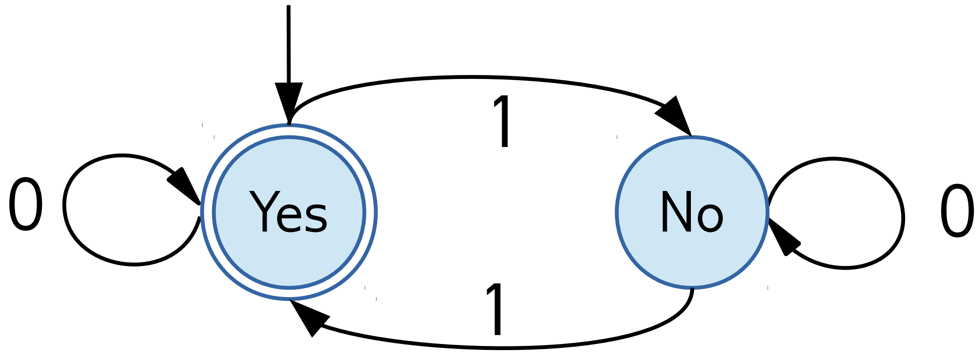
Nondeterministic Finite Automata and Regular Expressions

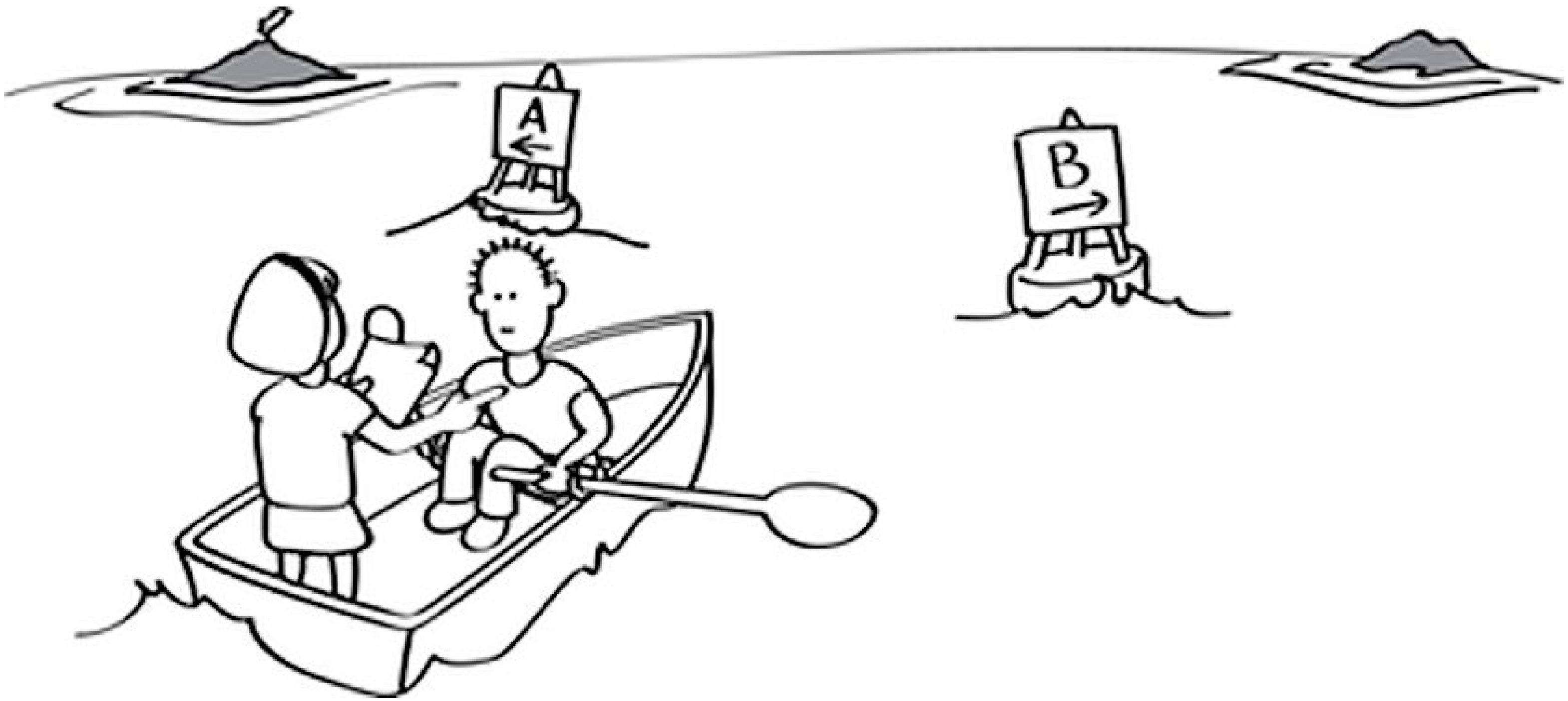
CS 2800: Discrete Structures, Spring 2015

Sid Chaudhuri

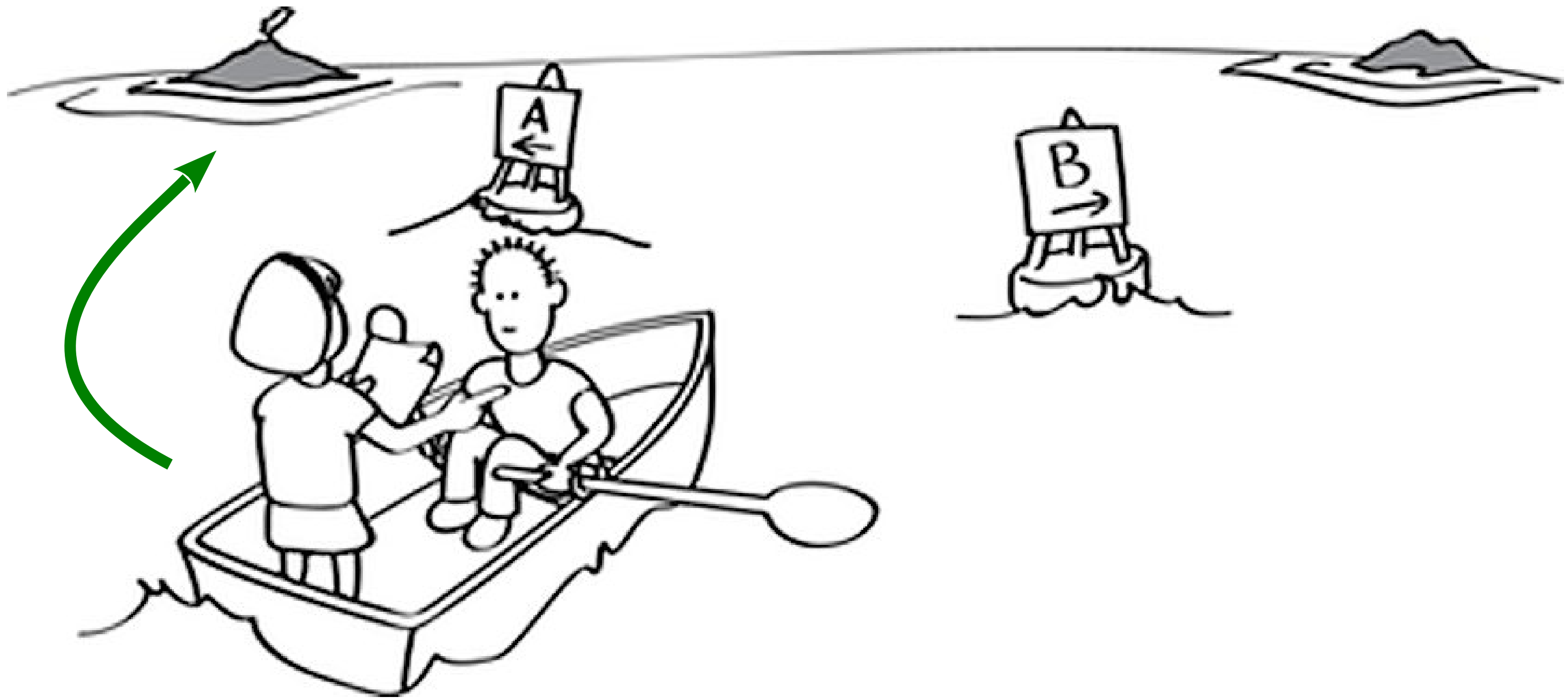
Recap: Deterministic Finite Automaton

- A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \to Q$
  - $q_0 \in Q$ is the start/initial state
  - $F \subseteq Q$ is the set of final/accepting states



