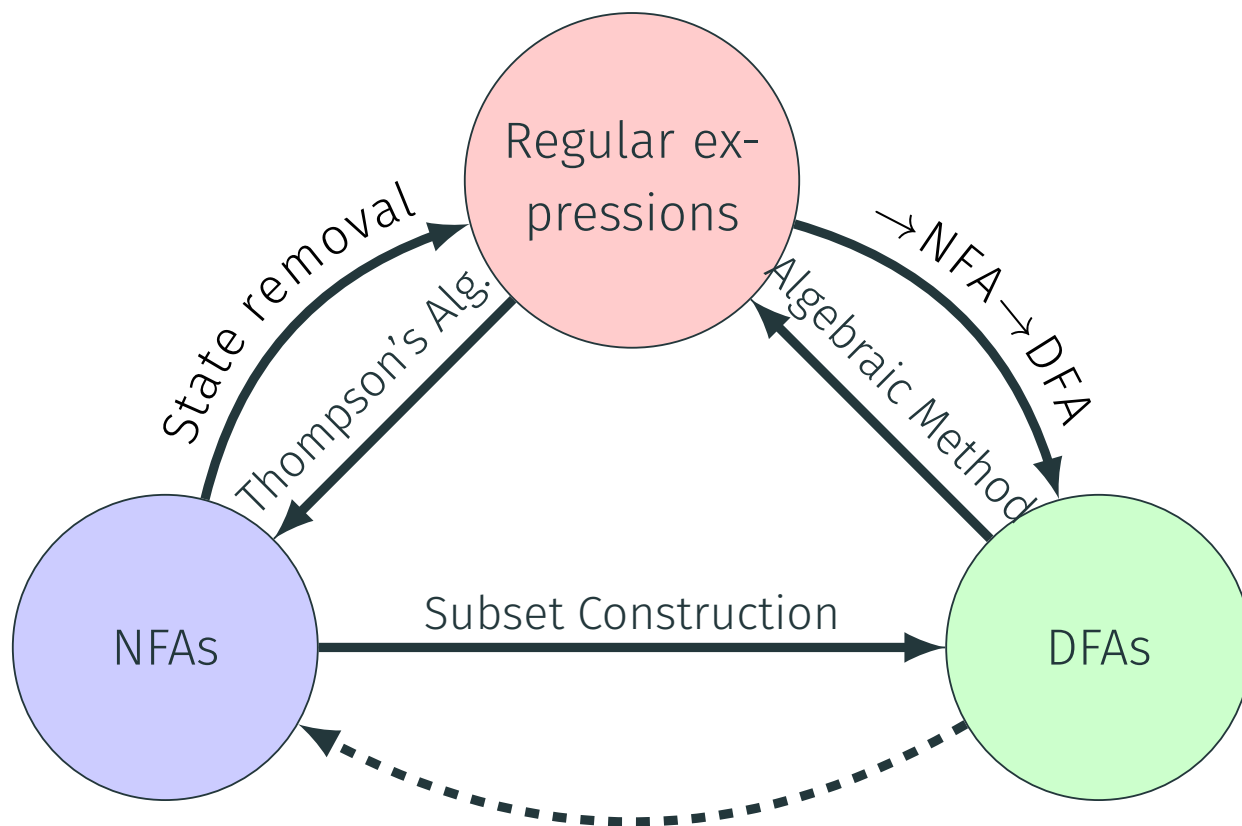
Conclusion

Proving equivalence

Regular languages



Proving equivalence



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