Pre-lecture brain teaser

Find the regular expressions for the following languages:

- 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.

CS/ECE-374: Lecture 5 - RegExp-DFA-NFA Equivalence

Lecturer: Nickvash Kani
Chat moderator: Samir Khan

February 09, 2021

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Find the regular expressions for the following languages:

- All strings that end in 1011

  $(0+1)^* 1011$

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

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

- All strings that do not contain 111 as a substring. ← can't be more than 3 ones in a row

  000 1 0 0000 11 0 11 000 0

  $((\varepsilon + 1 + 11) 0^*)^*$

  $0^*((\varepsilon + 1 + 11) 0^*)^*$

  $(\varepsilon + 1 + 11)(0^*(1 + 11))^*$

Regular Languages, DFAs, NFAs

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

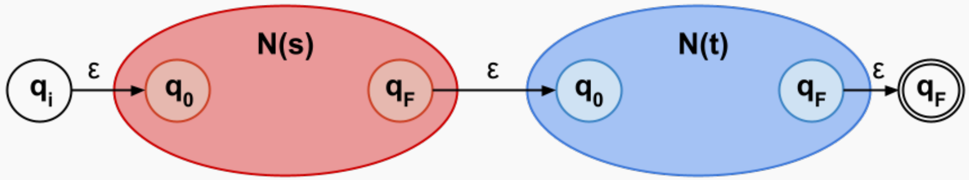
Regular Languages, DFAs, NFAs

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)

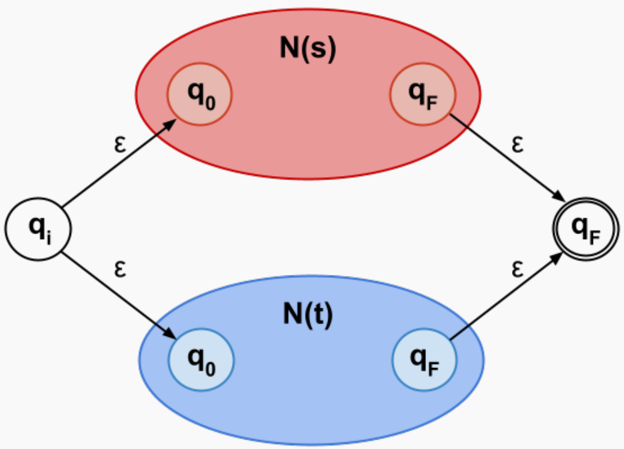
Thompson's algorithm

Given two NFAs s and t:

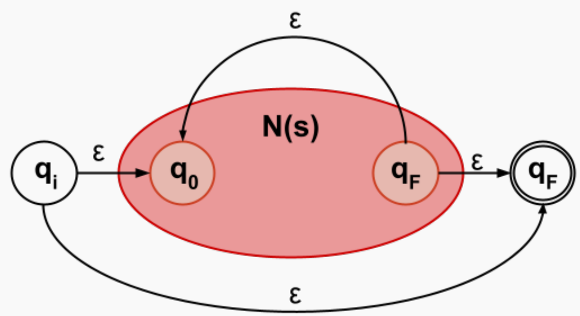


$L = L_s \cdot L_t$

$L = L_s \cup L_t$



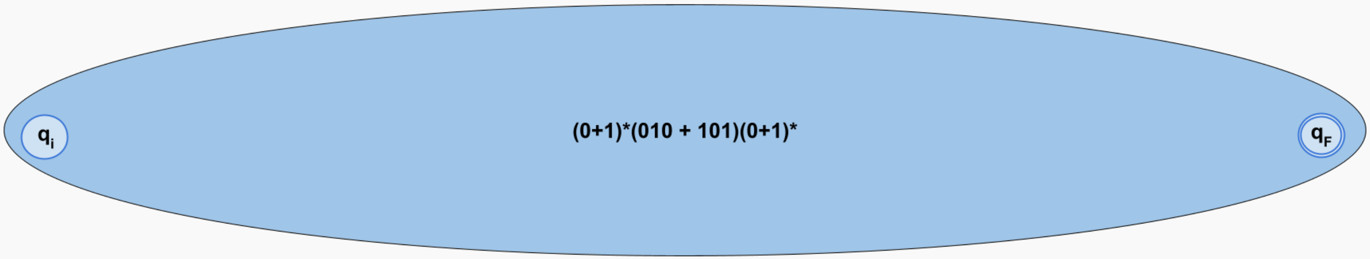
$L = (L_s)^*$



Regular expression to DFA example

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

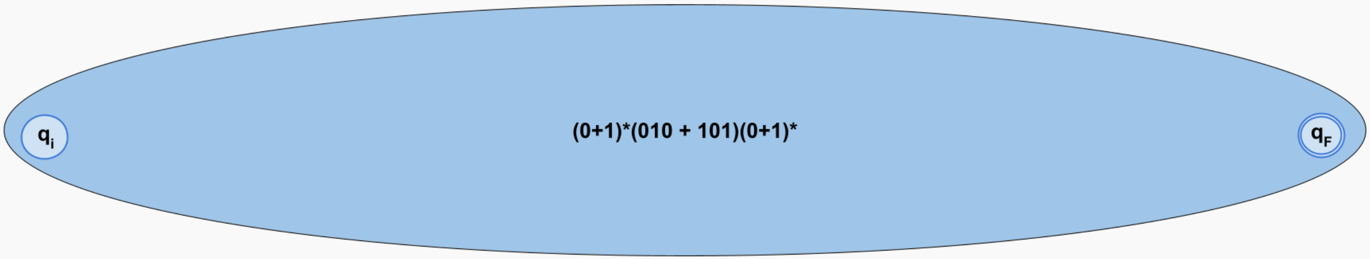
Example: $(0+1)^*(101+010)(0+1)^*$



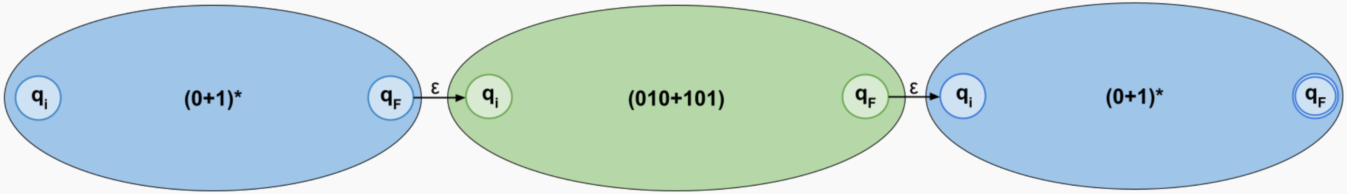
Regular expression to DFA example

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

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

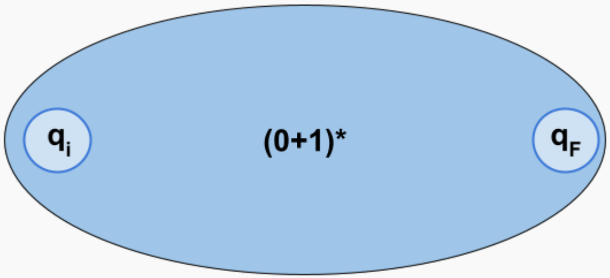


Using the concatenation rule:



