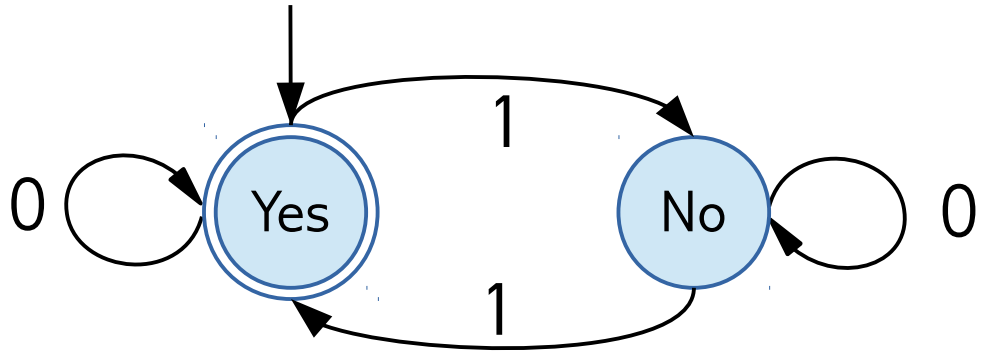
Finite Automata (contd)

CS 2800: Discrete Structures, Fall 2014

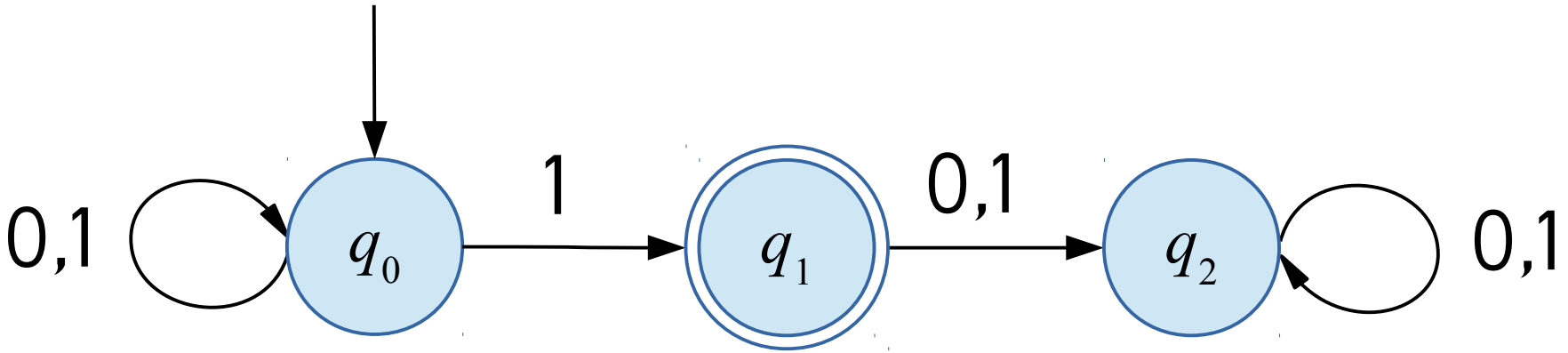
Sid Chaudhuri

Recap: Deterministic Finite Automaton

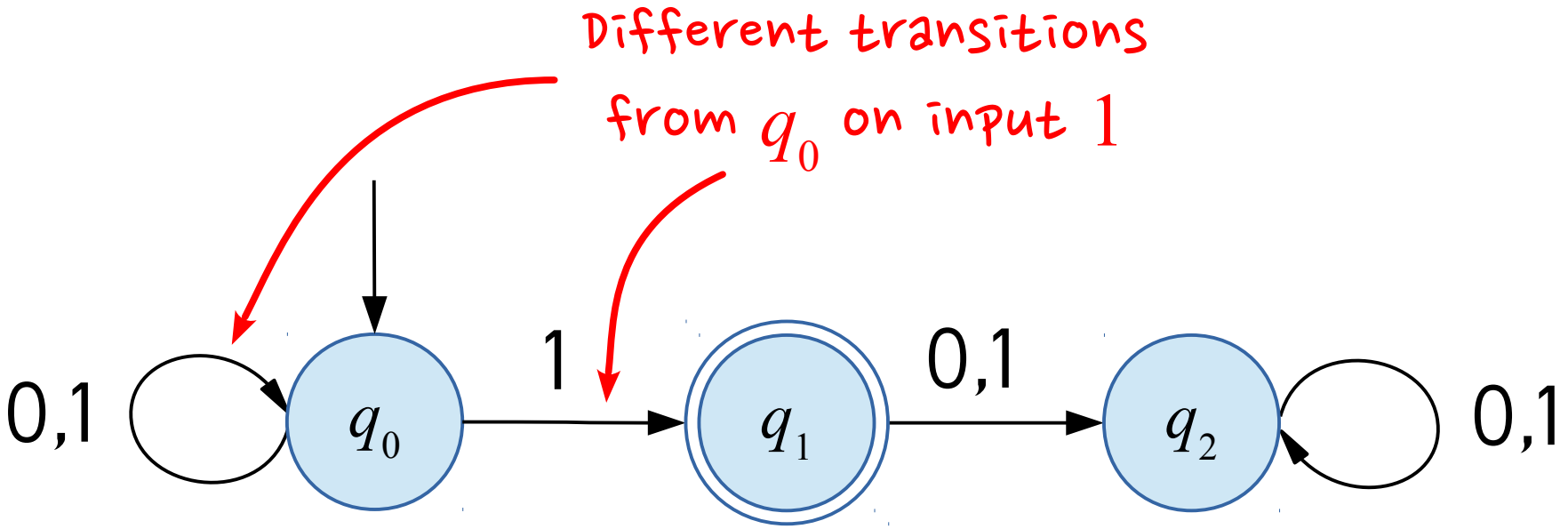
- A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \rightarrow Q$
  - $q_0 \in Q$ is the start/initial state
  - $F \subseteq Q$ is the set of final/accepting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



