CSE 311 Section 09

Models of Computation

Administrivia

Announcements & Reminders

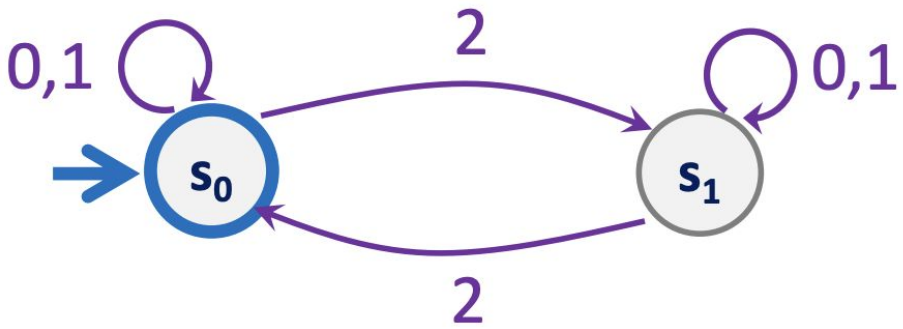
- HW8
  - Due Yesterday
- HW9
  - Due **Thursday,** June 4th @ **11:00 PM**
- check your section attendance on canvas
- **book one-on-ones!**

Deterministic Finite Automata

Deterministic Finite Automata

- A DFA is a finite-state machine that accepts or rejects a given string of symbols, by running through a state sequence uniquely determined by the string.

Strings with an even number of 2's



- An edge for every symbol in the language
- Every reject and accept state is handled

Task 1 – DFA Design

Construct DFAs to recognize each of the following languages.
Let $\Sigma = \{0, 1, 2, 3\}$.

b) All strings whose digits sum to an even number.

Work on this problem with the people around you.

Task 1 – DFA Design

Let Σ = {0, 1, 2, 3}.

b) All strings whose digits sum to an even number.

Hint: start with just the language of 1’s

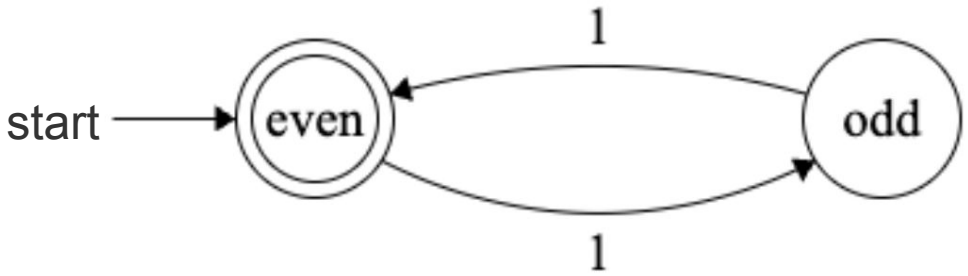
Task 1 – DFA Design

Let Σ = {0, 1, 2, 3}.

b) All strings whose digits sum to an even number.

Hint: start with just the language of 1's

If we had Σ = {1}

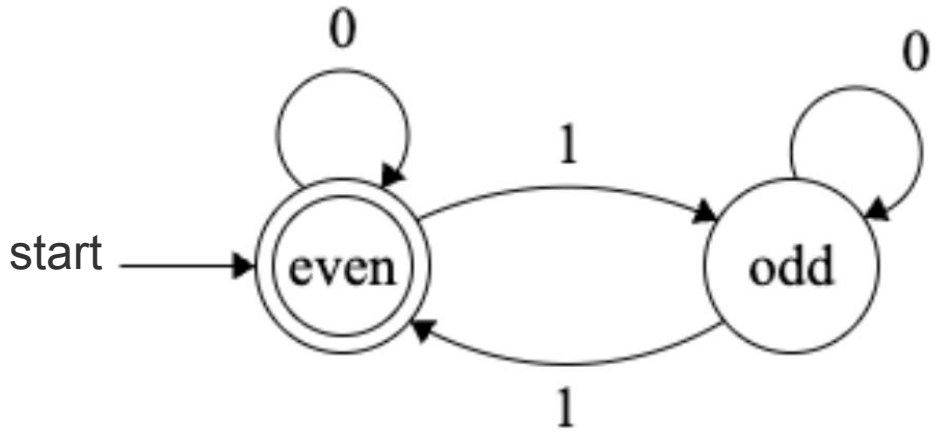


Task 1 – DFA Design

Let Σ = {0, 1, 2, 3}.

b) All strings whose digits sum to an even number.