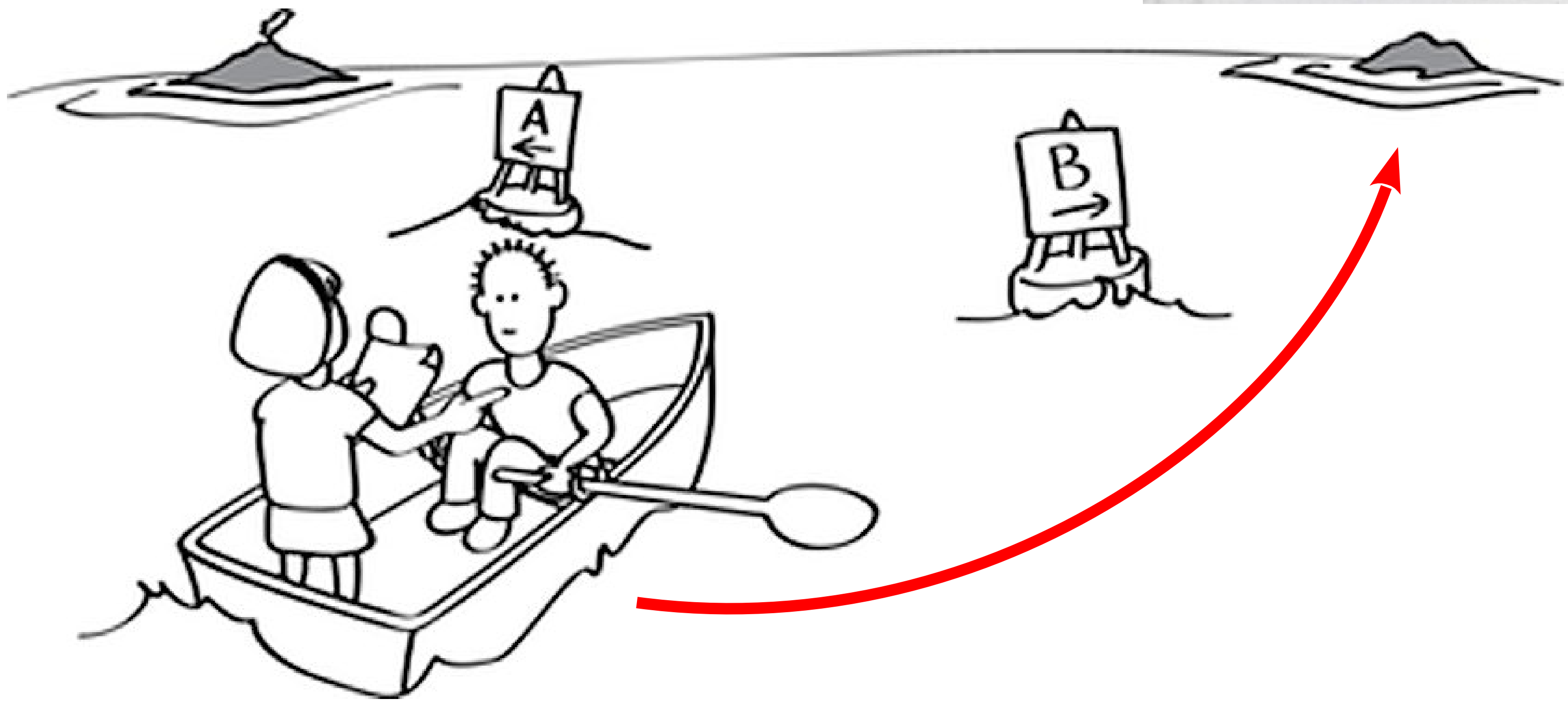

A
B

P = NP!!!



csunplugged.com, clubpenguin.wikia.com

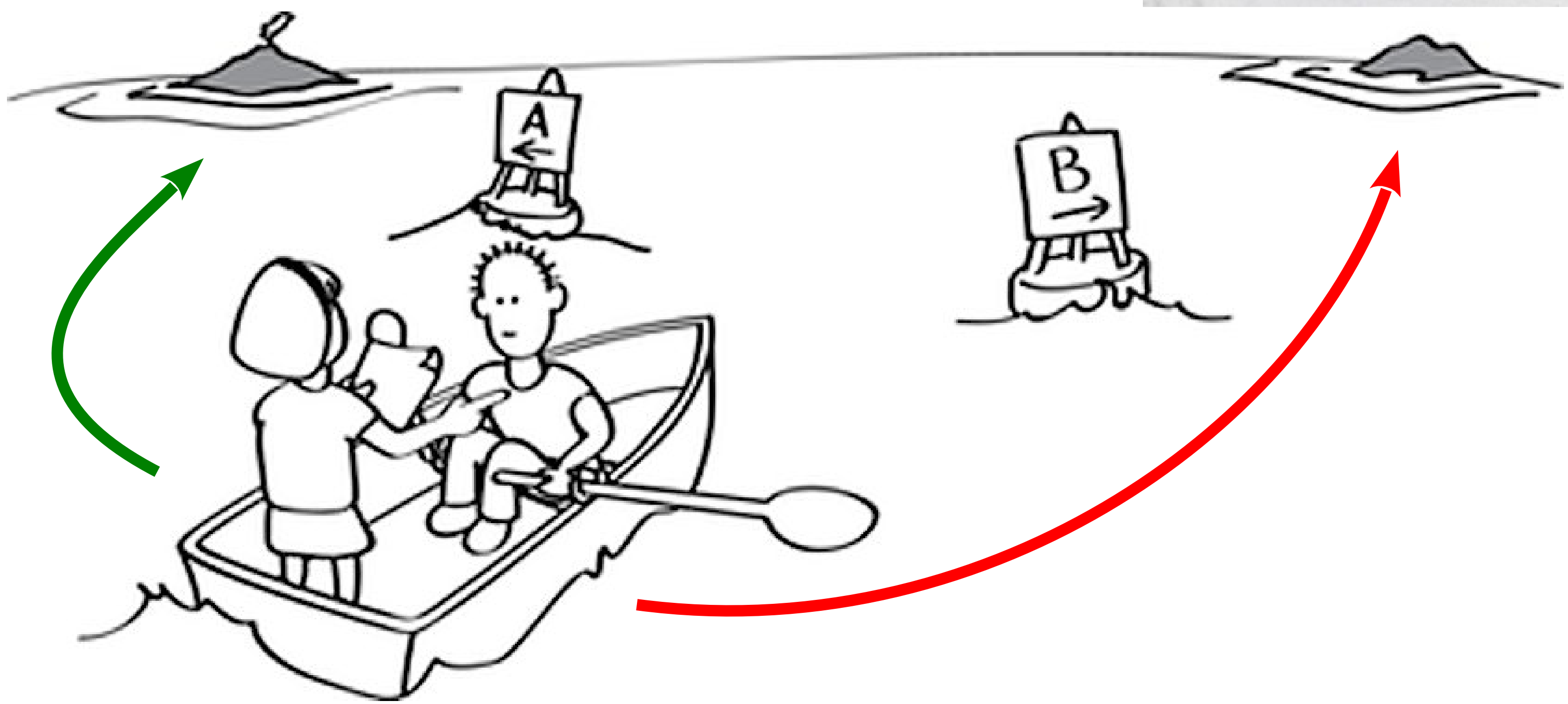
P = NP!!!