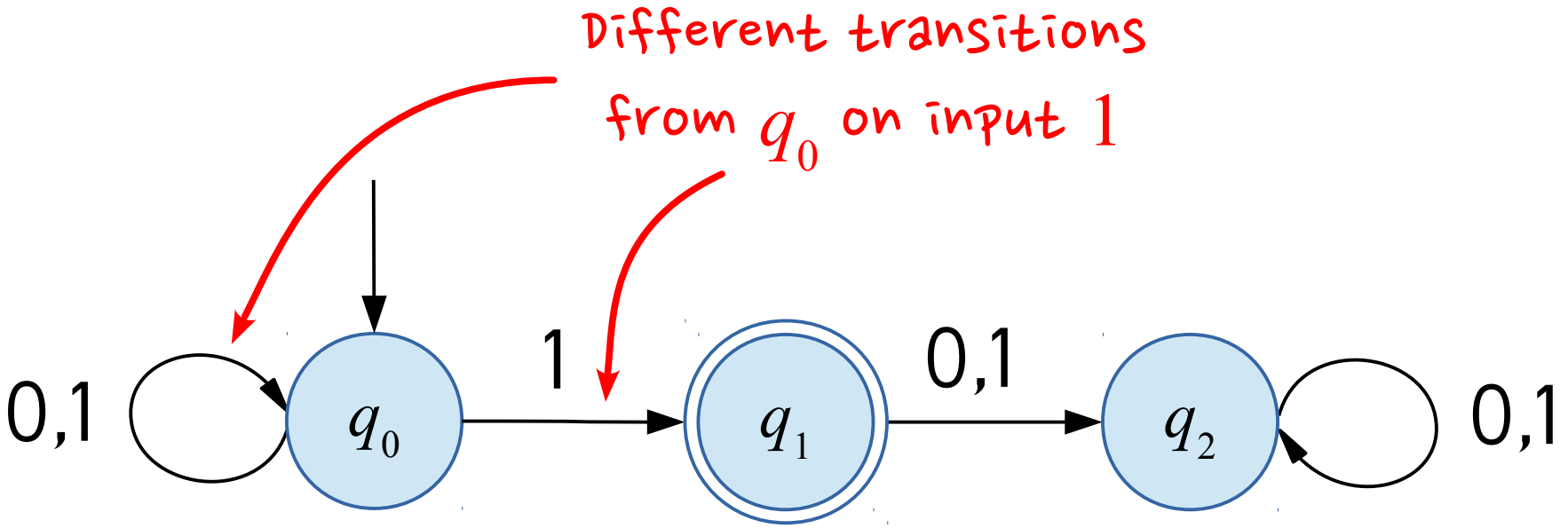
A ***non***-deterministic finite automaton



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

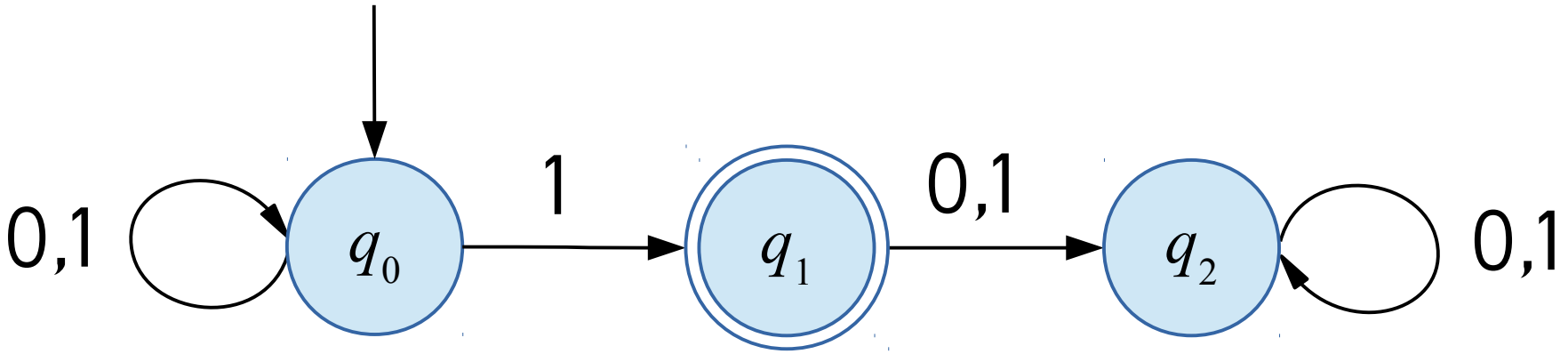


A ***non***-deterministic finite automaton



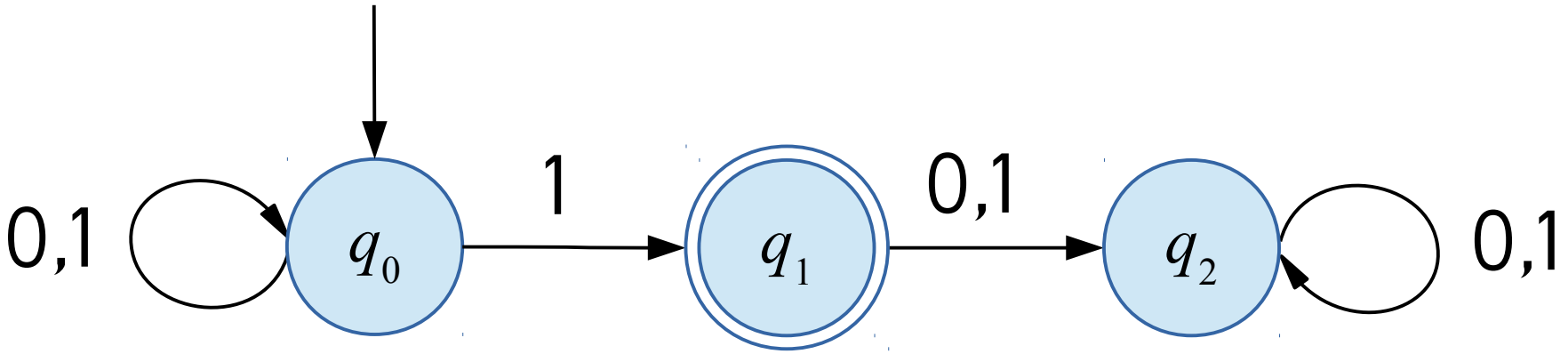
An NFA accepts a string x if it *can* get
to an accepting state on input x