If we had Σ = **{1,0}**

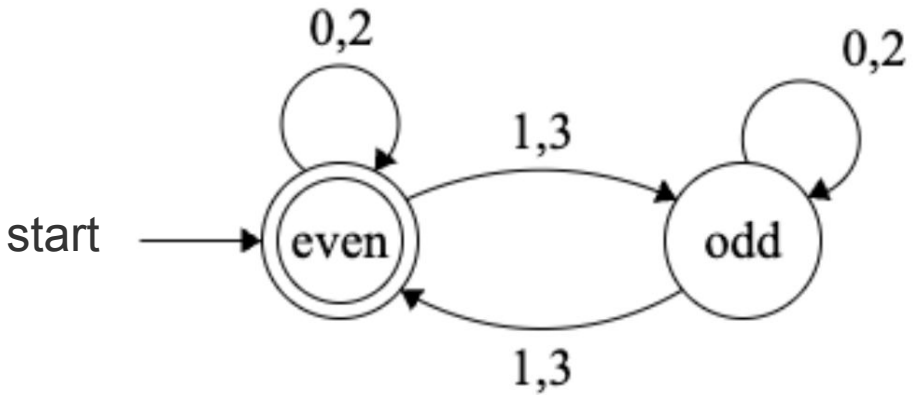


Task 1 – DFA Design

Let Σ = {0, 1, 2, 3}.

b) All strings whose digits sum to an even number.

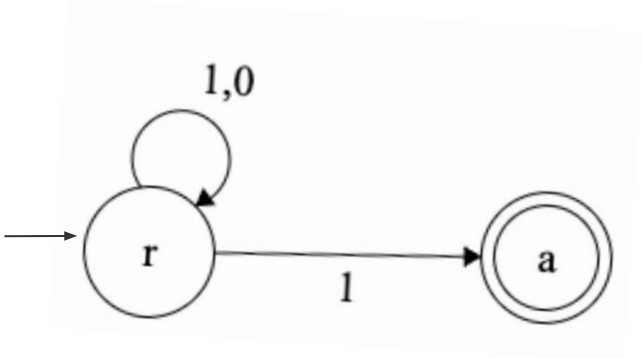
If we had Σ = **{0, 1, 2, 3}**



Nondeterministic Finite Automata

Nondeterministic Finite Automata

- **NFA in a nutshell** – A finite-state machine that explores all possibilities in parallel, it “knows” which path to take to get to an accept state. Specifies accept paths



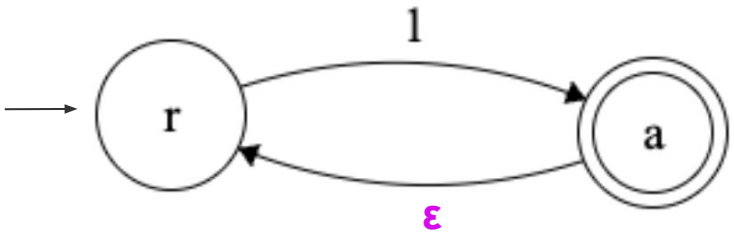
given two “1” edges, this will accept on 10101**1**

Nondeterministic Finite Automata

- **NFA in a nutshell** – A finite-state machine that explores all possibilities in parallel, it "knows" which path to take to get to an accept state. Specifies accept paths

- **ε-moves** – Special "free" hops that consume no input, letting the machine reposition before reading more symbols.

  think of strings will implicit **ε:**
  "11" as 1**ε1**

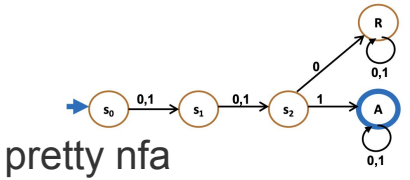


Nondeterministic Finite Automata

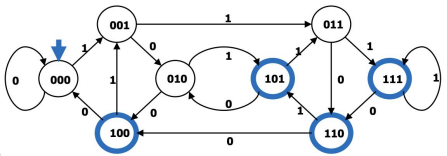
- **NFA in a nutshell** – A finite-state machine that explores all possibilities in parallel, it "knows" which path to take to get to an accept state. Specifies accept paths

- **ε-moves** – Special "free" hops that consume no input, letting the machine reposition before reading more symbols.

- **Key difference from a DFA** –
  - An NFA may have many (or zero) edges with the same label from one state, plus ε-edges
  - DFA has **exactly one per symbol** and **no ε-edges**
  - Makes for nicer looking machines!



pretty nfa



ugly dfa

Task 2 – NFAs

c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”

Work on this problem with the people around you.

