CS 121 Section 4: Finite Automata, Regular Languages, and their Limitations

James Roney

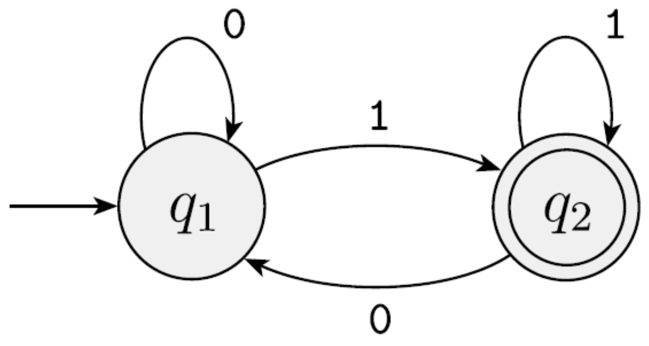
Outline

1. Review of Deterministic Finite Automata (DFAs) and Languages
2. Nondeterministic Finite Automata (NFAs)
3. NFAs, DFAs, and Regular Expressions are Equivalent
4. Proving Non-Regularity
5. The Pumping Lemma
6. Problems

Deterministic Finite Automata

- DFA = simple computer that can exist in a finite number of states
- Reads in a string of symbols, uses each symbol to determine next state
- At the end of the string, the machine “accepts” the input if it is in some predetermined set of “accepting states”
- In automata theory, a “language” is just a set of strings. If a DFA accepts the strings in language L, and only those strings, we say the DFA “recognizes” L.

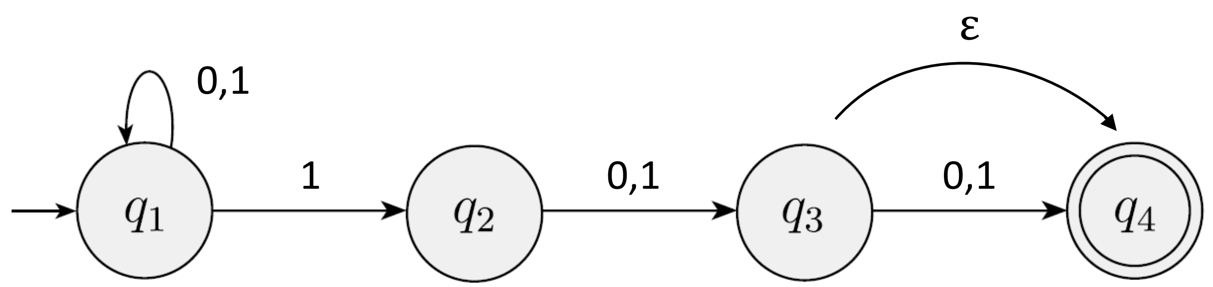
```python
def run_dfa(start_state, delta_fun, accept_states, input_str):
    curr_state = start_state
    for i in range(len(input_str)):
        curr_state = delta_fun(curr_state, input_str[i])
    if curr_state in accept_states:
        return "ACCEPT"
    return "REJECT"
```



This language recognized by this DFA is the set of strings which end in a 1.

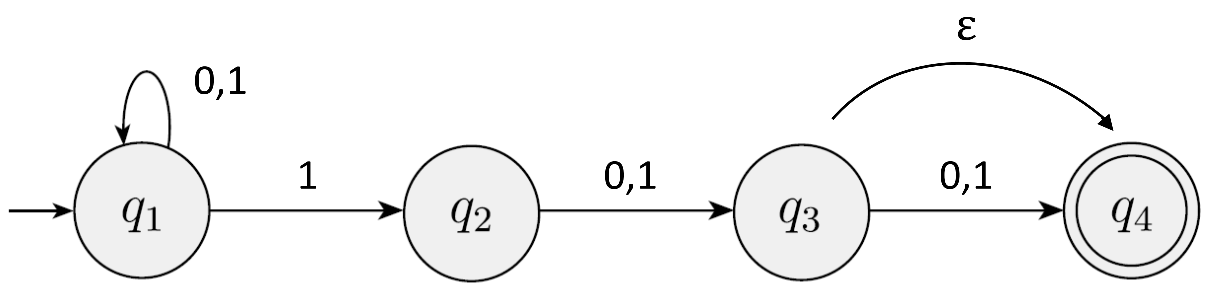
Non-Deterministic Finite Automata

- Unlike a DFA, a single state in an NFA can have any number of outward arrows labelled with the same symbol
  - For example, state q_1 below has 2 outward arrows for the symbol ‘1’, and state q_4 has no outward arrows at all
- Edges can also be labelled ε, which represents the empty string.
  - State q_3 below has an outward ε- arrow



Non-Deterministic Finite Automata

- Intuitive Behavior:
  - When an NFA finds multiple outward arrows corresponding to the current symbol, it makes multiple copies of itself and explores all the arrows at once
  - If it encounters an ε- arrow, it immediately copies itself to the end of the arrow without consuming any symbols. Then both copies continue consuming symbols in parallel
  - When a copy of the NFA finds itself in a state where there are no outward arrows for the current symbol, it "dies".
  - If a copy of the NFA survives until the end of the string and ends up in an accepting state, then entire computation accepts.