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



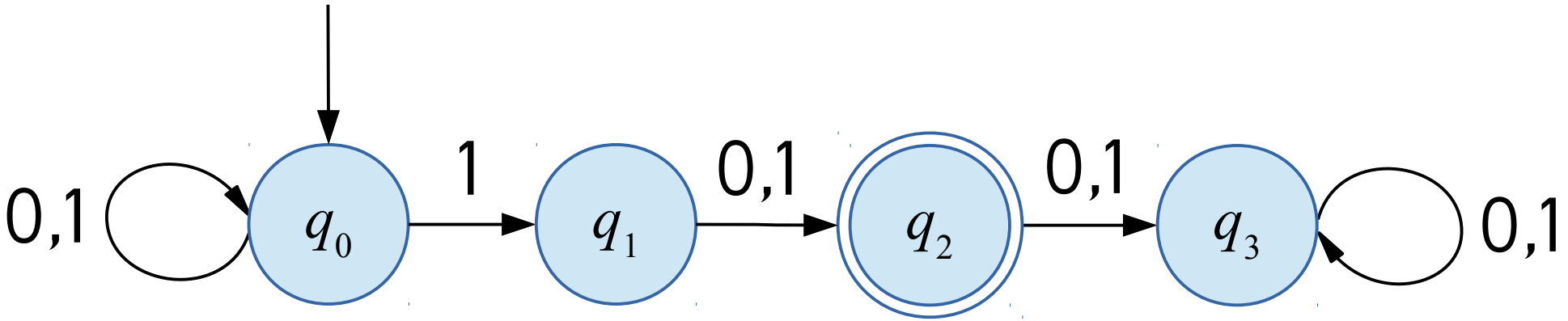
What language does this automaton accept?

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



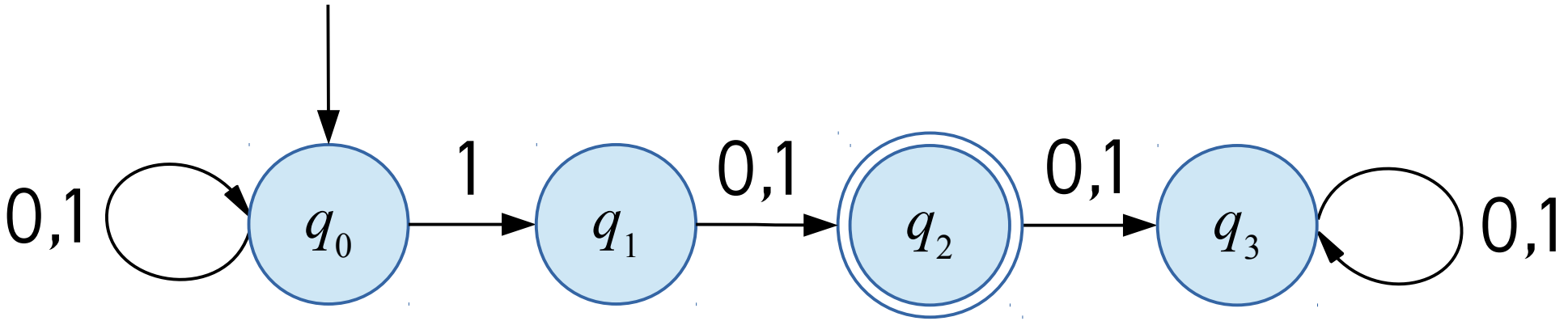
Answer: All strings ending with 1

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

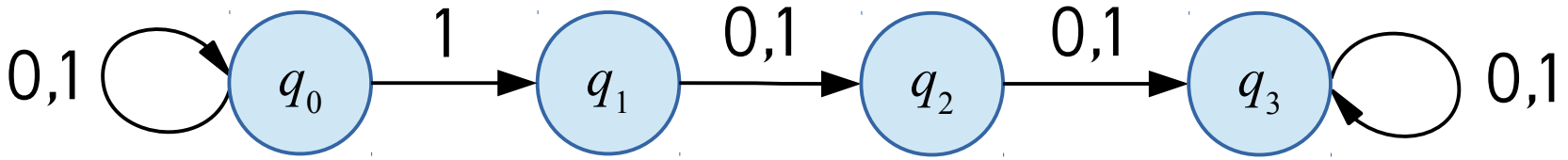
- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states

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

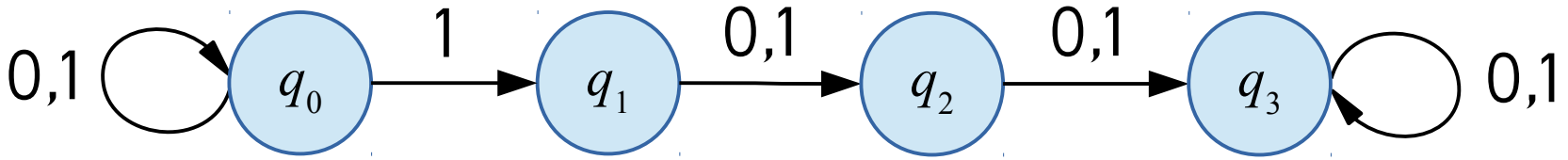
- An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$
  - Q is a finite set of states
  - Σ is a finite input alphabet (e.g. $\{0, 1\}$)
  - δ is a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$



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 $\delta : Q \times \Sigma \rightarrow 2^Q$