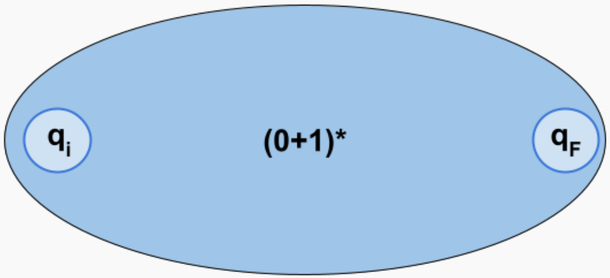
Regular expression to NFA example

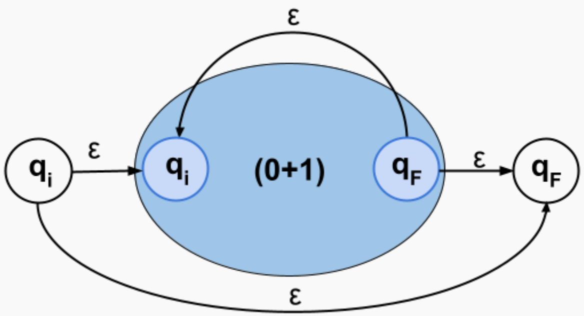
Find NFA for $(0+1)^*$



Regular expression to DFA example

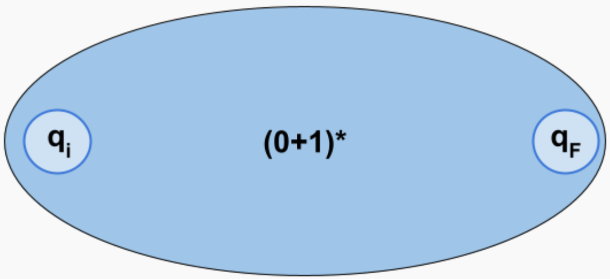
Find DFA for $(0 + 1)^*$

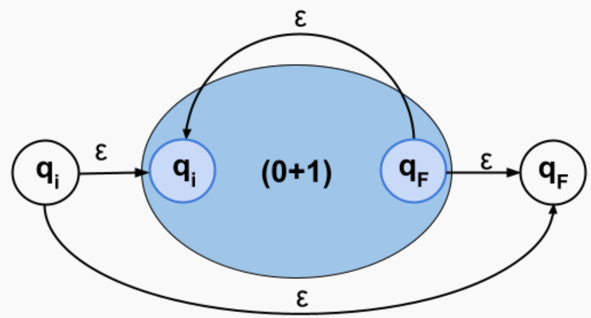


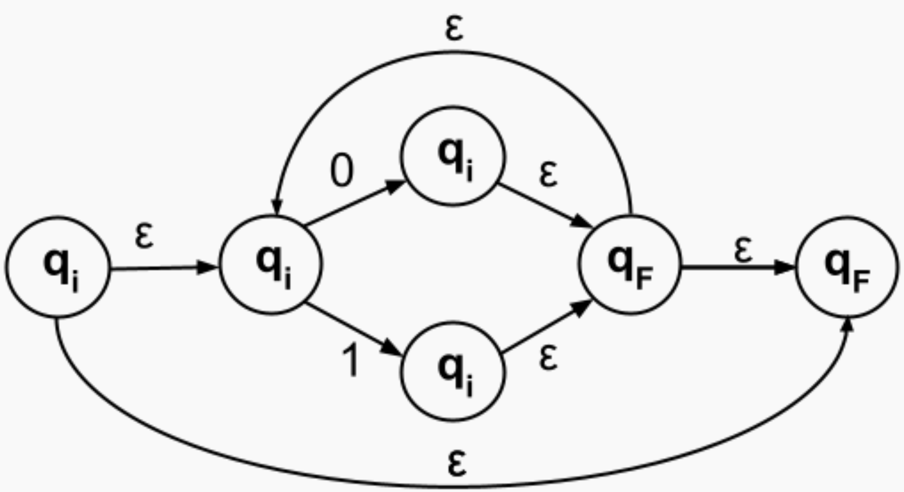


Regular expression to DFA example

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

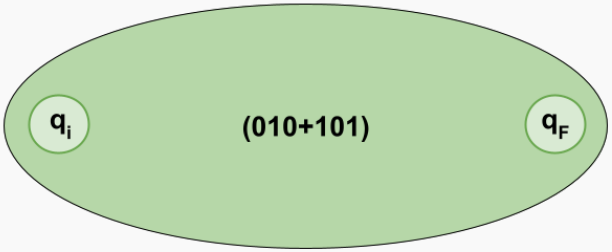






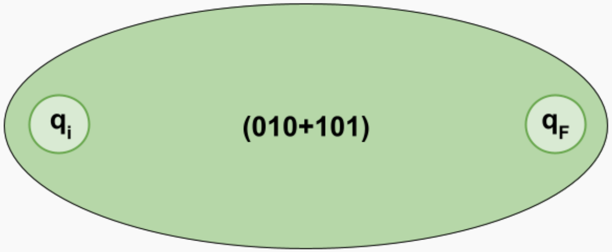
Regular expression to DFA example

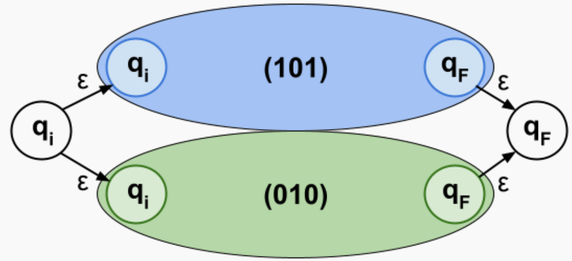
Find DFA for $(101 + 010)$



Regular expression to DFA example

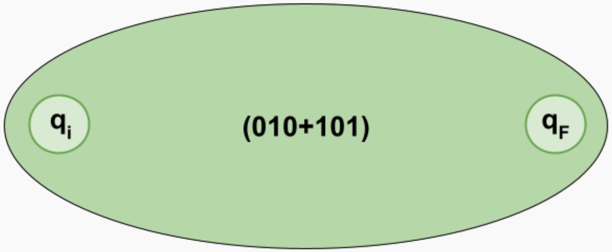
Find DFA for ($101 + 010$)