csunplugged.com, clubpenguin.wikia.com

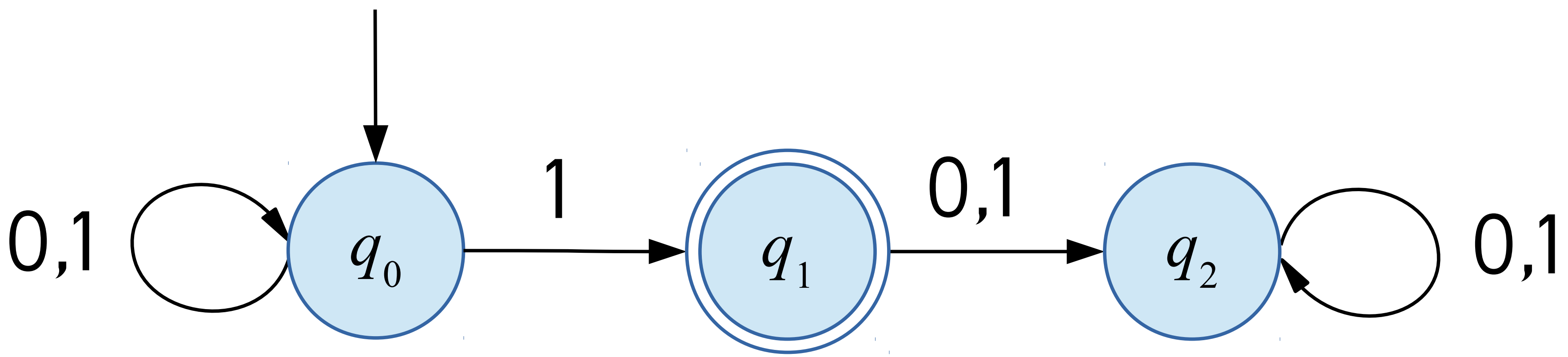
A NON-deterministic finite automaton lets you try all possible choices in parallel. If *ANY* choice leads you to the treasure, the pirate can't harm you!

P = NP!!!

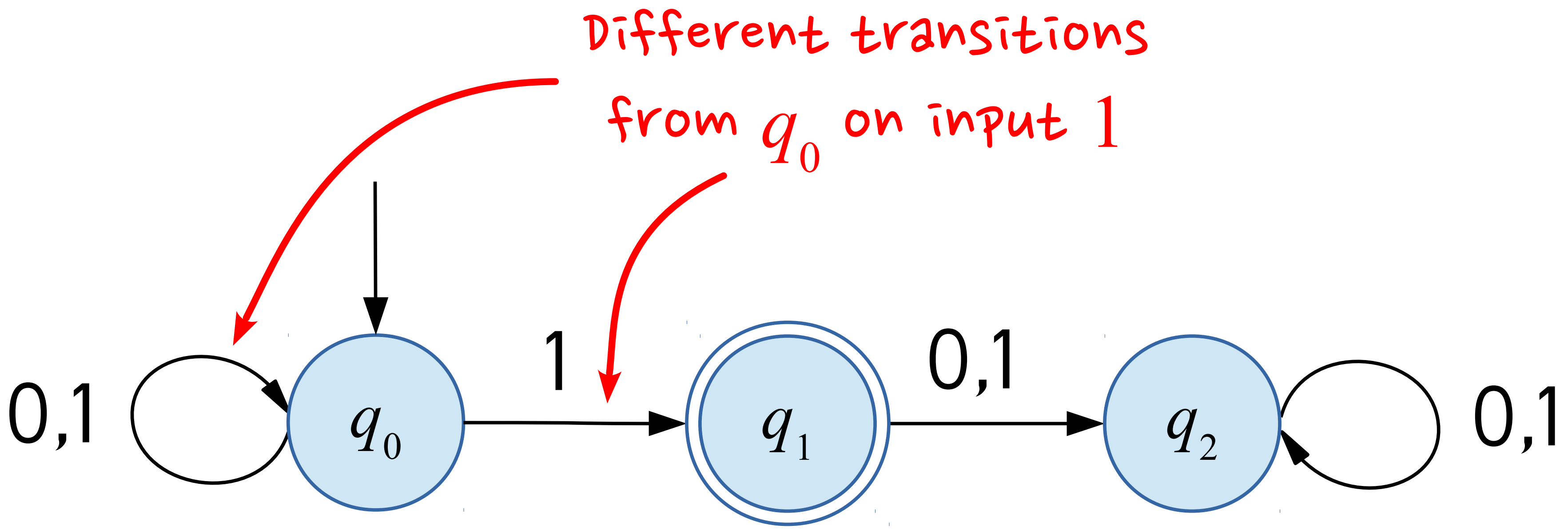


csunplugged.com, clubpenguin.wikia.com

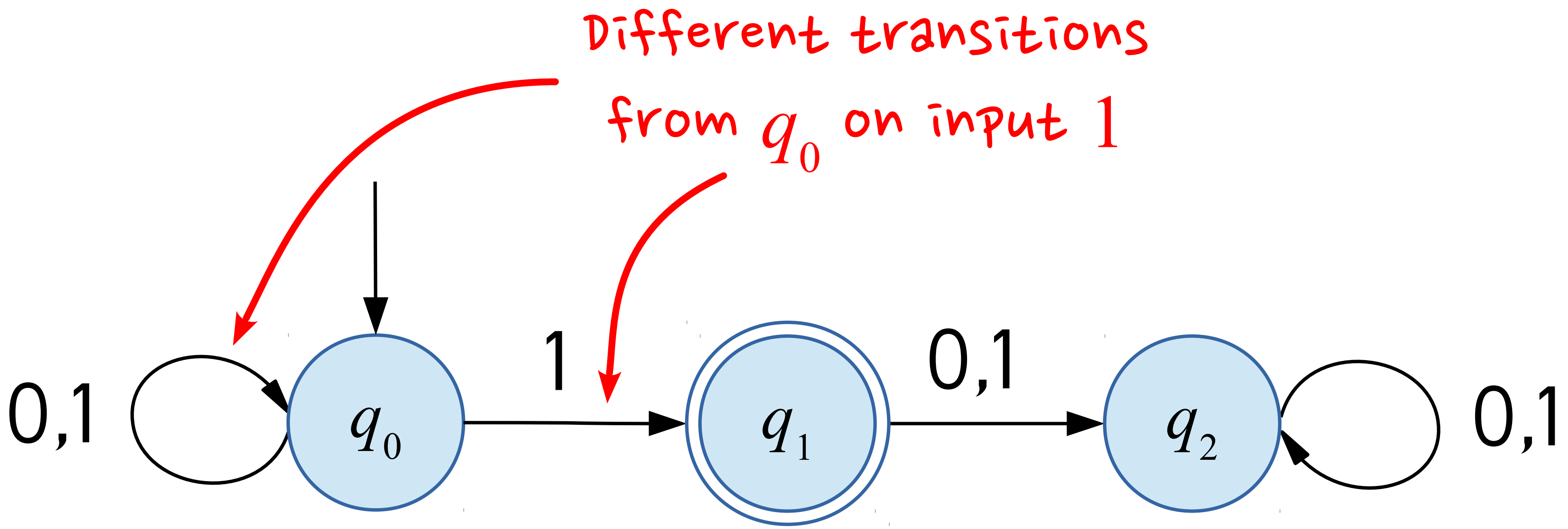
A ***non***-deterministic finite automaton