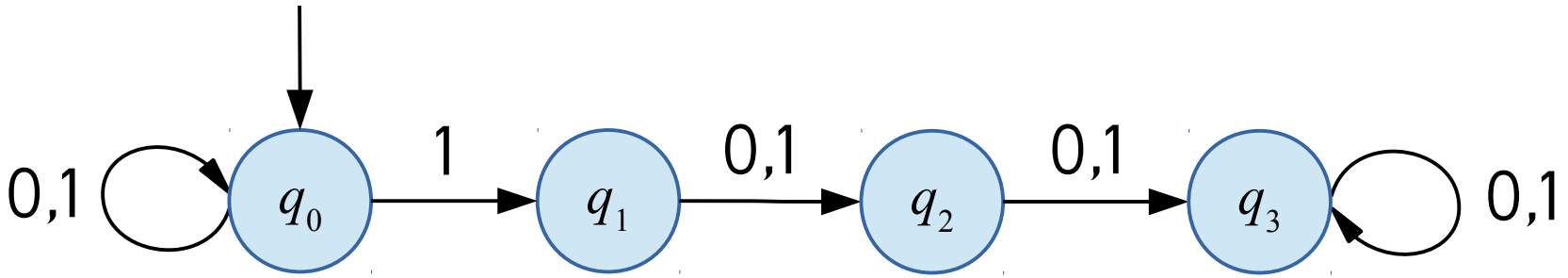
only change from DFAs



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

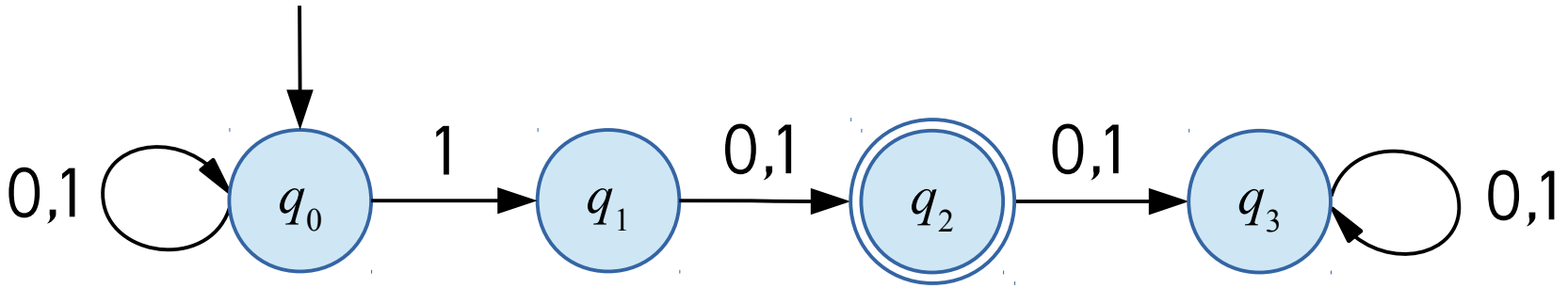
only change from DFAs



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 $\delta : Q \times \Sigma \rightarrow 2^Q$
  - $q_0 \in Q$ 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 at once, 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 at once, and hoping one path gets lucky
- A transition $f(state, symbol) \mapsto \varnothing$ (the empty set) 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 at once, and hoping one path gets lucky
- A transition $f(state, symbol) \mapsto \varnothing$ (the empty set) is possible
  - ... in this case, 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 at once, and hoping one path gets lucky
- A transition $f(state, symbol) \mapsto \varnothing$ (the empty set) is possible
  - ... in this case, 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

- 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

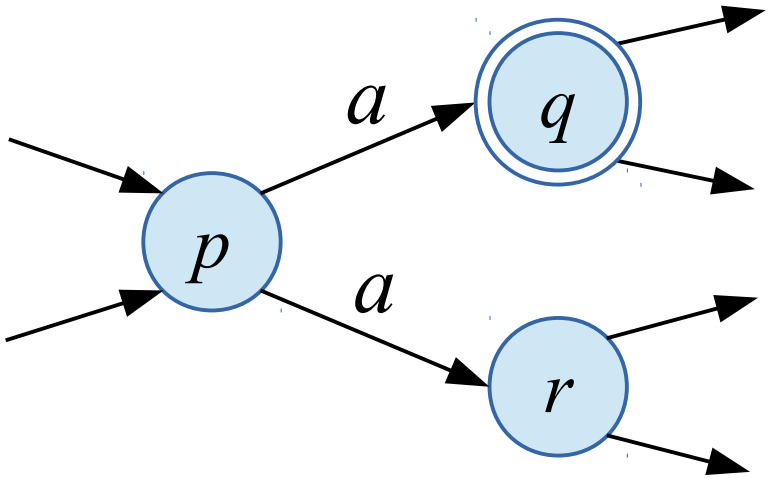
- Every DFA is an NFA
  - If we're strict with our notation, we need to replace the transition
    $f(state_1, symbol) \mapsto state_2$ with
    $f(state_1, symbol) \mapsto \{state_2\}$
- Every NFA can be simulated by a DFA
  - ... i.e. they accept exactly the same language
  - Exponential blowup: if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

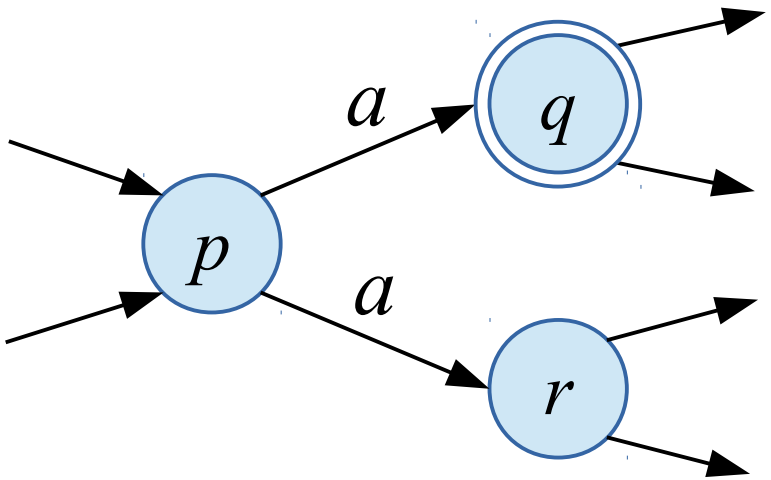
Every NFA can be simulated by a DFA

Every NFA can be simulated by a DFA



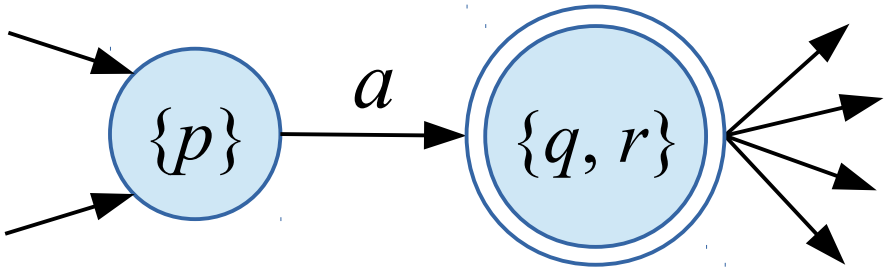
NFA
(fragment)