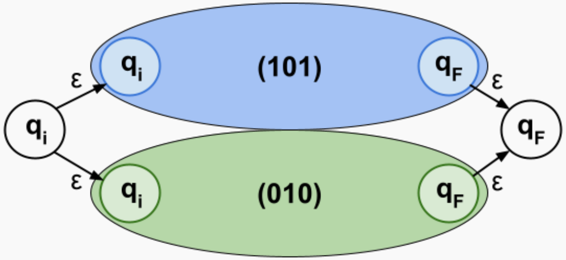


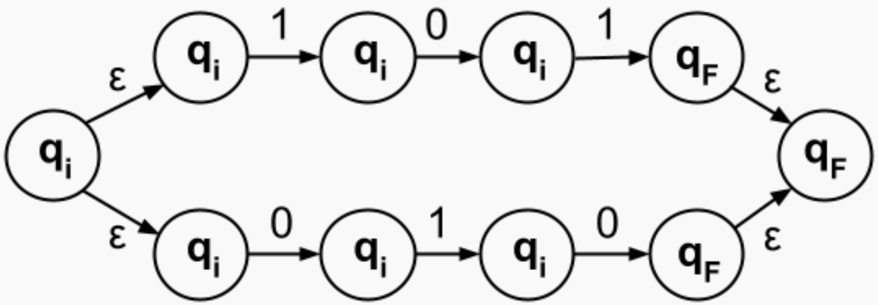


Regular expression to DFA example

Find DFA for $(101 + 010)$



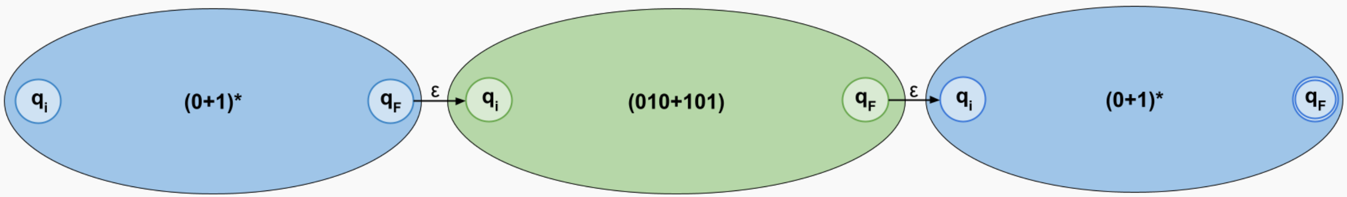




Regular expression to DFA example

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

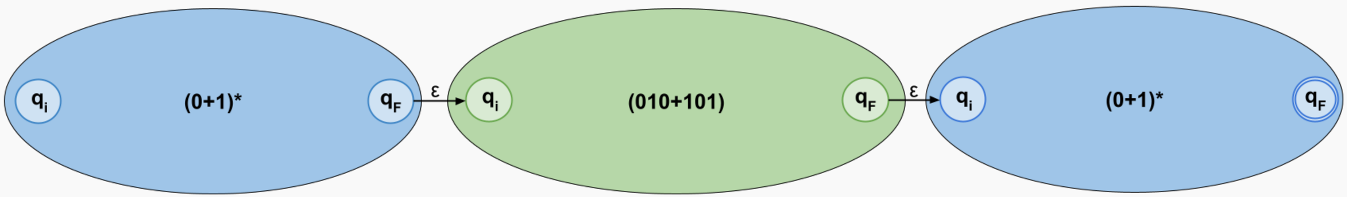
Example: $(0+1)^*(101+010)(0+1)^*$



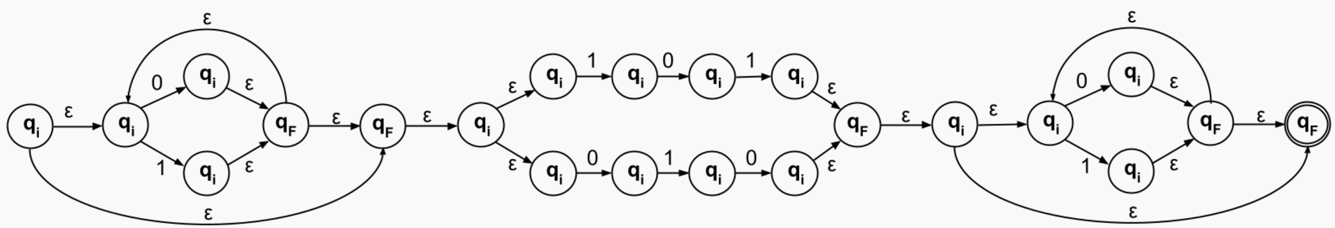
Regular expression to DFA example

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

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



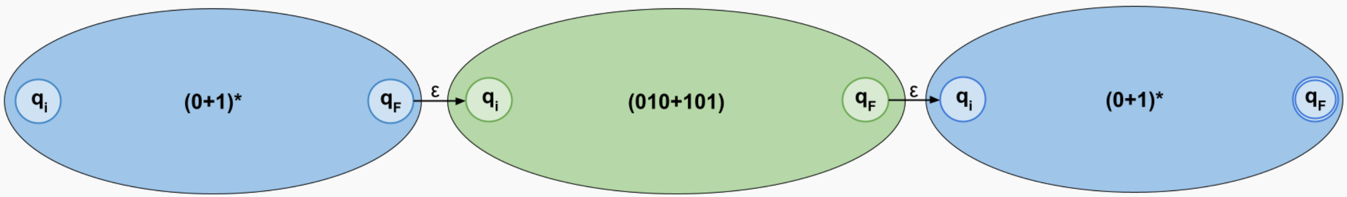
Using the concatenation rule:



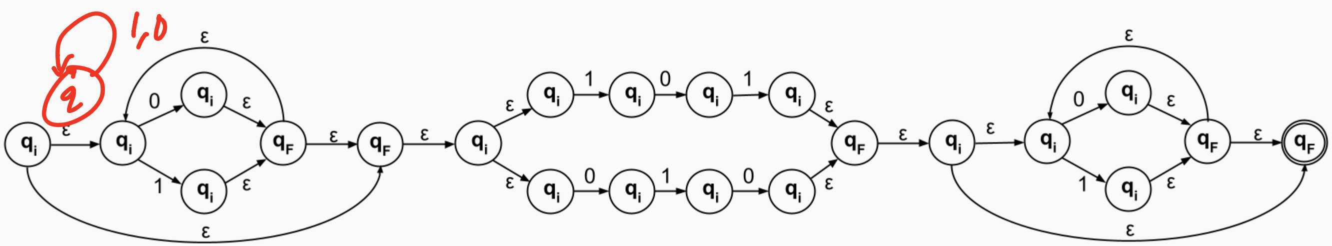
Regular expression to DFA example

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

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



Using the concatenation rule:

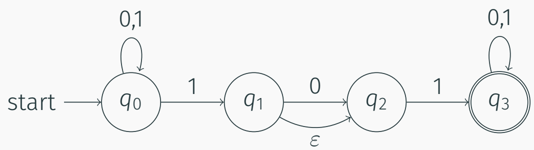


What does Thompson's algorithm mean?!

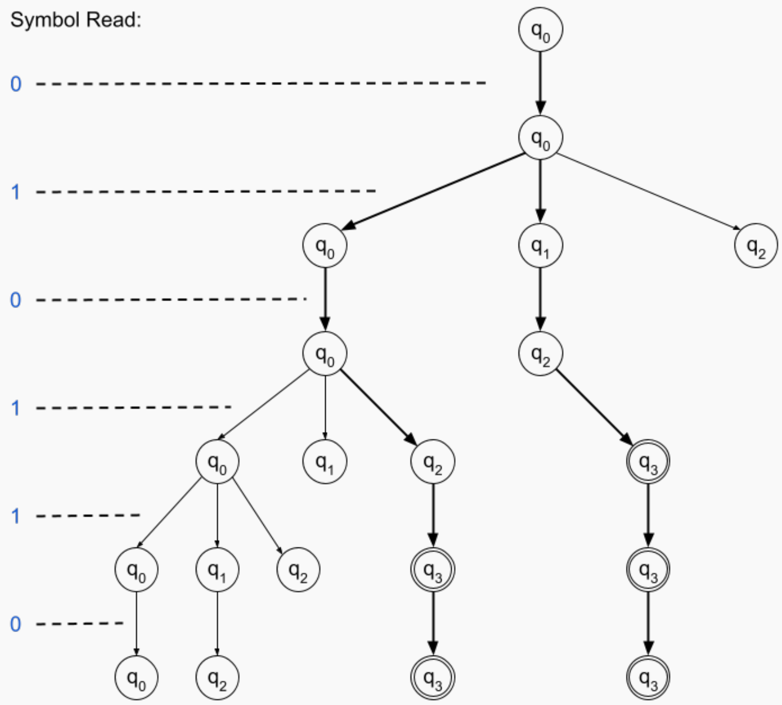
Every regular expression has a equivalent NFA

Equivalence of NFAs and DFAs

Another Way to look at NFAs

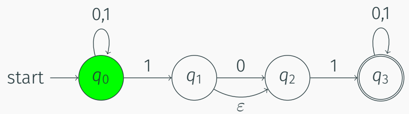


Is 010110 accepted?