A ***non***-deterministic finite automaton

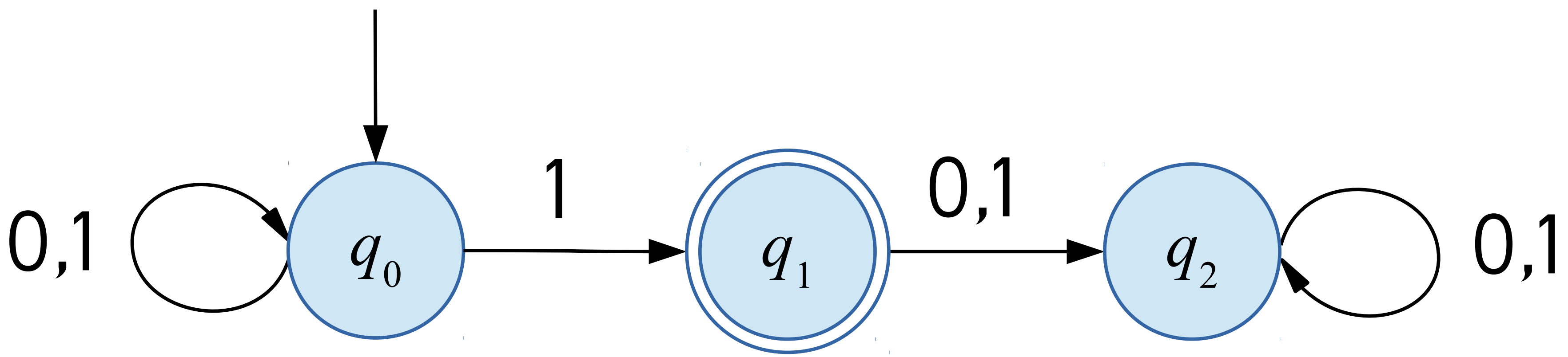


A ***non***-deterministic finite automaton



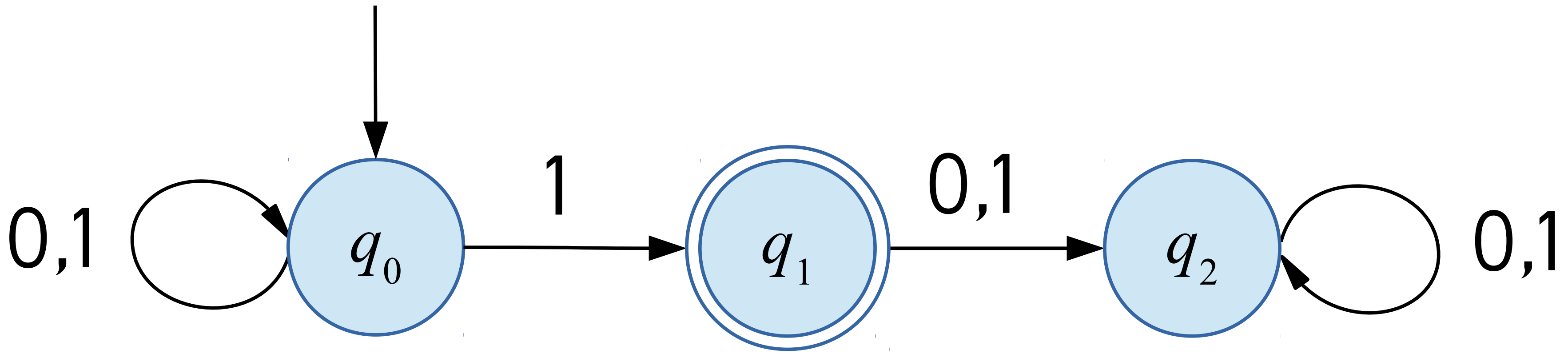
An NFA accepts a string x if it *can* get to an accepting state on input x

A ***non***-deterministic finite automaton



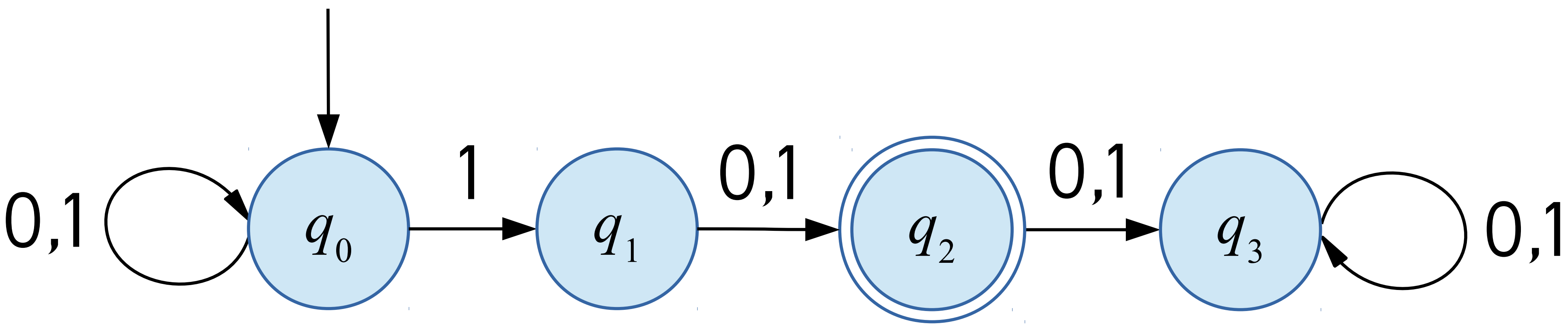
What language does this automaton accept?

A ***non***-deterministic finite automaton



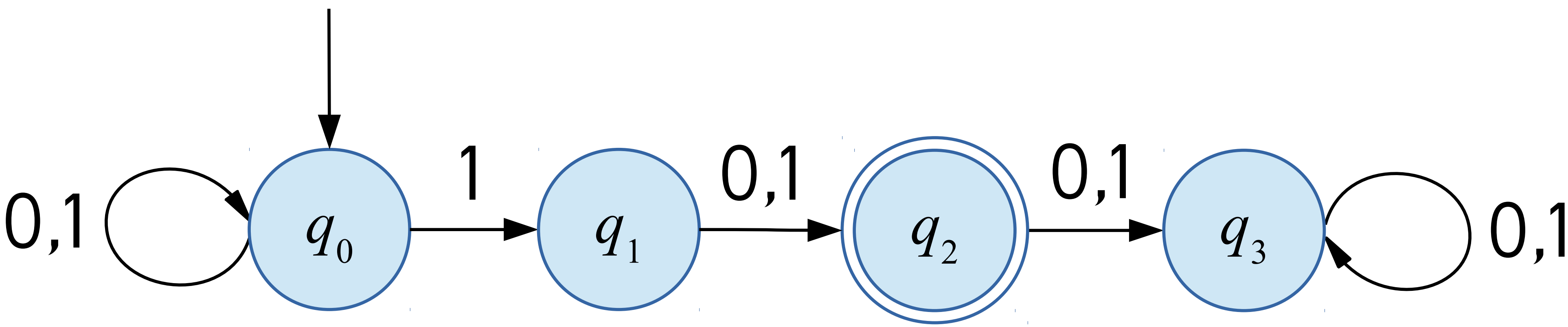
Answer: All strings ending with 1

Another NFA



What language does this automaton accept?