Every NFA can be simulated by a DFA

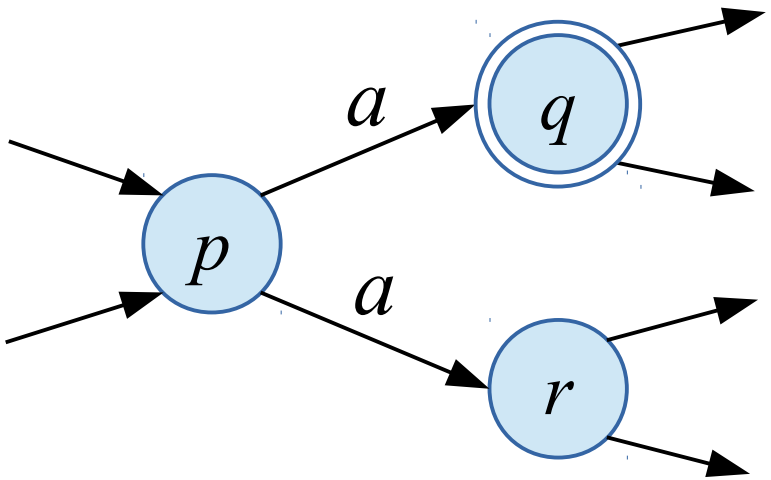


NFA
(fragment)

DFA (fragment)



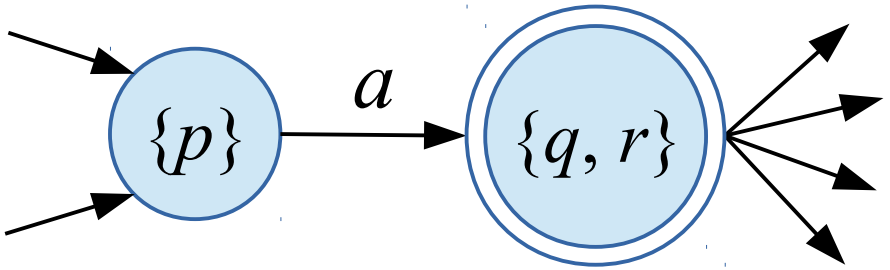
Every NFA can be simulated by a DFA



NFA
(fragment)

"Subset construction"

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	$F' = \{S \in 2^Q \mid S \cap F \neq \emptyset\}$

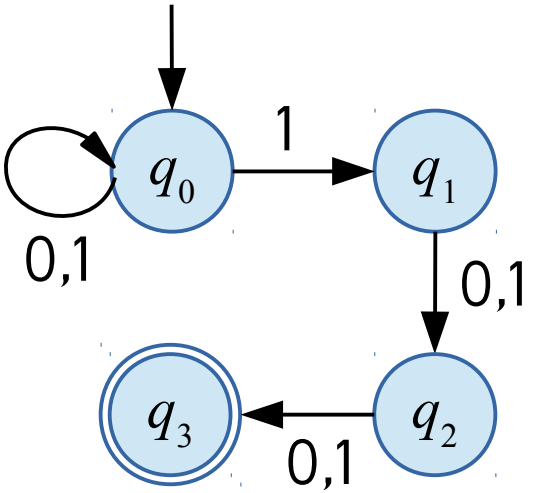
Why NFAs?

Why NFAs?

- They can be way more compact than DFAs

Why NFAs?

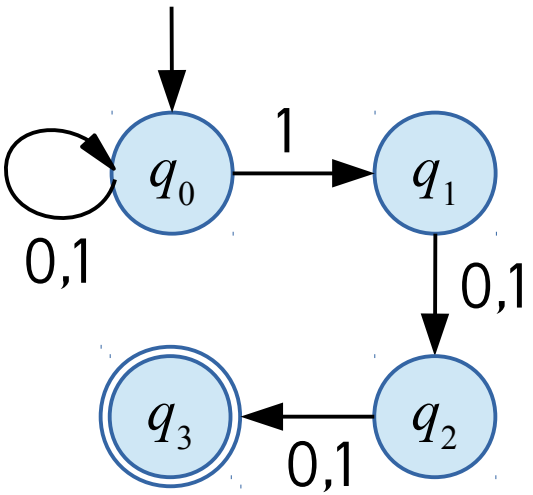
- They can be way more compact than DFAs



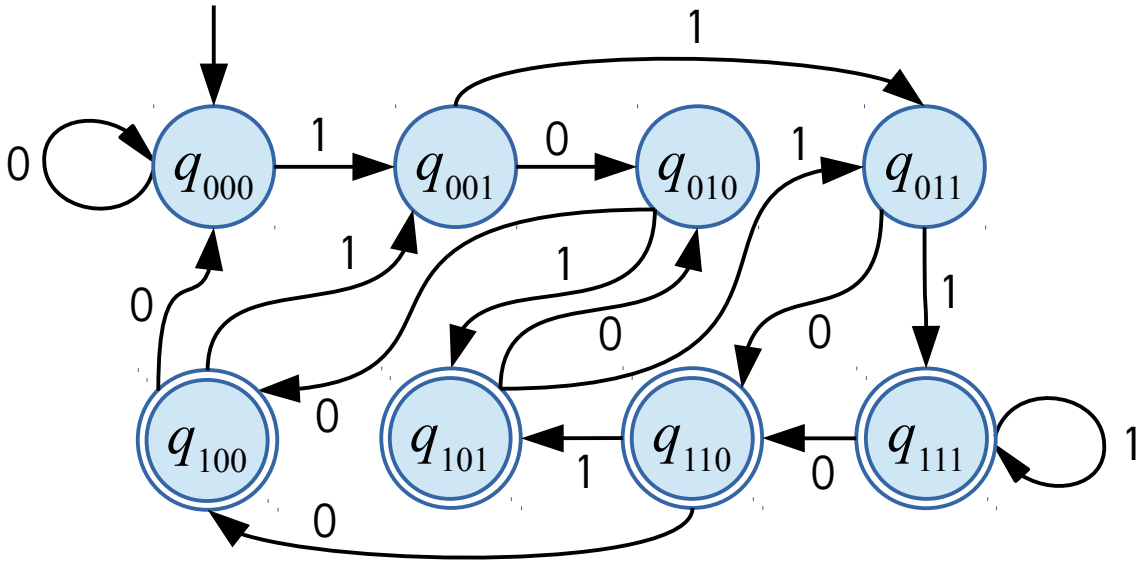
NFA

Why NFAs?