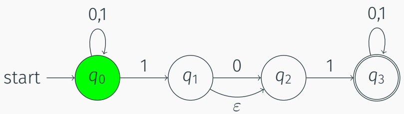


Another Way to look at NFAs

Is 010110 accepted?

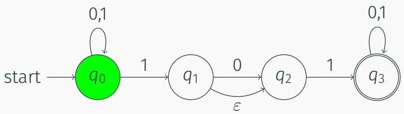


Another Way to look at NFAs

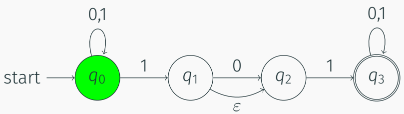


Is 010110 accepted?

0

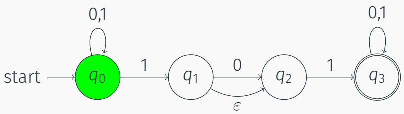


Another Way to look at NFAs

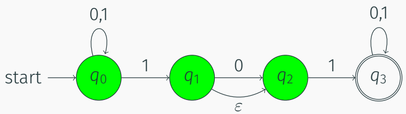


Is 010110 accepted?

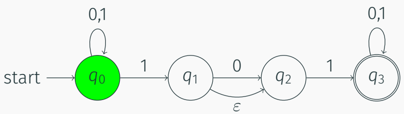
0



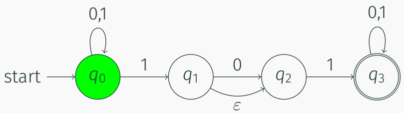
1

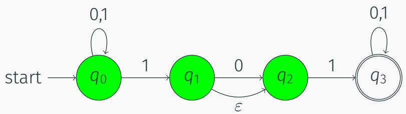


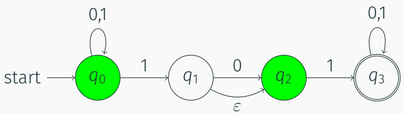
Another Way to look at NFAs



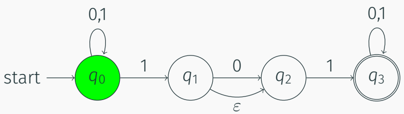
Is 010110 accepted?

0


1


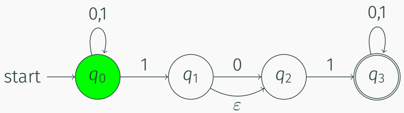
0


Another Way to look at NFAs

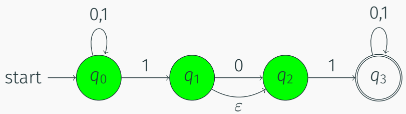


Is 010110 accepted?

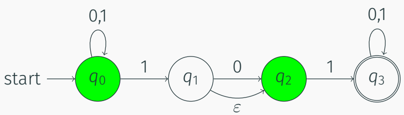
0



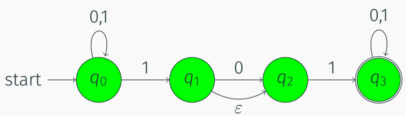
1



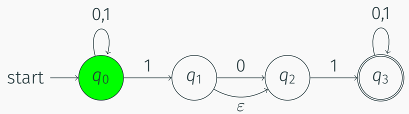
0



1

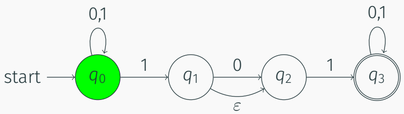


Another Way to look at NFAs

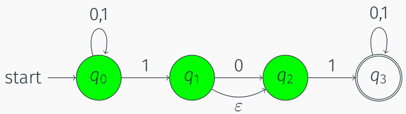


Is 010110 accepted?

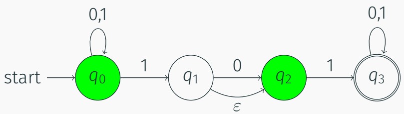
0



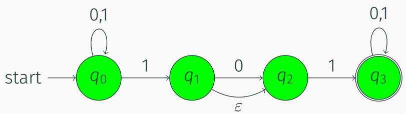
1



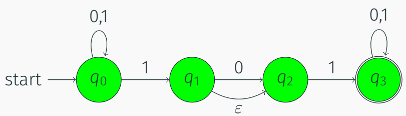
0



1

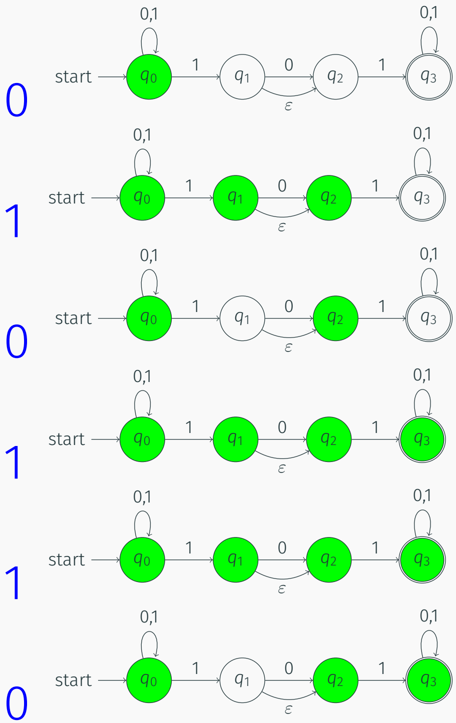


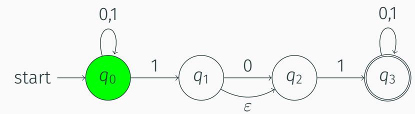
1

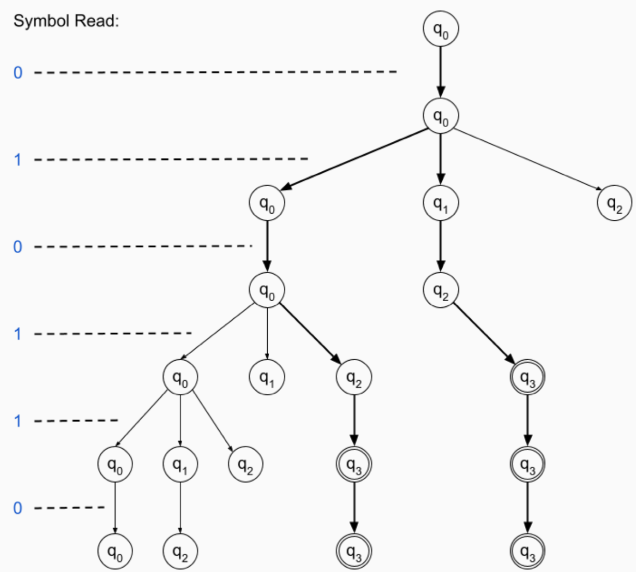


Another Way to look at NFAs

Is 010110 accepted?