Another NFA



Answer: All strings with 1 in the penultimate place

Non-deterministic Finite Automaton

- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Non-deterministic Finite Automaton

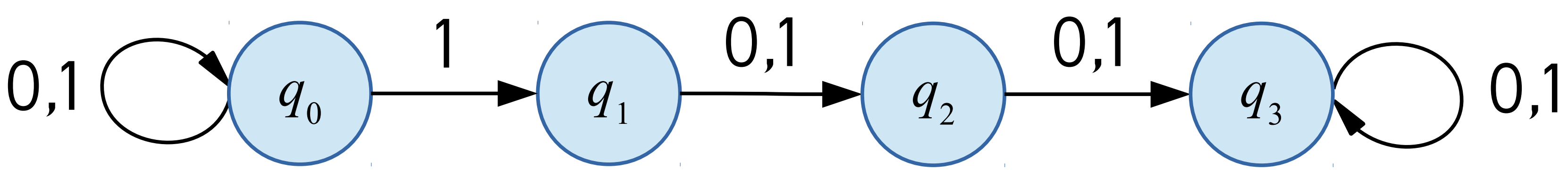
- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states

Non-deterministic Finite Automaton

- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)

Non-deterministic Finite Automaton

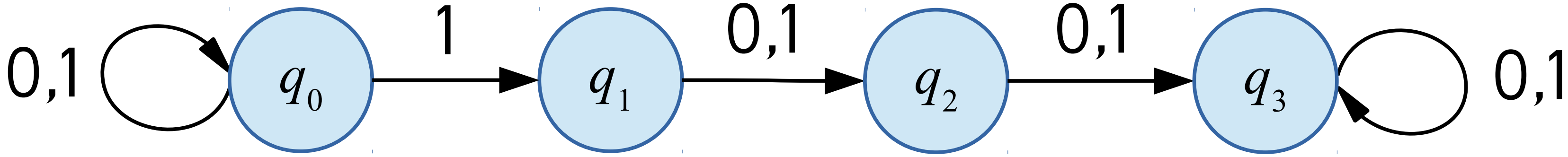
- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \to 2^Q$



Non-deterministic Finite Automaton

- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$

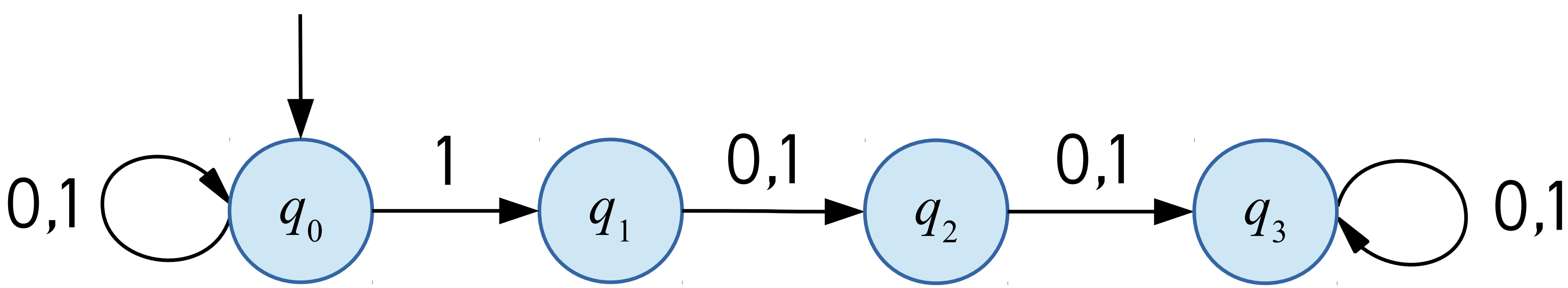
only change from DFAs



Non-deterministic Finite Automaton

- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$
  - $q_0 \in Q$ is the start/initial state

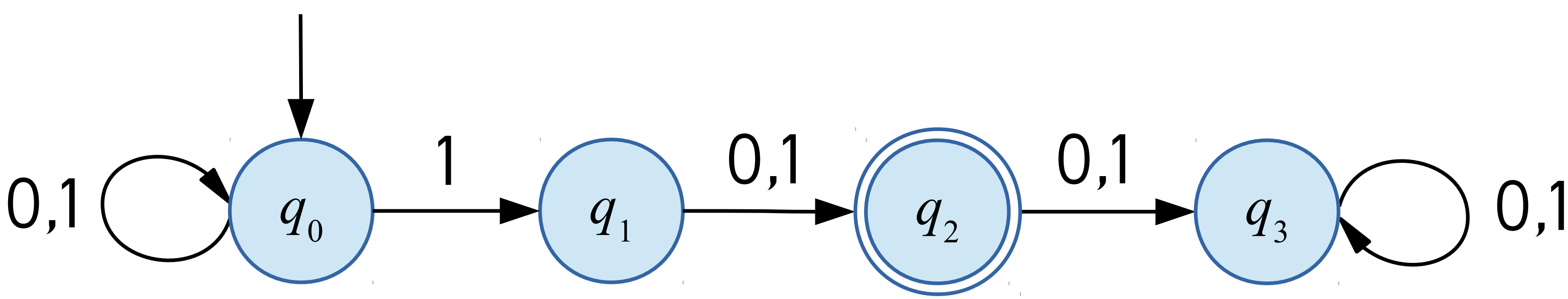
only change from DFAs



Non-deterministic Finite Automaton

- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \to 2^Q$
  - $q_0 \in Q$ is the start/initial state
  - $F \subseteq Q$ is the set of final/accepting states

only change from DFAs



Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x

Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x
  - Think of it as trying many options in parallel, and hoping one path gets lucky

Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x
  - Think of it as trying many options in parallel, and hoping one path gets lucky
- Transition $f(state, symbol) \mapsto \emptyset$ is possible

Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x
  - Think of it as trying many options in parallel, and hoping one path gets lucky
- Transition $f(state, symbol) \mapsto \varnothing$ is possible
  - … the NFA treats this as a rejecting path (the string may still reach an accepting state by another path)

Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x
  - Think of it as trying many options in parallel, and hoping one path gets lucky
- Transition $f(state, symbol) \mapsto \varnothing$ is possible
  - … the NFA treats this as a rejecting path (the string may still reach an accepting state by another path)
  - A convenient shortcut for our "hell/black-hole" state

Non-deterministic Finite Automaton

- An NFA accepts a string x if it *can* get to an accepting state on input x
  - Think of it as trying many options in parallel, and hoping one path gets lucky
- Transition $f(state, symbol) \mapsto \varnothing$ is possible
  - … the NFA treats this as a rejecting path (the string may still reach an accepting state by another path)
  - A convenient shortcut for our “hell/black-hole” state
  - <u>Class convention</u>: Draw all possible transitions for DFA. Not required for NFA (missing transitions lead to hell).