- They can be way more compact than DFAs



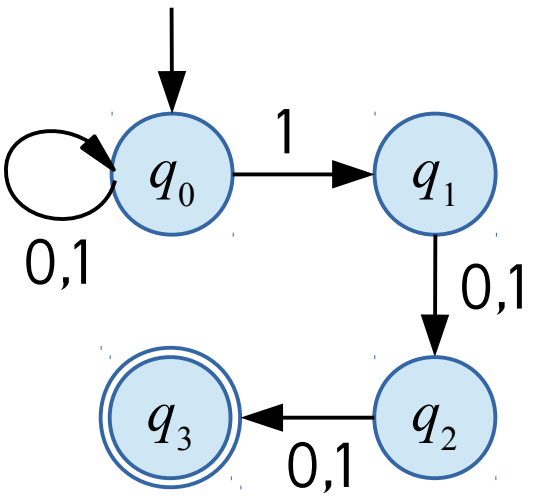
NFA



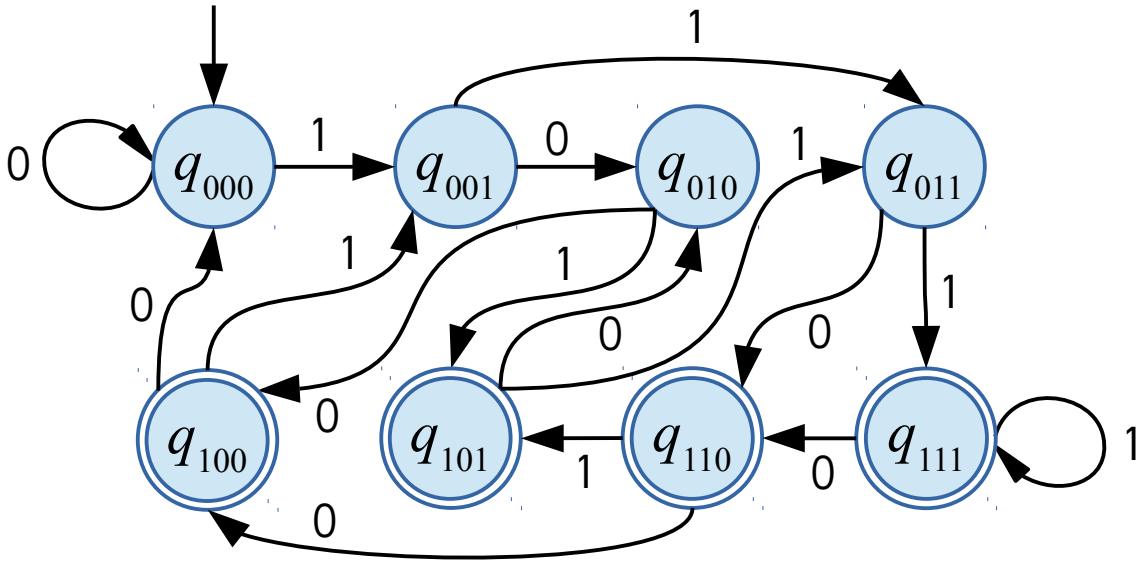
Equivalent minimal DFA

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

Finite automata struggle to count

- Any FA (deterministic or not) that recognizes the language $\{1^c\}$ (the single string of c 1's) must have more than c states
  - The parent alphabet is irrelevant (but must of course contain 1)

Proof: FAs struggle to count

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states
- On input 1^c, M can traverse states $q_0, q_1, q_2, \ldots, q_c$

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states
- On input 1^c, M can traverse states $q_0, q_1, q_2, \ldots, q_c$
  - ... where q_c is an accepting state

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states
- On input 1^c, M can traverse states $q_0, q_1, q_2, \ldots, q_c$
  - ... where q_c is an accepting state
- If M has $\leq c$ states, some state must be repeated in this sequence, say $q_i = q_j$ with $i < j$

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states
- On input 1^c, M can traverse states $q_0, q_1, q_2, \ldots, q_c$
  - ... where q_c is an accepting state
- If M has $\leq c$ states, some state must be repeated in this sequence, say $q_i = q_j$ with $i < j$

Pigeonhole Principle

Proof: FAs struggle to count

- Imagine there is some FA M that recognizes the language $\{1^c\}$ and has $\leq c$ states
- On input 1^c, M can traverse states $q_0, q_1, q_2, \ldots, q_c$
  - ... where q_c is an accepting state
- If M has $\leq c$ states, some state must be repeated in this sequence, say $q_i = q_j$ with $i < j$
  - Let $t = j - i$

Proof: FAs struggle to count

- M can take the following steps on input 1^c

Proof: FAs struggle to count

- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i

Proof: FAs struggle to count

- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$

Proof: FAs struggle to count

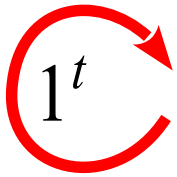
- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}

Proof: FAs struggle to count

- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}
- Step 2 can be repeated indefinitely by inserting copies of 1^t in the string!

Proof: FAs struggle to count

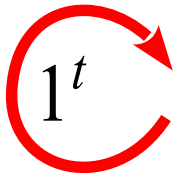
- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}



- Step 2 can be repeated indefinitely by inserting copies of 1^t in the string!