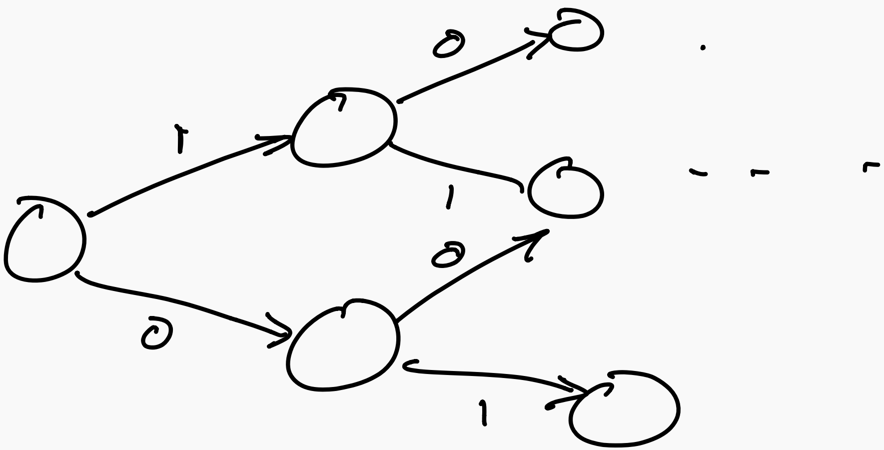
The idea of the conversion of NFA to DFA

Equivalence of NFAs and DFAs

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

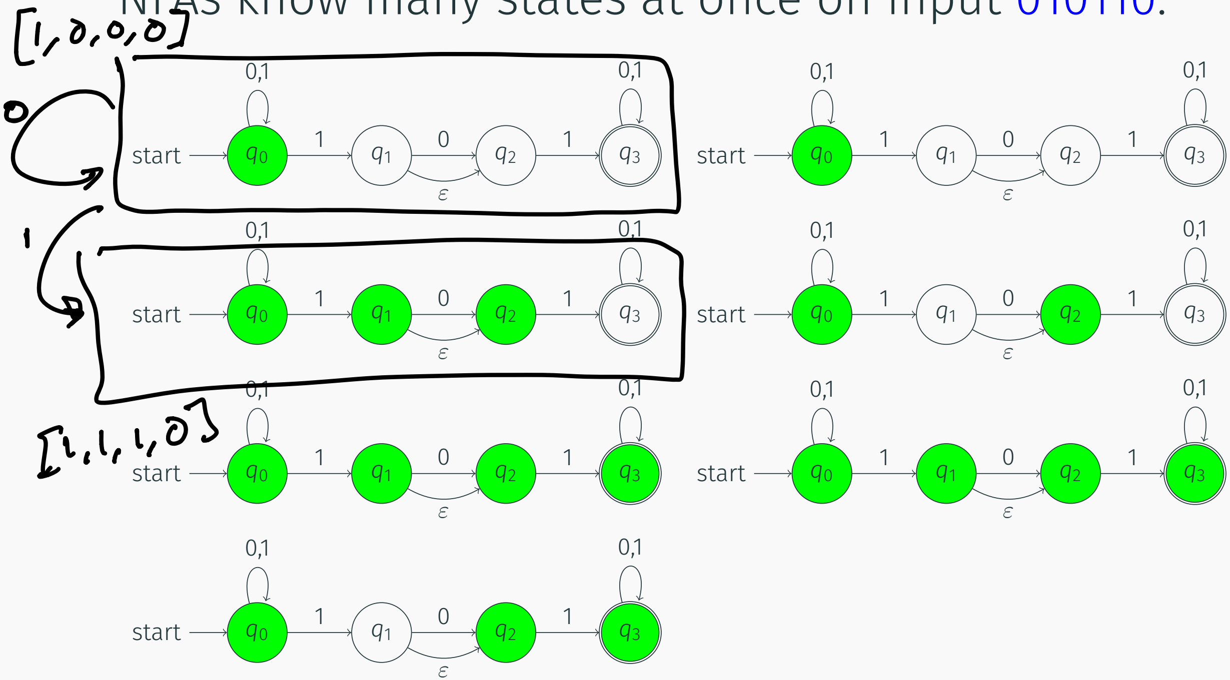
DFAs are memoryless...

- 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.



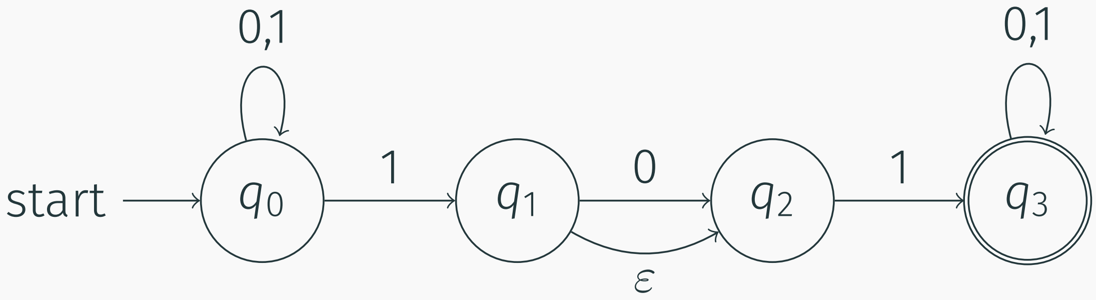
Simulating NFA

NFAs know many states at once on input 010110.



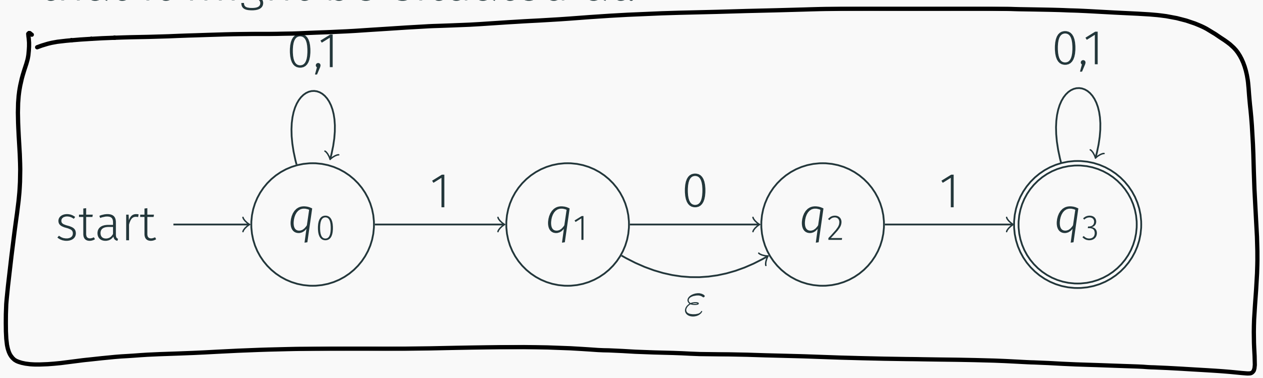
The state of the NFA

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

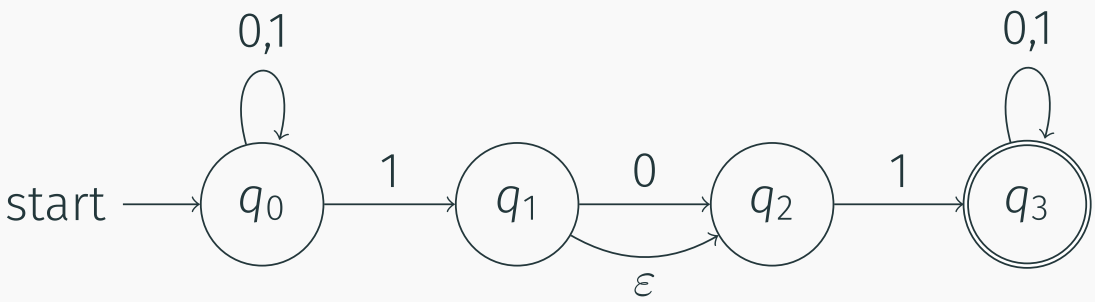
It is easy to state that the state of the automata is the states that it might be situated at.