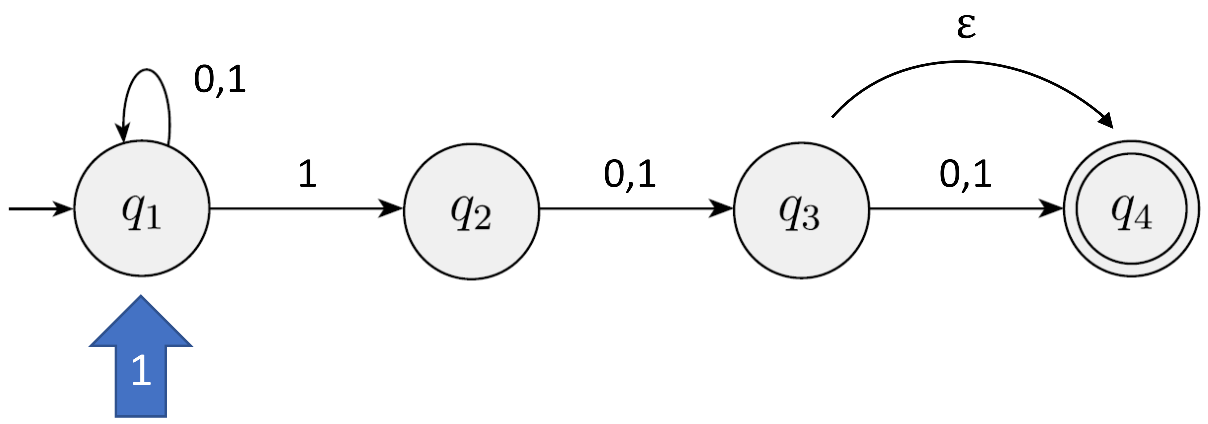
NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

$\uparrow$

Machine:

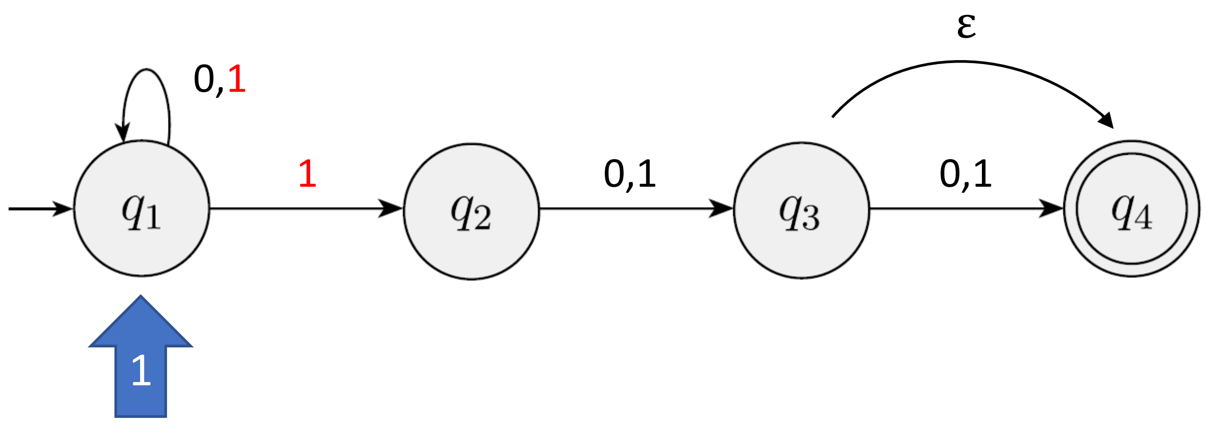


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010
$\uparrow$

Machine:

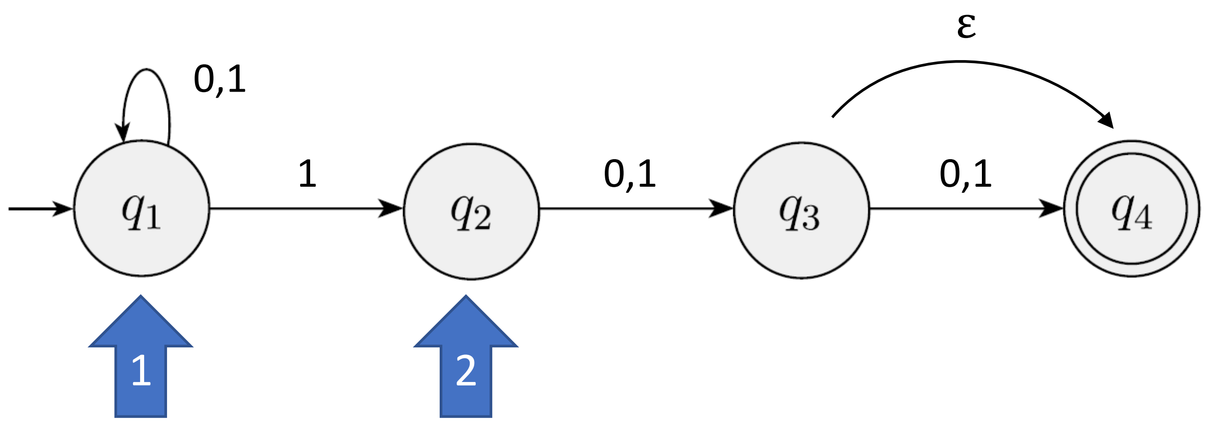


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010
$\uparrow$

Machine:

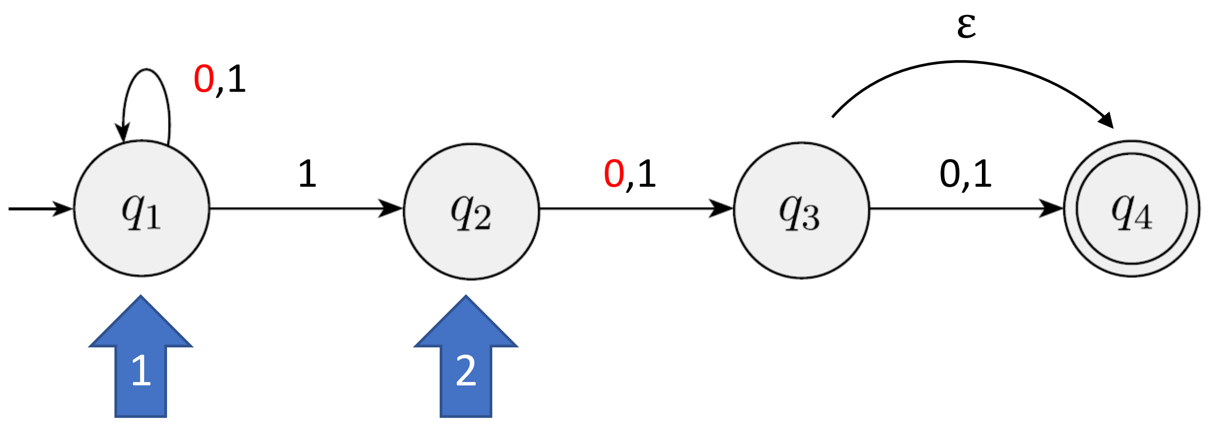


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

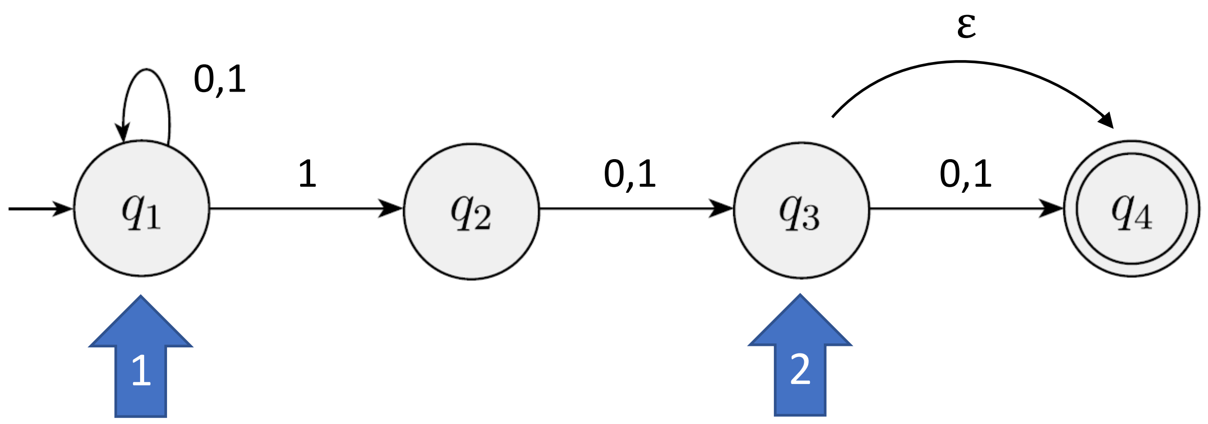


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010
$\uparrow$

Machine:

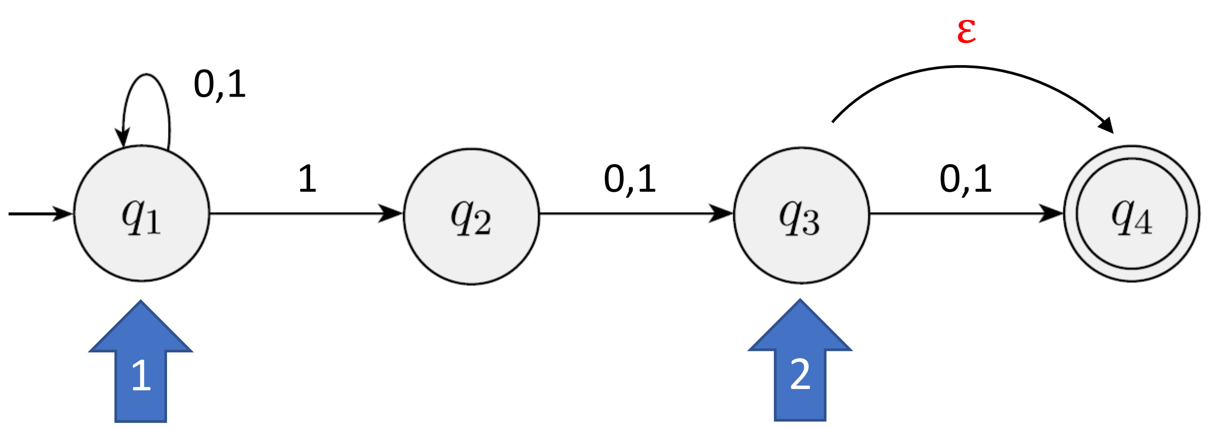


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

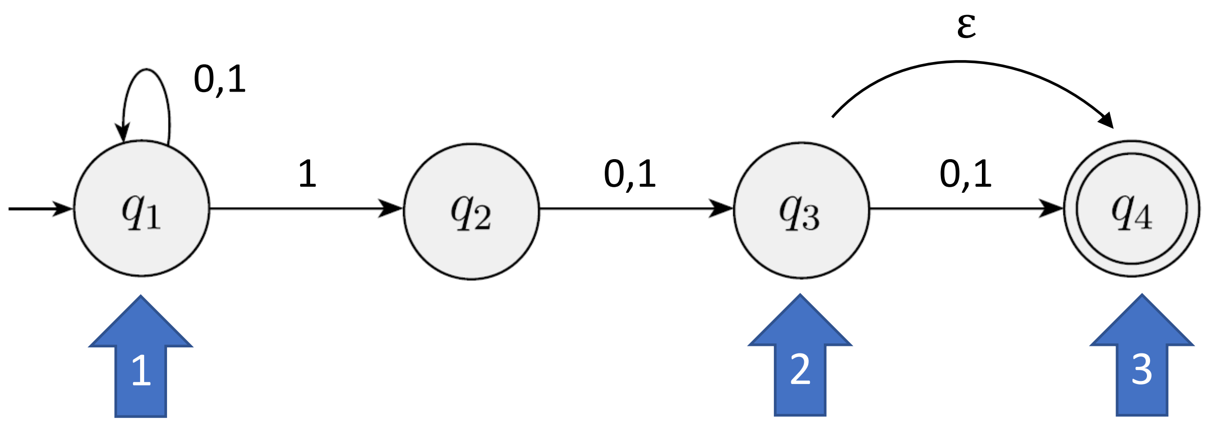


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: $1\underset{\uparrow}{0}10$

Machine:

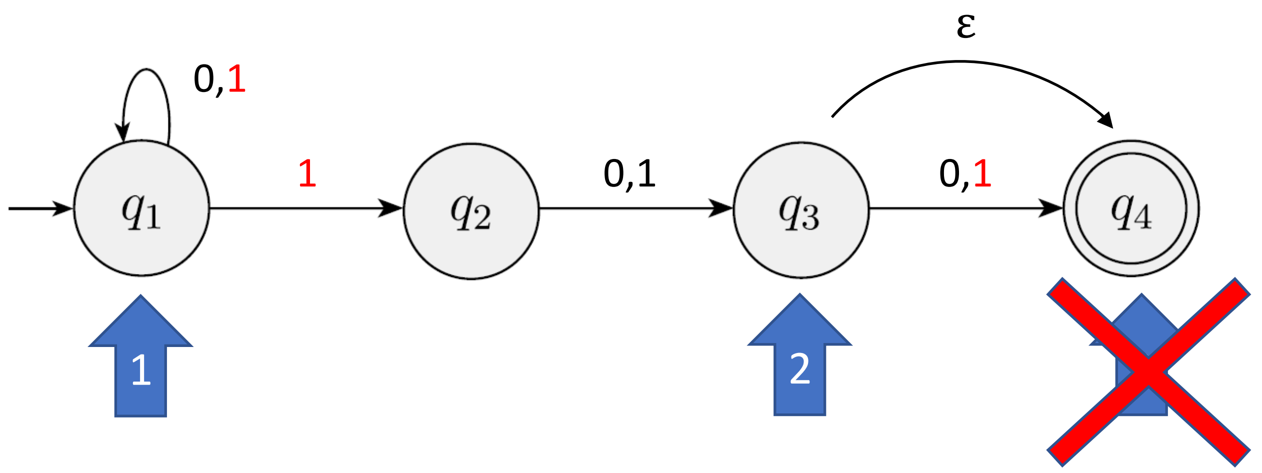


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

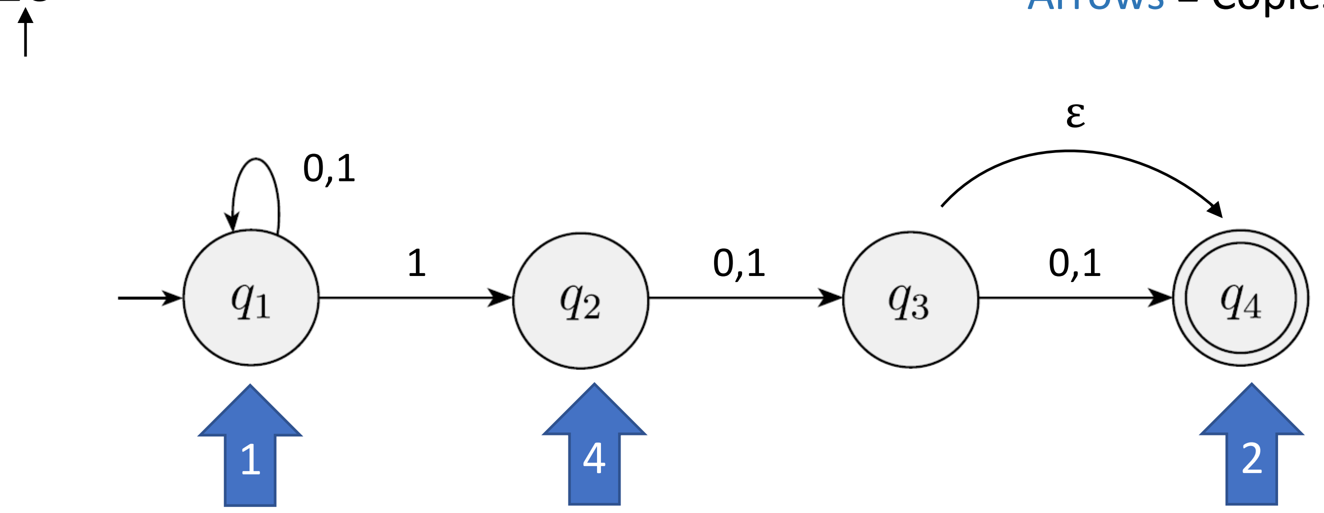


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

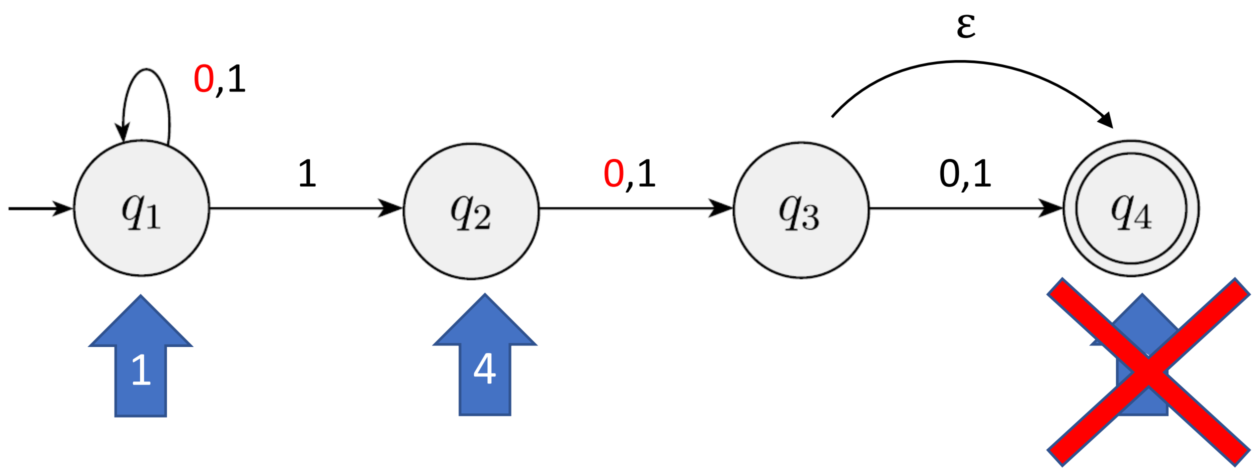


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010
$\uparrow$

Machine:

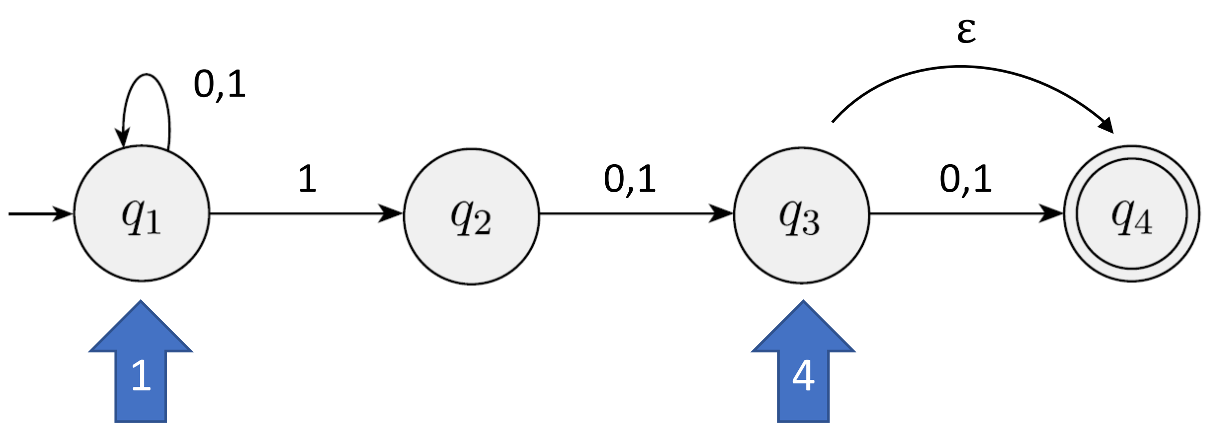


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

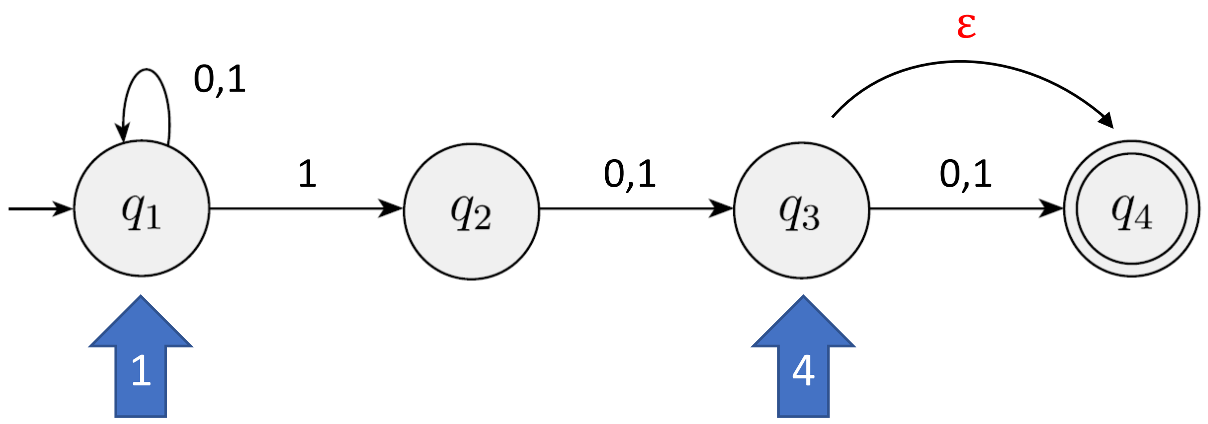


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

Input: 1010

Machine:

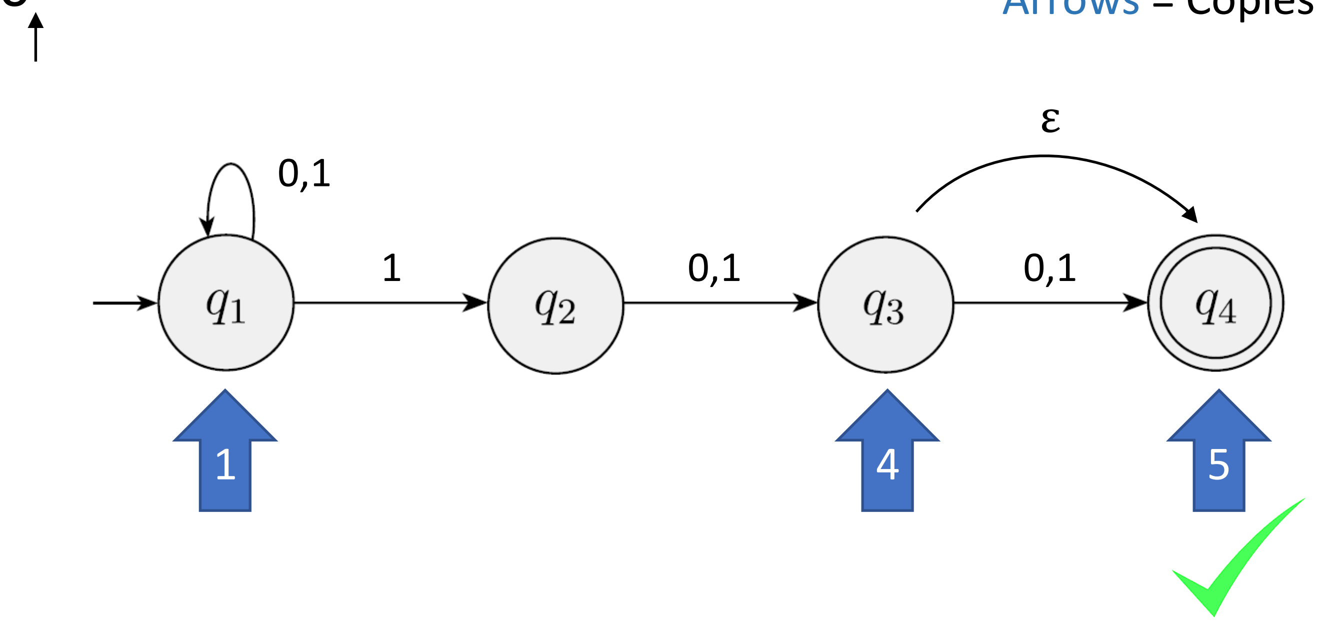


NFAs: Worked Example

Red Symbols = Symbols Being Currently Processed and edges being traversed
Arrows = Copies of the NFA

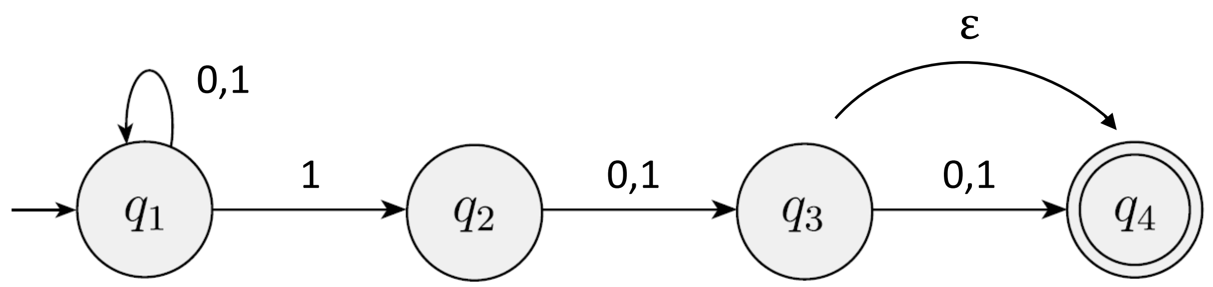
Input: 1010

Machine:



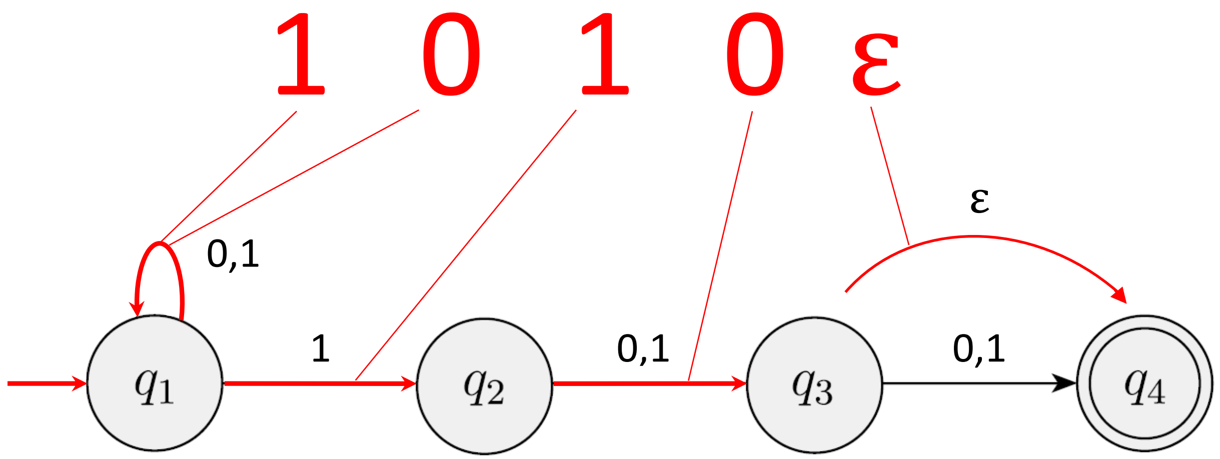
NFAs: Worked Example

- This machine recognizes the set of all strings with a ‘1’ in the third-to-last position or the second-to-last position
- Intuition: Each time a ‘1’ appears in the input sequence, a new copy of the machine tests whether that ‘1’ is the third-to-last or second-to-last symbol



NFAs: Formal Definition of Acceptance

- Formally, an NFA accepts an input if there is a path from the start state to an accepting state using ε-edges and edges with labels that match the symbols of the input.
- For the input 1010, such a path is illustrated on the machine below:

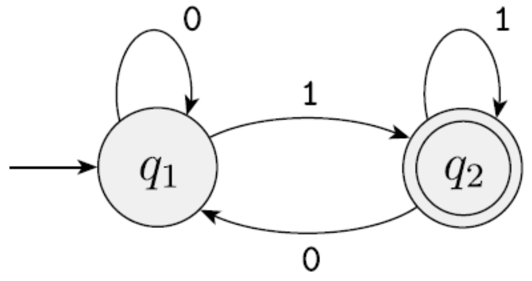


Equivalence of NFAs and DFAs

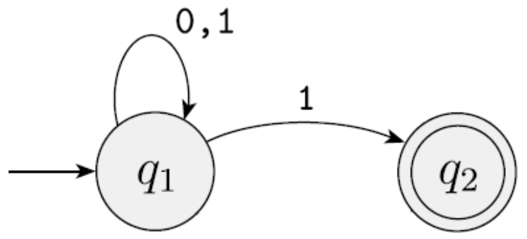
- While NFAs may seem much more complicated than DFAs, it turns out that NFAs and DFAs are actually equivalent!
- This means that any NFA can be converted into a DFA that recognizes the exact same language.
- Proof Idea:
  - If the NFA has a set of states Q we create a DFA with states $Q' = P(Q)$, or the set of all subsets of Q.
    - $P(Q)$ is known as the "power set" of Q.
    - Example: $P(\{0,1,2\}) = \{\{\ \},\{0\},\{1\},\{2\},\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}$
  - We simulate the behavior of the NFA using the DFA. At each step, the state of the DFA is just the set of all states occupied by copies of the NFA at that stage of the computation.

Why do we care about NFAs?

- It can be much easier to think of an NFA that solves a problem than a DFA, but we know that they are equivalent at the end of the day.
- In lecture we showed how to transform a regular expression into an equivalent NFA that recognizes the same strings. This would have been much harder with vanilla DFAs!
- This proof led us to a key result: DFAs, NFAs, and Regular Expressions are all equivalent models of computation.



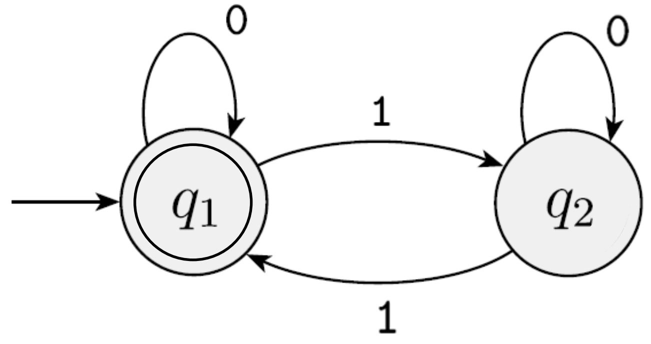
=



= (0|1)*1

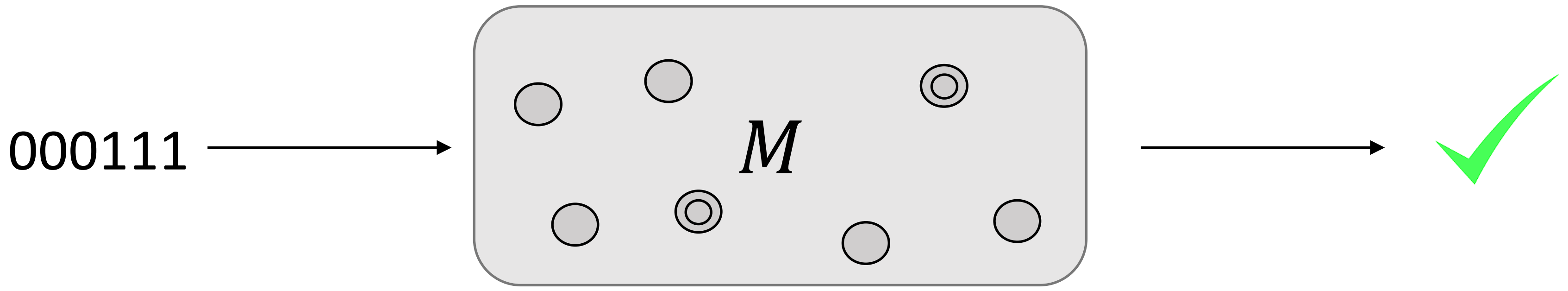
Exercise

- Convert the following DFA to a regular expression:


$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "0"] \arrow[l, bend left, "1"] \end{tikzcd}$$

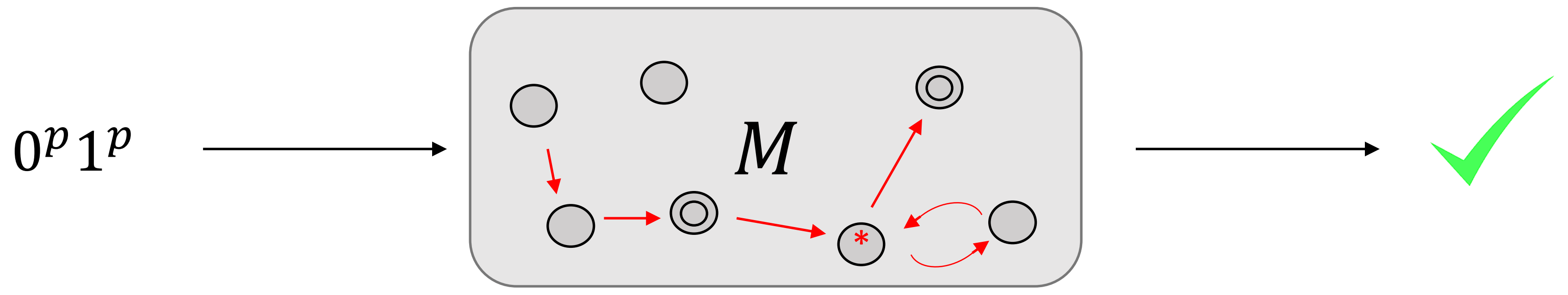
Proving Regularity and Non-Regularity

- Recall that a language is *regular* if it is recognized by some DFA.
- To prove that a language is regular, just describe a DFA, NFA, or Regular Expression that will recognize the language.
- How can we prove that a language is not regular?
- Example: Prove that the language $A = \{0^n 1^n \mid n \geq 0\}$ (i.e. language with n 0's followed by n 1's) is not regular.
- Proof by contradiction: Suppose we have some magic DFA called M that recognizes the language A, and let p be the number of states in M.



Example Contd.

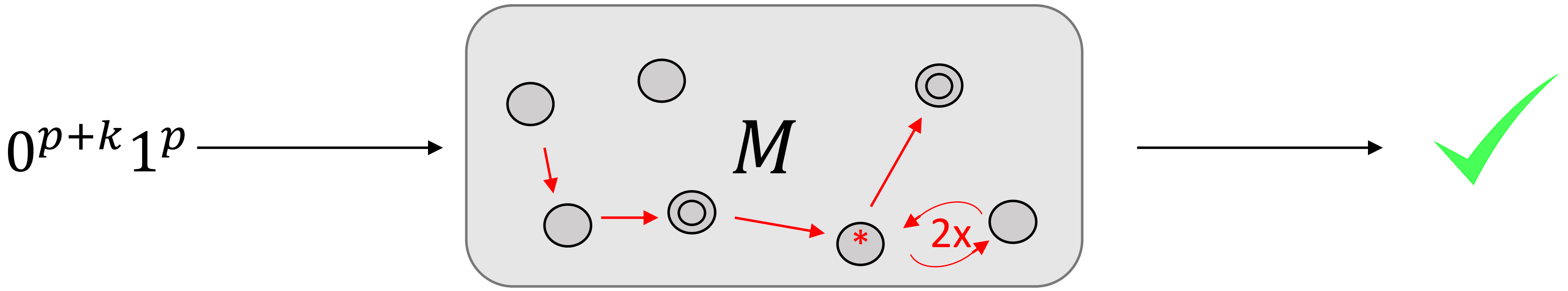
- Consider passing in the string $0^p 1^p$



- M only has p states, so it must have repeated a state a by the time it has consumed p symbols.
  - Otherwise it would have occupied p+1 unique states, including the start state
- However, the first p symbols are all 0. Therefore M must have visited some state *, processed a string of k 0's, and returned to the state * (where k is just a nonzero int).

Example Contd.

- If we add another k 0's to the first half of the input, we will just cause M to repeat this cycle twice, and then proceed exactly as before, ending in an accepting state



- However, $0^{p+k}1^p$ is not a member of the language A! Therefore M does not actually recognize A, which is a contradiction
- Thus A cannot be regular.

The Pumping Lemma

- The argument from the previous slide can be generalized to prove the Pumping Lemma:
- Suppose A is a regular language. Then there exists a number p such that for any $s \in A$ satisfying $|s| \geq p$, s can be broken into three parts $s = xyz$ that satisfy:
  - $xy^iz \in A$ for all $i \geq 0$
  - $|\mathrm{y}| > 0$
  - $|xy| \leq p$
- In the statement of the theorem, the segment y is analogous to the string of zeros we used to drive the DFA in a circle.

Section Problems

1. Give an NFA with three states that recognizes the set of strings ending in “00”.
2. Prove that the set of strings that are palindromes is not a regular language
3. Suppose A is a regular language.

   Let $f(w_1 w_2 \ldots w_k) = \begin{cases} 0 w_2 w_3 \ldots w_k & |w| > 0 \\ "" & |w| = 0 \end{cases}$

   (i.e. f replaces the first character of a string with 0, except the empty string)

   Show that $A' = \{f(w) | w \in A\}$ is regular.