Proof: FAs struggle to count

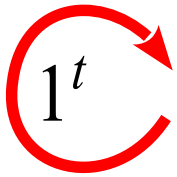
- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}



- Step 2 can be repeated indefinitely by inserting copies of 1^t in the string!
- So M also accepts 1^{c+t}, 1^{c+2t}, 1^{c+3t} etc

Proof: FAs struggle to count

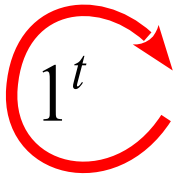
- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}



- Step 2 can be repeated indefinitely by inserting copies of 1^t in the string!
- So M also accepts 1^{c+t}, 1^{c+2t}, 1^{c+3t} etc
  - ... which is a contradiction!

Proof: FAs struggle to count

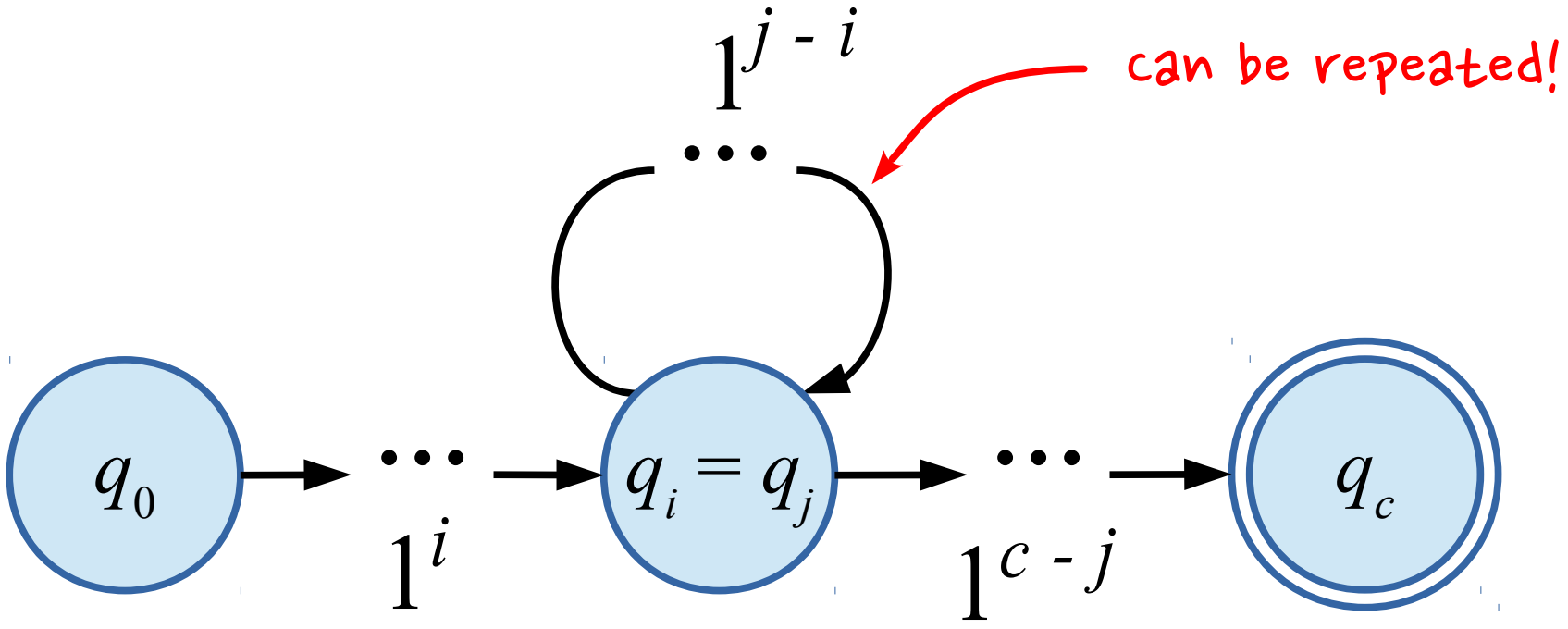
- M can take the following steps on input 1^c
  1. After reading 1^i, M is in state q_i
  2. Then it reads 1^t and loops back to $q_j = q_i$
  3. Then it finishes off by reading 1^{c-j}



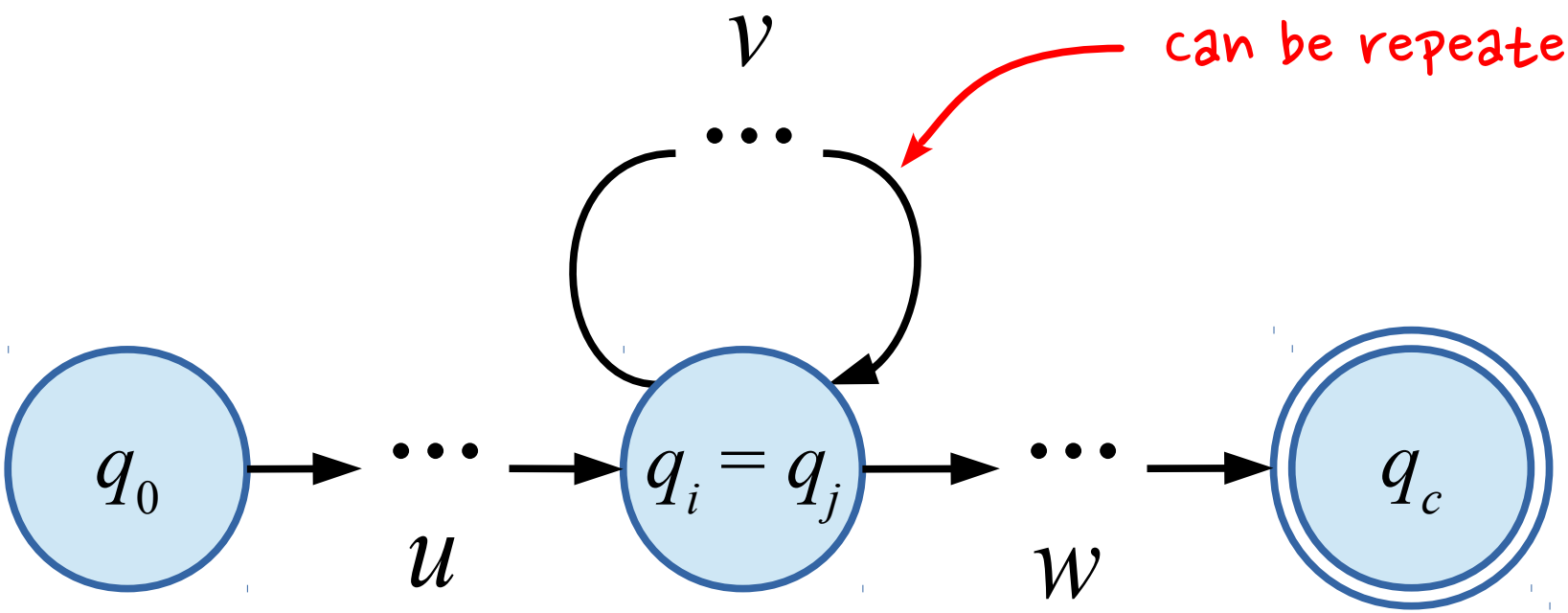
- Step 2 can be repeated indefinitely by inserting copies of 1^t in the string!
- So M also accepts 1^{c+t}, 1^{c+2t}, 1^{c+3t} etc
  - ... which is a contradiction!