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.



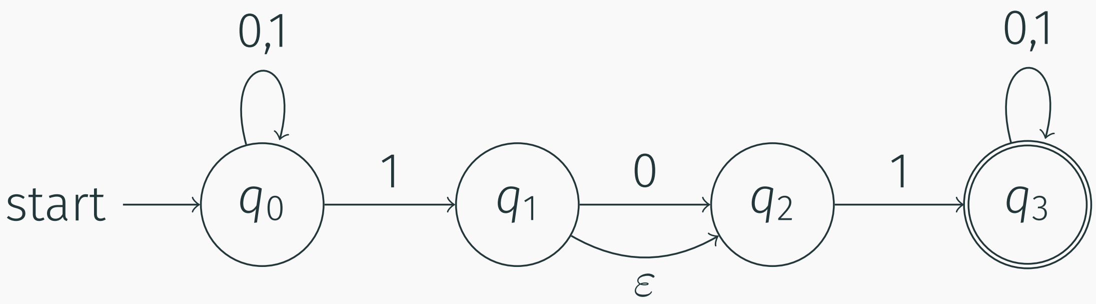
configuration: A set of states the automata might be in.

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

$\delta(q, a) = \in P(q)$

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.

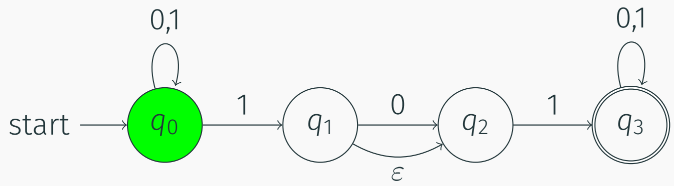


configuration: A set of states the automata might be in.

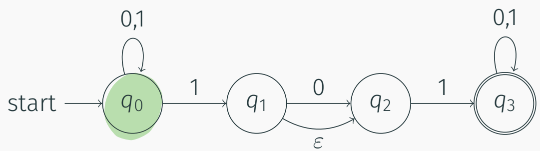
Possible configurations: $\mathcal{P}(q) = \emptyset$, $\{q_0\}$, $\{q_0, q_1\}$...

Big idea: Build a DFA on the configurations.

Example

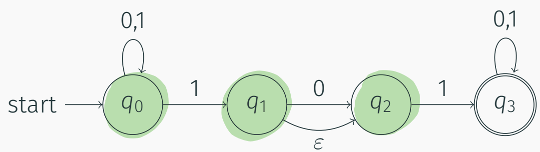


If receives 0 :

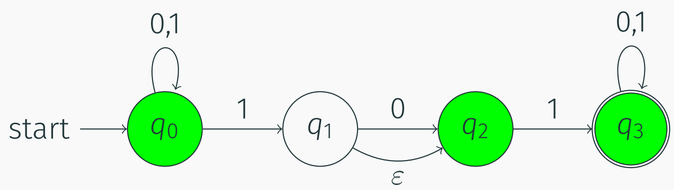


[1, 0, 0, 0]

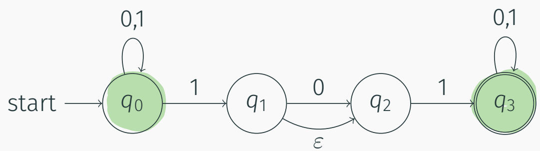
If receives 1 :



[1, 1, 1, 0]

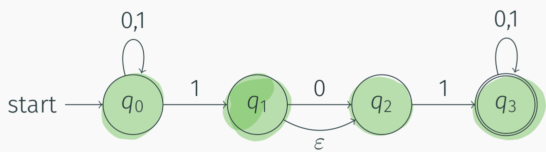


If receives 0 :



[1, 0, 0, 1]

If receives 1 :



[1, 1, 1, 1]

Simulating an NFA by a DFA

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?

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?

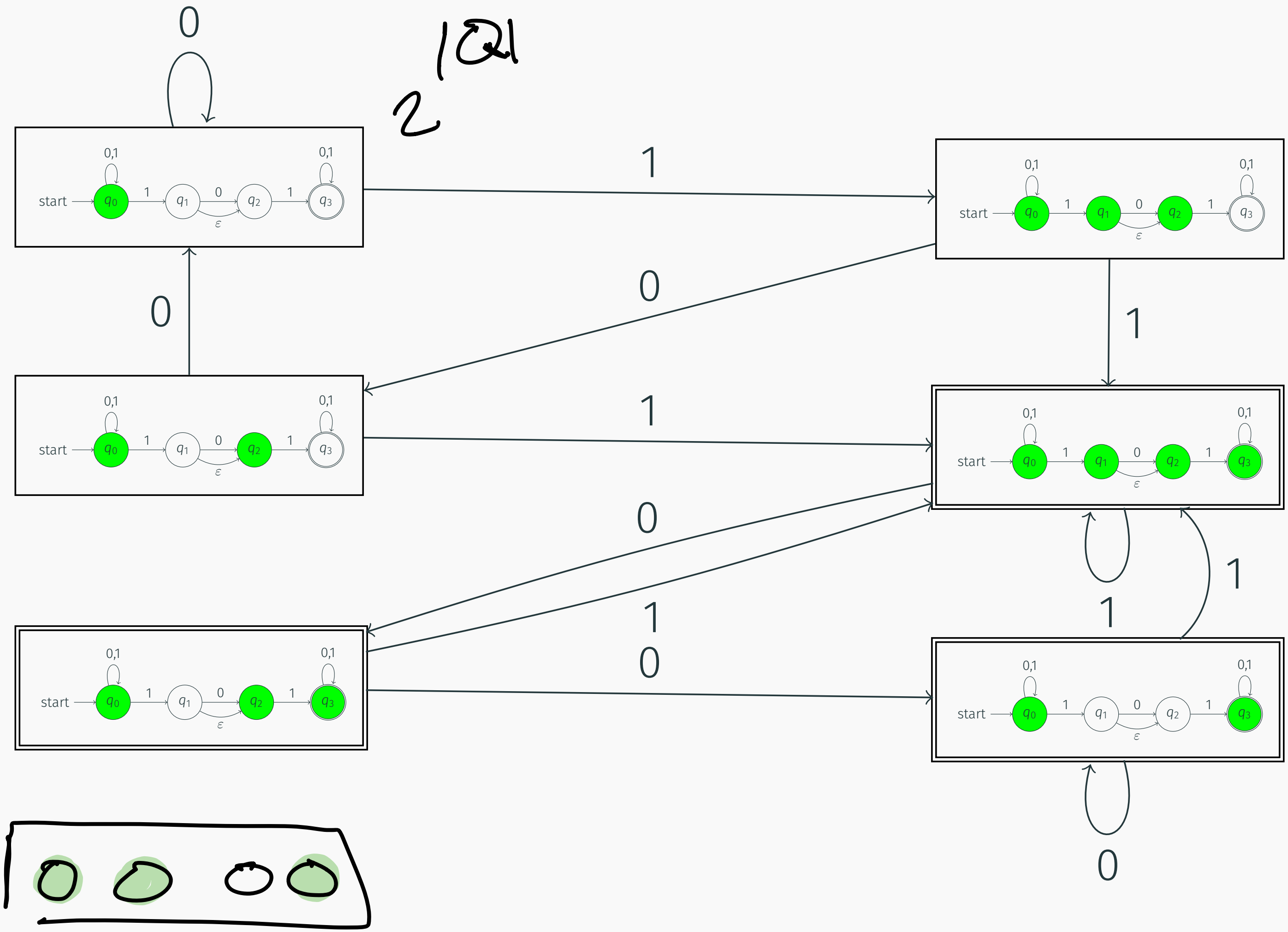
Simulating an NFA by a DFA

- 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.