Non-deterministic Finite Automaton

- Every DFA is an NFA

Non-deterministic Finite Automaton

- Every DFA is an NFA
  - If we're strict with our notation, we need to replace the transition
    $f(state_1, symbol) \mapsto state_2$ with
    $f(state_1, symbol) \mapsto \{state_2\}$

Non-deterministic Finite Automaton

- Every DFA is an NFA
  - If we're strict with our notation, we need to replace the transition
    $f(state_1, symbol) \mapsto state_2$ with
    $f(state_1, symbol) \mapsto \{state_2\}$
- Every NFA can be simulated by a DFA

Non-deterministic Finite Automaton

- Every DFA is an NFA
  - If we're strict with our notation, we need to replace the transition
    $f(state_1, symbol) \mapsto state_2$ with
    $f(state_1, symbol) \mapsto \{state_2\}$
- Every NFA can be simulated by a DFA
  - ... i.e. they accept exactly the same language

Non-deterministic Finite Automaton

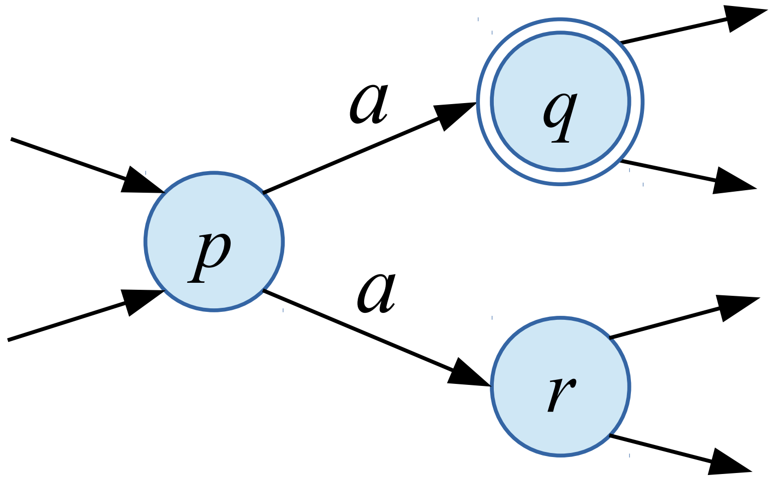
- Every DFA is an NFA
  - If we're strict with our notation, we need to replace the transition
    $f(state_1, symbol) \mapsto state_2$ with
    $f(state_1, symbol) \mapsto \{state_2\}$
- Every NFA can be simulated by a DFA
  - ... i.e. they accept exactly the same language
  - Exponential blowup: if the NFA has n states, the DFA can require up to 2^n states

Thought for the Day #1

Find an NFA with n states that can't be simulated by a DFA with less than 2^n states

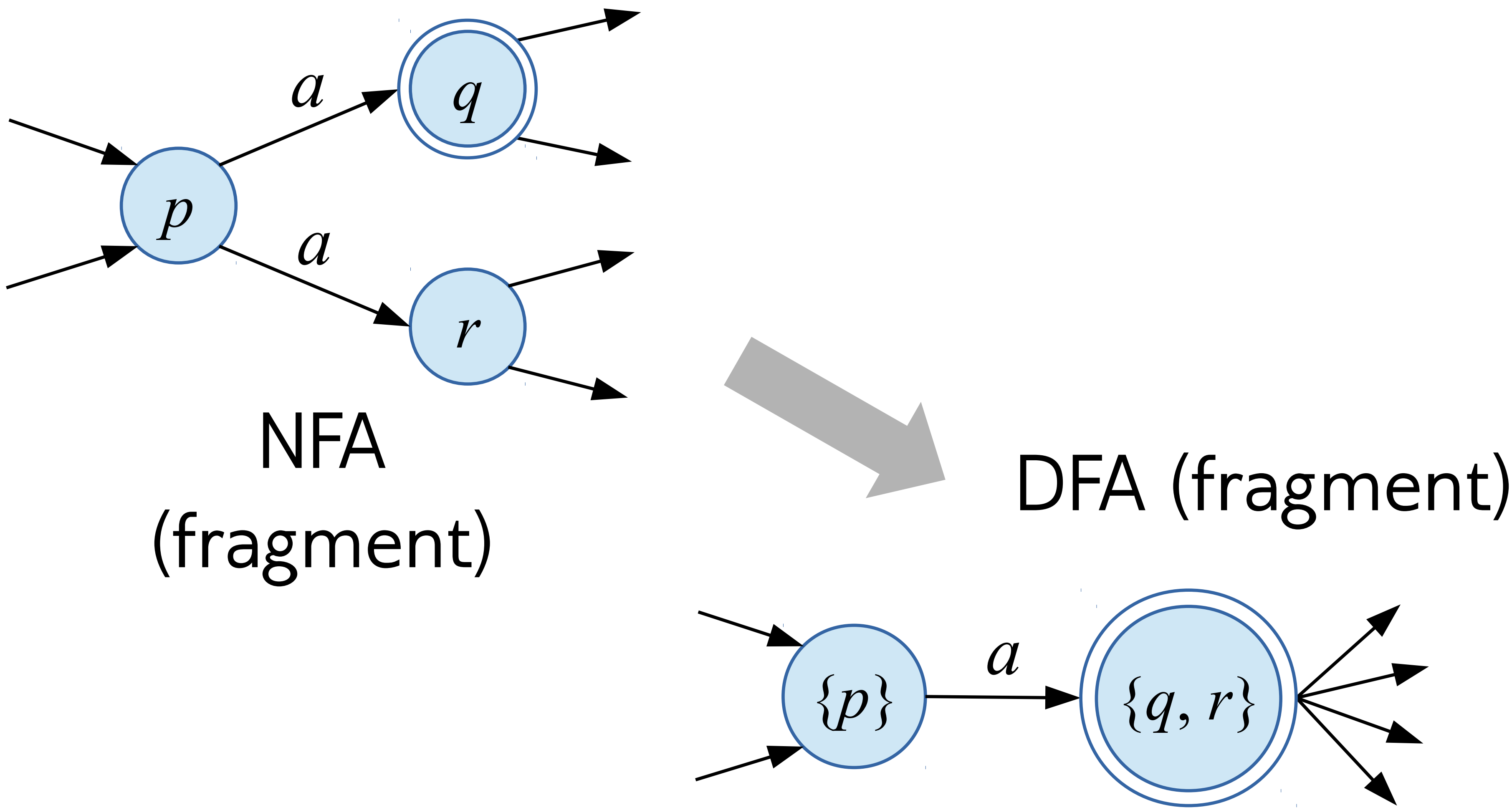
Every NFA can be simulated by a DFA

Every NFA can be simulated by a DFA



NFA
(fragment)