QED

FAs struggle to count



Generalizing...



Pumping Lemma

Pumping Lemma

- If M is an FA with k states

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that
  - $x = uvw$,

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that
  - $x = uvw$,
  - $|uv| \leq k$,

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that
  - $x = uvw$,
  - $|uv| \leq k$,
  - $|v| \geq 1$

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that
  - $x = uvw$,
  - $|uv| \leq k$,
  - $|v| \geq 1$
- ... and uv^iw is accepted by M for all $i \in \mathbf{N} \cup \{0\}$

Pumping Lemma

- If M is an FA with k states
- ... and M accepts some string x with $|x| \geq k$,
- ... then there exist strings u, v and w such that
  - $x = uvw$,
  - $|uv| \leq k$,
  - $|v| \geq 1$
- ... and uv^iw is accepted by M for all $i \in \mathbf{N} \cup \{0\}$

Exercise: write out the proof formally!

Why is the pumping lemma useful?

Why is the pumping lemma useful?

- Used to prove languages *cannot* be recognized by any FA whatsoever

Why is the pumping lemma useful?

- Used to prove languages *cannot* be recognized by any FA whatsoever
- E.g. the language with all strings of 0's concatenated with equally long strings of 1's, i.e. $0^n 1^n$ for all $n \in \boldsymbol{N}$

Why is the pumping lemma useful?

- Used to prove languages *cannot* be recognized by any FA whatsoever
- E.g. the language with all strings of 0's concatenated with equally long strings of 1's, i.e. 0^n1^n for all $n \in \boldsymbol{N}$
  - FA's don't have unlimited “memory”, so they can't store the number of 0's they have seen

Proof: $0^n 1^n$ is not recognized by any FA

Proof: $0^n 1^n$ is not recognized by any FA

- We'll prove it by contradiction

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \boldsymbol{N}$

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \boldsymbol{N}$
- Consider 0^k1^k, which is accepted by M

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \boldsymbol{N}$
- Consider 0^k1^k, which is accepted by M
- It's longer than k, so by the Pumping Lemma, there exist u, v, w s.t. $uvw = 0^k1^k$, $|uv| \leq k$ and $|v| \geq 1$

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \boldsymbol{N}$
- Consider 0^k1^k, which is accepted by M
- It's longer than k, so by the Pumping Lemma, there exist u, v, w s.t. $uvw = 0^k1^k$, $|uv| \leq k$ and $|v| \geq 1$
  - ... and v must consist entirely of 0's (since $|uv| \leq k$)

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \mathbf{N}$
- Consider 0^k1^k, which is accepted by M
- It's longer than k, so by the Pumping Lemma, there exist u, v, w s.t. $uvw = 0^k1^k$, $|uv| \leq k$ and $|v| \geq 1$
  - ... and v must consist entirely of 0's (since $|uv| \leq k$)
- So M accepts all strings of the form $0^{|u|}0^{i|v|}0^{k - |uv|}1^k$, such as $0^{|u|}0^{k - |uv|}1^k$, $0^{|u|}0^{2|v|}0^{k - |uv|}1^k$, $0^{|u|}0^{3|v|}0^{k - |uv|}1^k$ etc

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \mathbf{N}$
- Consider 0^k1^k, which is accepted by M
- It's longer than k, so by the Pumping Lemma, there exist u, v, w s.t. $uvw = 0^k1^k$, $|uv| \leq k$ and $|v| \geq 1$
  - ... and v must consist entirely of 0's (since $|uv| \leq k$)
- So M accepts all strings of the form $0^{|u|}0^{i|v|}0^{k - |uv|}1^k$, such as $0^{|u|}0^{k - |uv|}1^k$, $0^{|u|}0^{2|v|}0^{k - |uv|}1^k$, $0^{|u|}0^{3|v|}0^{k - |uv|}1^k$ etc
  - ... which are not of the form 0^n1^n (since $|v| \geq 1$)

Proof: 0^n1^n is not recognized by any FA

- We'll prove it by contradiction
- Assume there is some FA M with k states that accepts a string iff it is of the form 0^n1^n for any $n \in \mathbf{N}$
- Consider 0^k1^k, which is accepted by M
- It's longer than k, so by the Pumping Lemma, there exist u, v, w s.t. $uvw = 0^k1^k$, $|uv| \leq k$ and $|v| \geq 1$
  - ... and v must consist entirely of 0's (since $|uv| \leq k$)
- So M accepts all strings of the form $0^{|u|}0^{i|v|}0^{k - |uv|}1^k$, such as $0^{|u|}0^{k - |uv|}1^k$, $0^{|u|}0^{2|v|}0^{k - |uv|}1^k$, $0^{|u|}0^{3|v|}0^{k - |uv|}1^k$ etc
  - ... which are not of the form 0^n1^n (since $|v| \geq 1$)
  - ... hence proved by contradiction