Algorithm for converting NFA to DFA

Recall I

Extending the transition function to strings

Definition
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ-transitions.

Definition
Inductive definition of $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

- if $w = \varepsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$:
  $\delta^*(q, a) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a) \right)$
- if $w = ax$:
  $\delta^*(q, w) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \bigcup_{r \in \delta^*(p,a)} \delta^*(r, x) \right)$

Recall II

Formal definition of language accepted by N

Definition
A string w is accepted by NFA N if $\delta_N^*(s, w) \cap A \neq \emptyset$.

Definition
The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$

Subset Construction

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

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

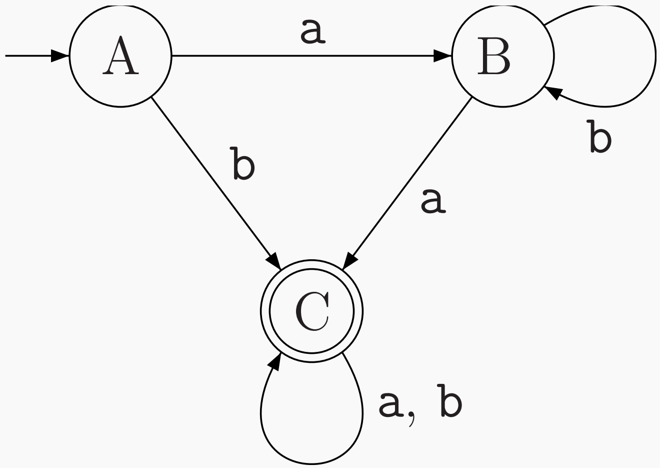