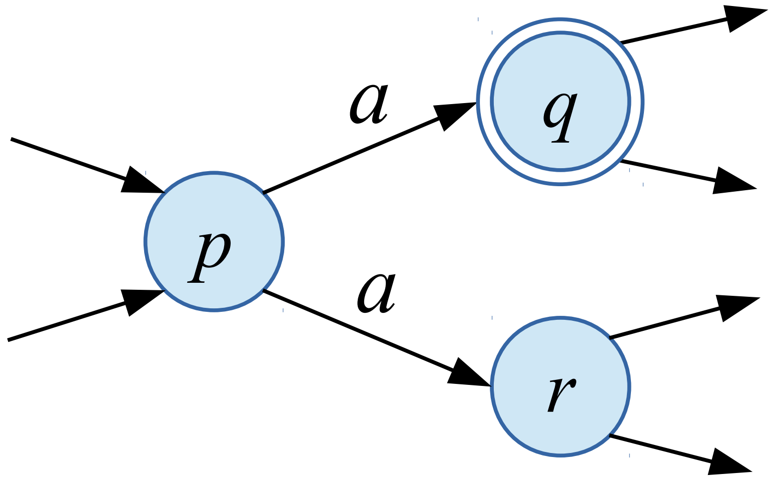
Every NFA can be simulated by a DFA



Every NFA can be simulated by a DFA

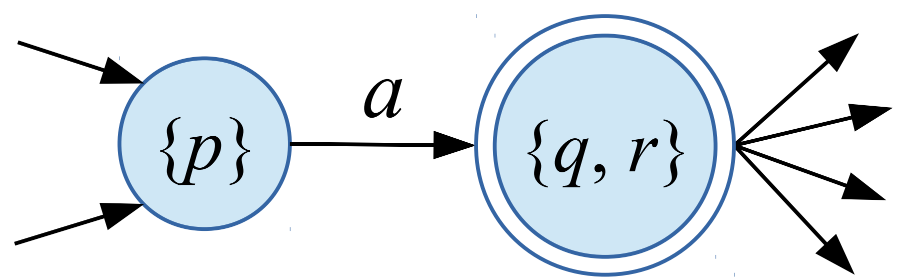
"Subset construction"

Main idea: ONE state of DFA tracks current states of ALL evolving paths of NFA



NFA (fragment)

DFA (fragment)



(This is merely illustrative – formal construction on following slides)

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification		
States		
Alphabet		
Transition Function		
Initial State		
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States		
Alphabet		
Transition Function		
Initial State		
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States	Q	2^Q
Alphabet		
Transition Function		
Initial State		
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States	Q	2^Q
Alphabet	Σ	Σ
Transition Function		
Initial State		
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States	Q	2^Q
Alphabet	Σ	Σ
Transition Function	δ	$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$
Initial State		
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States	Q	2^Q
Alphabet	Σ	Σ
Transition Function	δ	$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$
Initial State	q_0	$q'_0 = \{q_0\}$
Accepting States		

Every NFA can be simulated by a DFA

	NFA	Equivalent DFA
Specification	$(Q, \Sigma, \delta, q_0, F)$	$(2^Q, \Sigma, \delta', q'_0, F')$
States	Q	2^Q
Alphabet	Σ	Σ
Transition Function	δ	$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$
Initial State	q_0	$q'_0 = \{q_0\}$
Accepting States	F	$F' = \{S \in 2^Q \mid S \cap F \neq \varnothing\}$

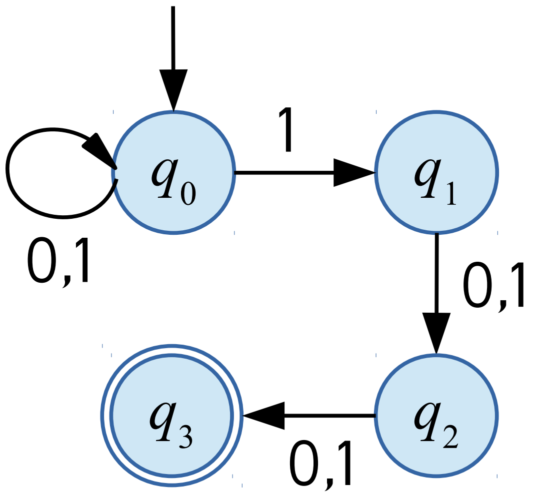
Why NFAs?

Why NFAs?

- They can be way more compact than DFAs

Why NFAs?

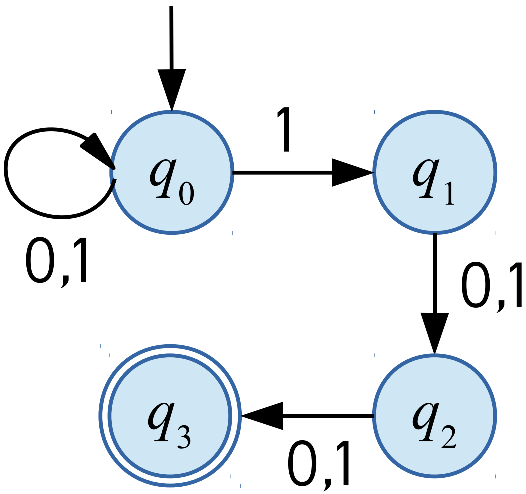
- They can be way more compact than DFAs



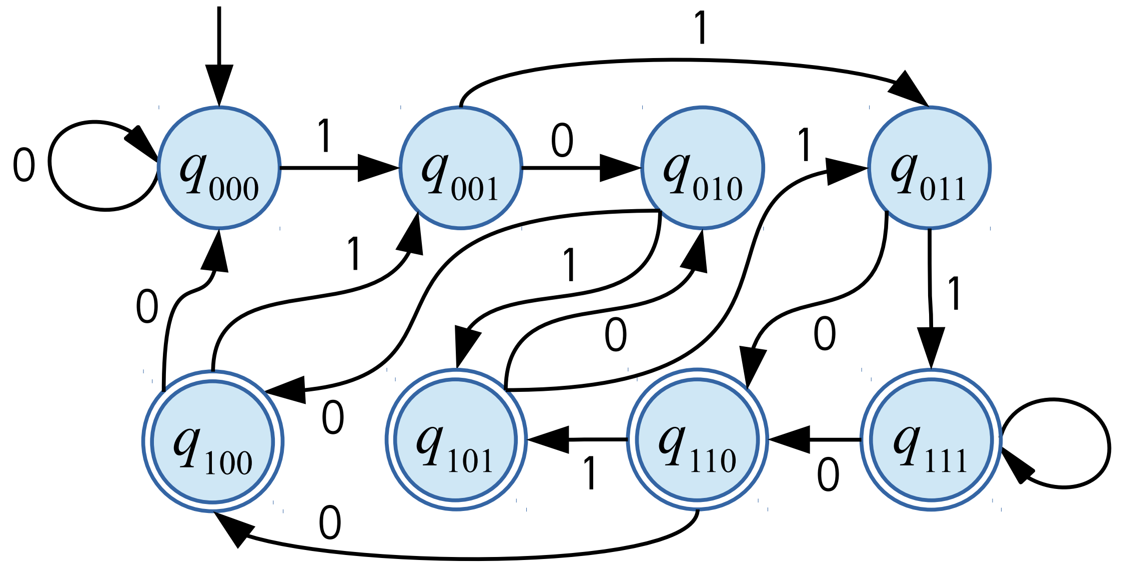
NFA

Why NFAs?

- They can be way more compact than DFAs



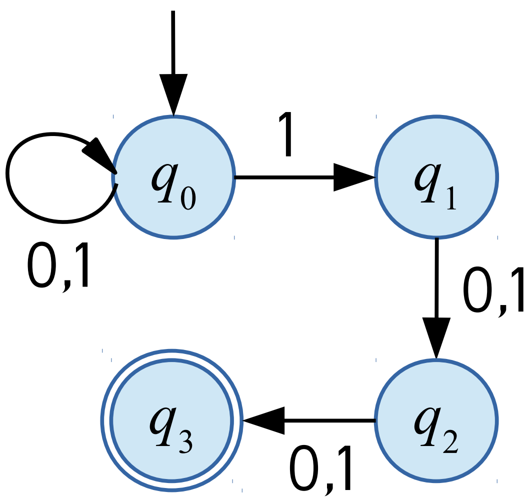
NFA



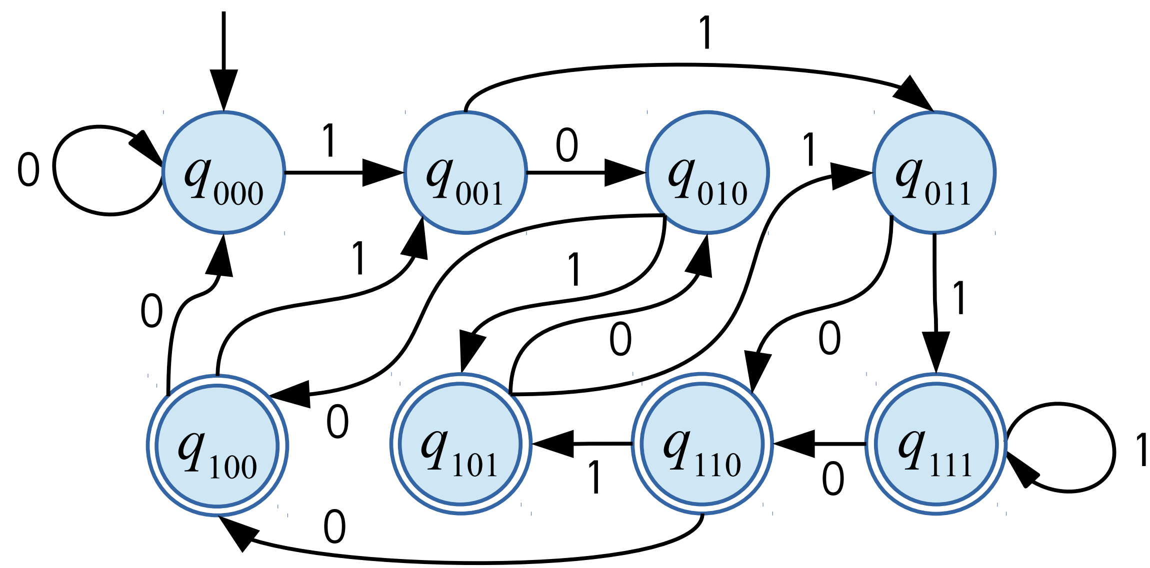
Equivalent minimal DFA

Why NFAs?

- They can be way more compact than DFAs



NFA



Equivalent minimal DFA

- It's easier to directly convert regular expressions (“wildcard search” on steroids) to NFAs

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

XKCD

Playing with regexes

- http://regex101.com/
- http://rubular.com/
- http://www.google.com/search?q=online+regex+tester

Regular expressions (“regex”-es) are defined by *structural induction*

(start with simple base expressions, construct more complicated ones recursively)

Empty set

$$L(\emptyset) = \emptyset$$

Empty string

$$L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$$

Literal character

$$L(\mathbf{x}) = \{x\}$$

e.g. $L(\mathbf{1}) = \{1\}, L(\mathbf{2}) = \{2\}, L(\mathbf{a}) = \{a\}$

Concatenation

$$L(AB) = \{ab \mid a \in L(A), \mathrm{b} \in L(B)\}$$

e.g. $L(\mathbf{12}) = \{12\}, L(\mathbf{aabb}) = \{aabb\}$

Concatenation

$$L(AB) = \{ab \mid a \in L(A), \mathrm{b} \in L(B)\}$$

e.g. $L(\mathbf{12}) = \{12\}$, $L(\mathbf{aabb}) = \{aabb\}$,
$L(\mathbf{a\varepsilon}) = ?$

Concatenation

$$L(AB) = \{ab \mid a \in L(A), \mathrm{b} \in L(B)\}$$

e.g. $L(\mathbf{12}) = \{12\}$, $L(\mathbf{aabb}) = \{aabb\}$,
$L(\mathbf{a\varepsilon}) = \{a\}$

Concatenation

$$L(AB) = \{ab \mid a \in L(A), \mathrm{b} \in L(B)\}$$

e.g. $L(\mathbf{12}) = \{12\}$, $L(\mathbf{aabb}) = \{aabb\}$,
$L(\mathbf{a\varepsilon}) = \{a\}$, $L(\mathbf{a}\varnothing) = ?$

Concatenation

$$L(AB) = \{ab \mid a \in L(A), \mathrm{b} \in L(B)\}$$

e.g. $L(\mathbf{12}) = \{12\}, L(\mathbf{aabb}) = \{aabb\},$
$L(\mathbf{a\varepsilon}) = \{a\}, L(\mathbf{a}\varnothing) = \varnothing$

Alternation

$$L(A|B) = L(A) \cup L(B)$$

e.g. $L(\mathbf{1}|\mathbf{2}) = \{1, 2\}$, $L(\mathbf{aa}|\mathbf{bb}) = \{aa, bb\}$

Alternation

$$L(A|B) = L(A) \cup L(B)$$

e.g. $L(\mathbf{1}|\mathbf{2}) = \{1, 2\}$, $L(\mathbf{aa}|\mathbf{bb}) = \{aa, bb\}$,
$L(\mathbf{a}|\boldsymbol{\varepsilon}) = ?$

Alternation

$$L(A|B) = L(A) \cup L(B)$$

e.g. $L(\mathbf{1}|\mathbf{2}) = \{1, 2\}$, $L(\mathbf{aa}|\mathbf{bb}) = \{aa, bb\}$,
$L(\mathbf{a}|\boldsymbol{\varepsilon}) = \{a, \varepsilon\}$

Alternation

$$L(A|B) = L(A) \cup L(B)$$

e.g. $L(\mathbf{1}|\mathbf{2}) = \{1, 2\}$, $L(\mathbf{aa}|\mathbf{bb}) = \{aa, bb\}$,
$L(\mathbf{a}|\boldsymbol{\varepsilon}) = \{a, \varepsilon\}$, $L(\mathbf{a}|\emptyset) = ?$

Alternation

$$L(A|B) = L(A) \cup L(B)$$

e.g. $L(\mathbf{1}|\mathbf{2}) = \{1, 2\}$, $L(\mathbf{aa}|\mathbf{bb}) = \{aa, bb\}$,
$L(\mathbf{a}|\boldsymbol{\varepsilon}) = \{a, \varepsilon\}$, $L(\mathbf{a}|\emptyset) = \{a\}$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = ?$

Kleene star

$$L(A^*) = \{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\ x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,
$L((\mathbf{a}|\mathbf{b})^*) = ?$

Kleene star

$$L(A^*) =$$

$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,
$L((\mathbf{a}|\mathbf{b})^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,
$L((\mathbf{a}|\mathbf{b})^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$,
$L(\boldsymbol{\varepsilon}^*) = ?$

Kleene star

$$L(A^*) =$$

$$\{\varepsilon\} \cup \{\ x_1 x_2 \ldots x_n \mid n \in \mathbf{N}, x_i \in L(A)\ \}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,

$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,

$L((\mathbf{a}|\mathbf{b})^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$,

$L(\boldsymbol{\varepsilon}^*) = \{\varepsilon\}$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,
$L((\mathbf{a}|\mathbf{b})^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$,
$L(\boldsymbol{\varepsilon}^*) = \{\varepsilon\}$, $L(\varnothing^*) = ?$

Kleene star

$$L(A^*) =$$
$$\{\varepsilon\} \cup \{\, x_1 x_2 \ldots x_n \mid n \in \mathbf{N},\, x_i \in L(A) \,\}$$

e.g. $L(\mathbf{a}^*) = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$,
$L((\mathbf{ab})^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$,
$L((\mathbf{a}|\mathbf{b})^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$,
$L(\boldsymbol{\varepsilon}^*) = \{\varepsilon\}$, $L(\varnothing^*) = \{\varepsilon\}$