regular exp → NFA → DFA

DFA → regular exp

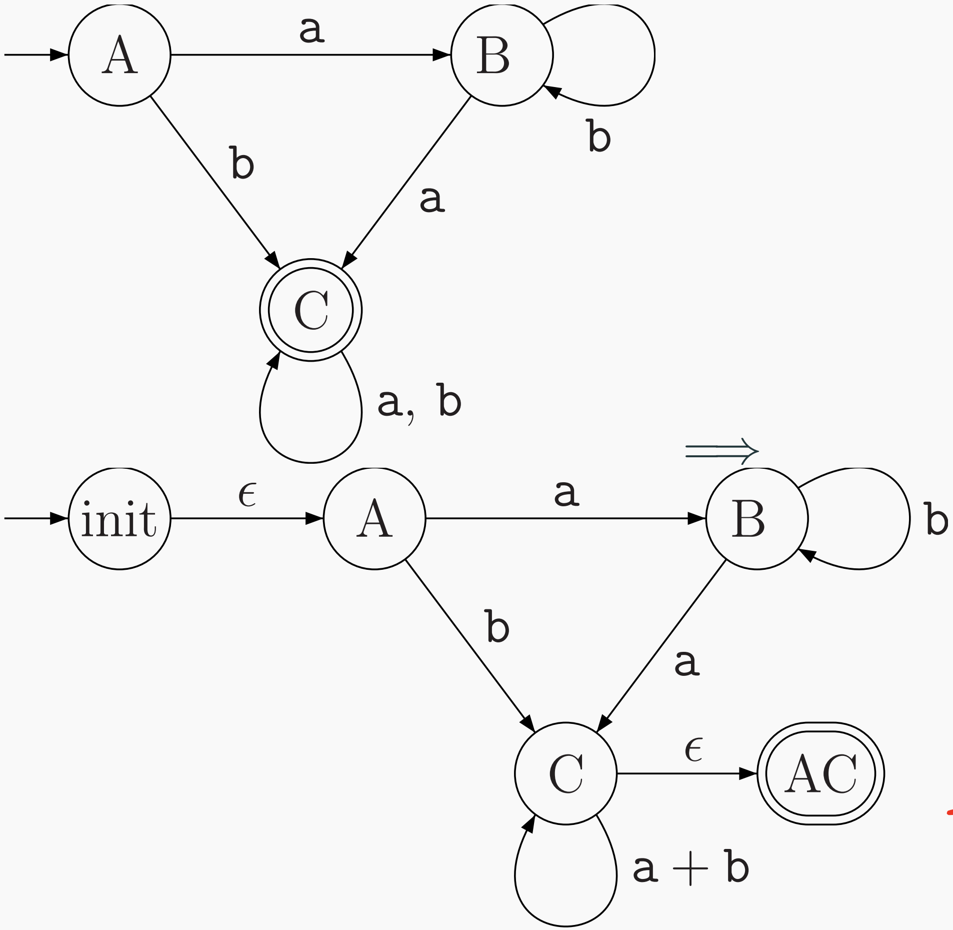
DFA → NFA ≡

Algorithm for converting NFA into regular expression

Stage 0: Input

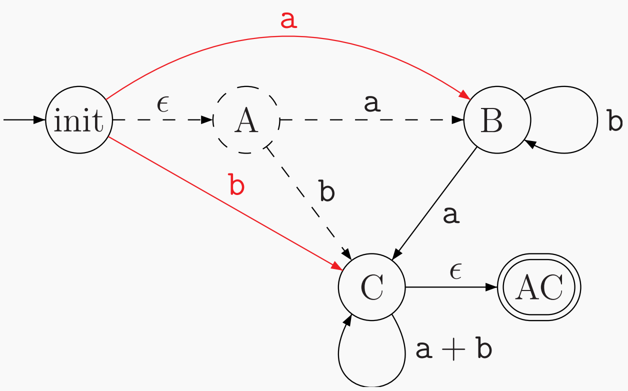


Stage 1: Normalizing



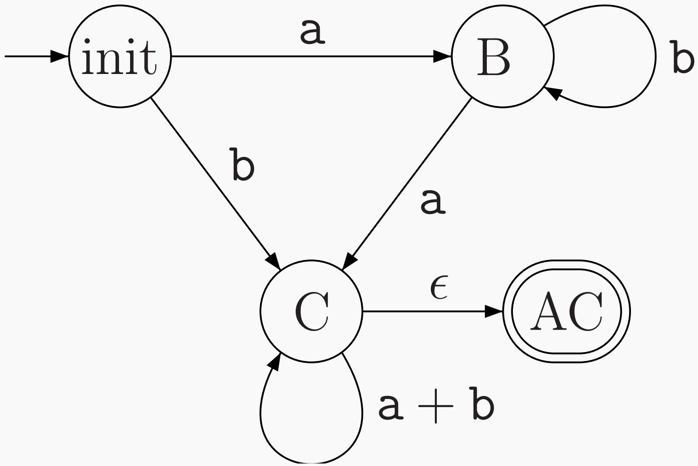
want regular expression

Stage 2: Remove state A



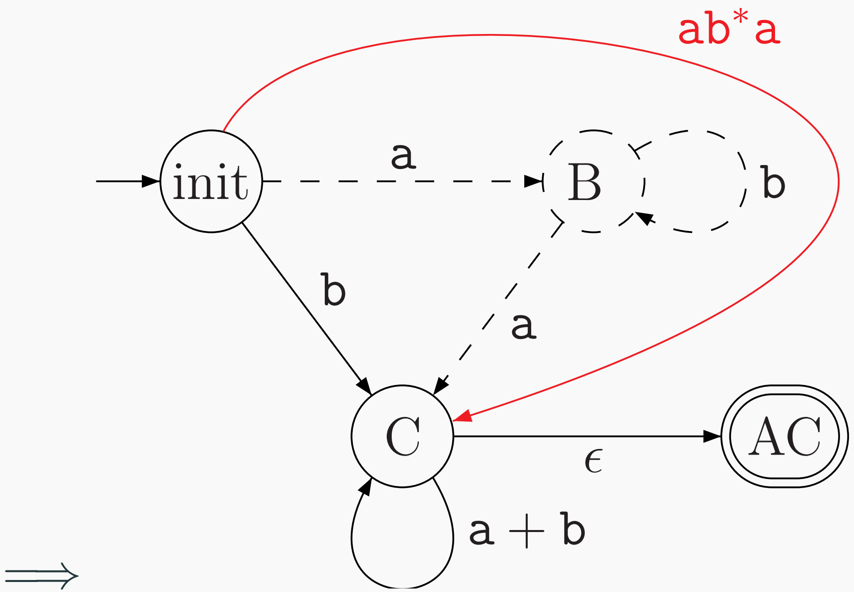
$\Longrightarrow$

Stage 4: Redrawn without old edges

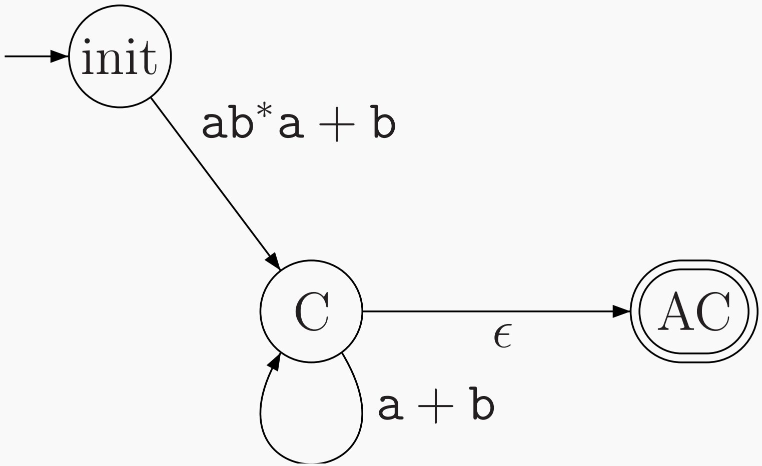


$\Longrightarrow$

Stage 4: Removing B

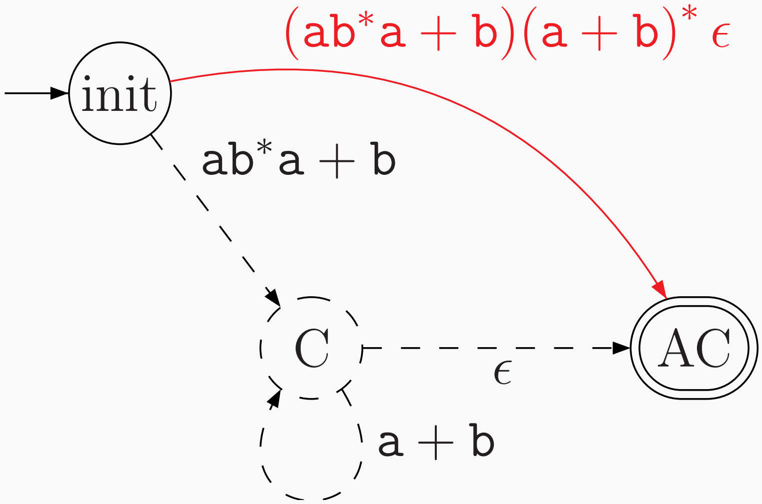


Stage 5: Redraw



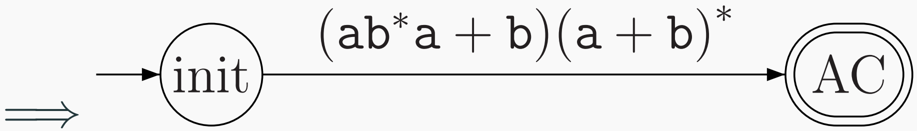
$\Longrightarrow$

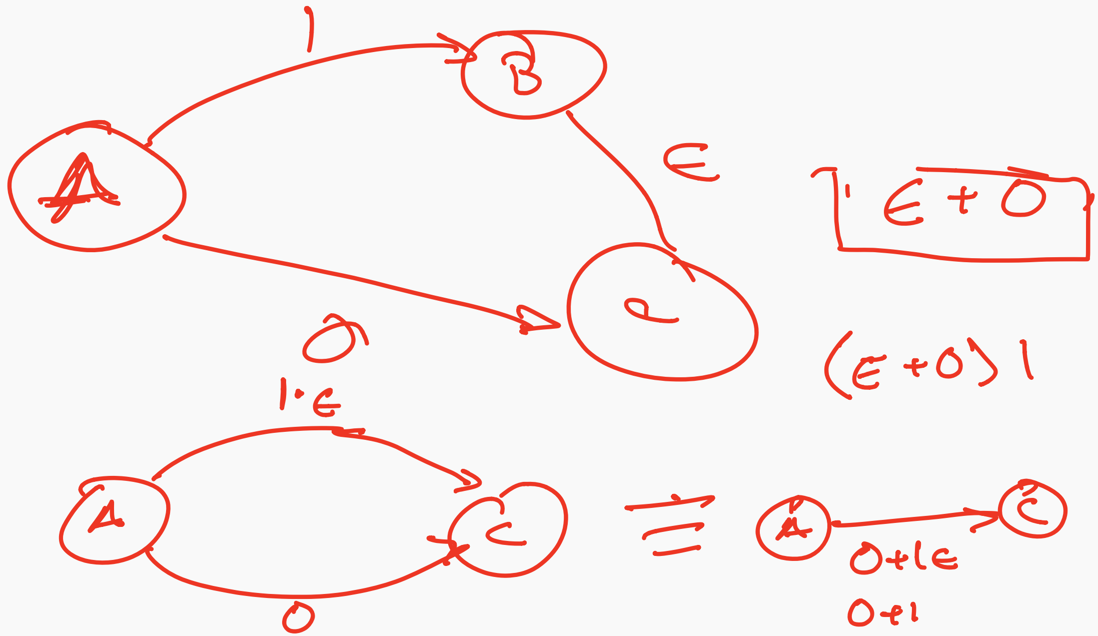
Stage 6: Removing C



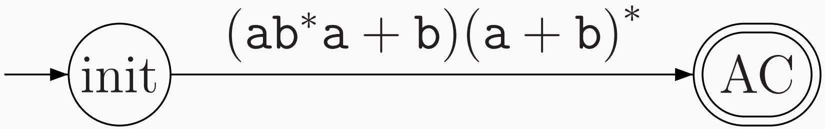
$\Longrightarrow$

Stage 7: Redraw





Stage 8: Extract regular expression



Thus, this automata is equivalent to the regular expression

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