Task 2 – NFAs

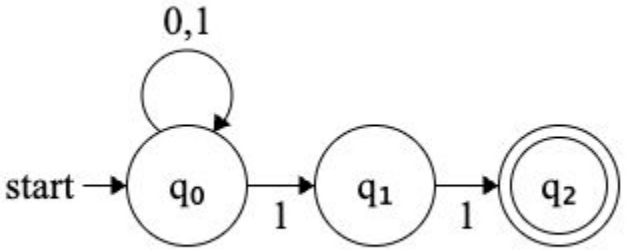
c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”

Break it into steps!
How can we make an NFA that ends in a 11?

Work on this problem with the people around you.

Task 2 – NFAs

c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”



Break it into steps!
How can we make an NFA that ends in a 11?

Work on this problem with the people around you.

Task 2 – NFAs

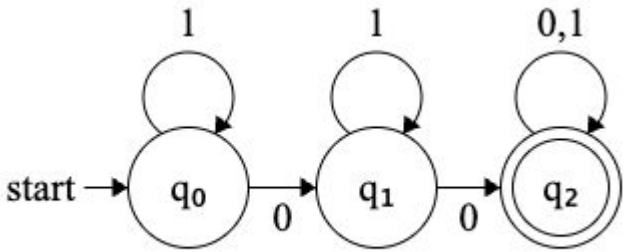
c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”

Break it into steps!
How can we make an NFA that has at least two 0’s?

Work on this problem with the people around you.

Task 2 – NFAs

c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”

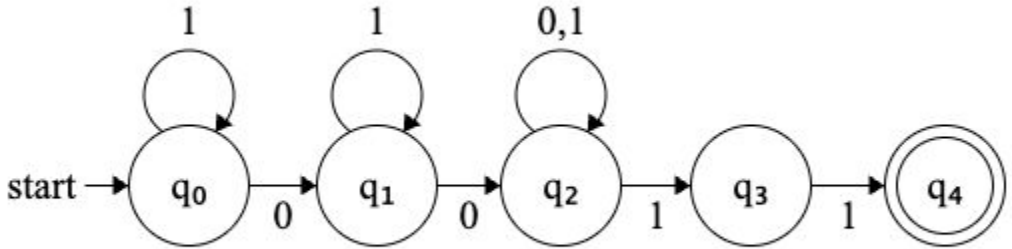


Break it into steps!
How can we make an NFA that has at least two 0’s?

Work on this problem with the people around you.

Task 2 – NFAs

c) Create an NFA for the language “all binary strings that have at least two 0’s and ends in a 11.”



Work on this problem with the people around you.

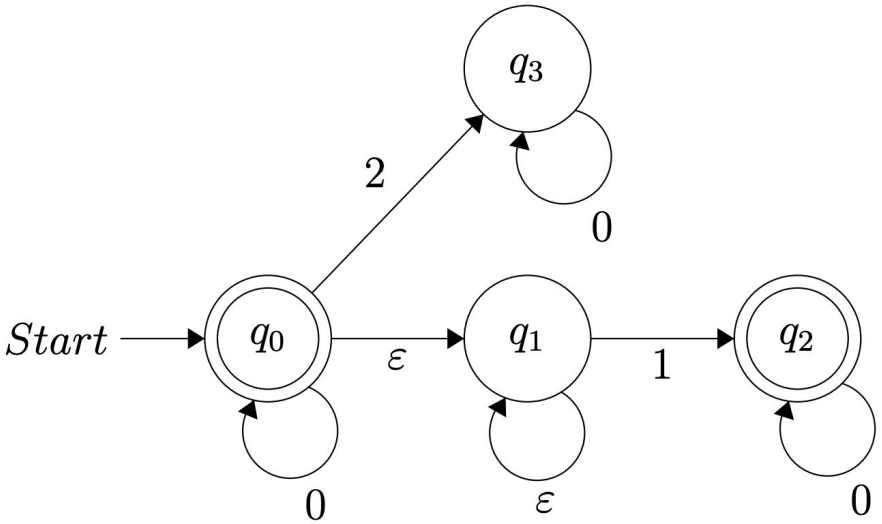
NFA to DFA

DFAs and NFAs

- Any language that can be accepted by a DFA can be accepted by an NFA
- A DFA is an NFA!
- Every language that can be accepted by an NFA can be accepted by a DFA also!
- Using the following algorithm:
  - Start at the set of states which can be reached from the empty state
  - Construct a table with all symbols of the ∑, and what each symbol takes the set of states to
  - Keep going until there are no more unique states

Task 4

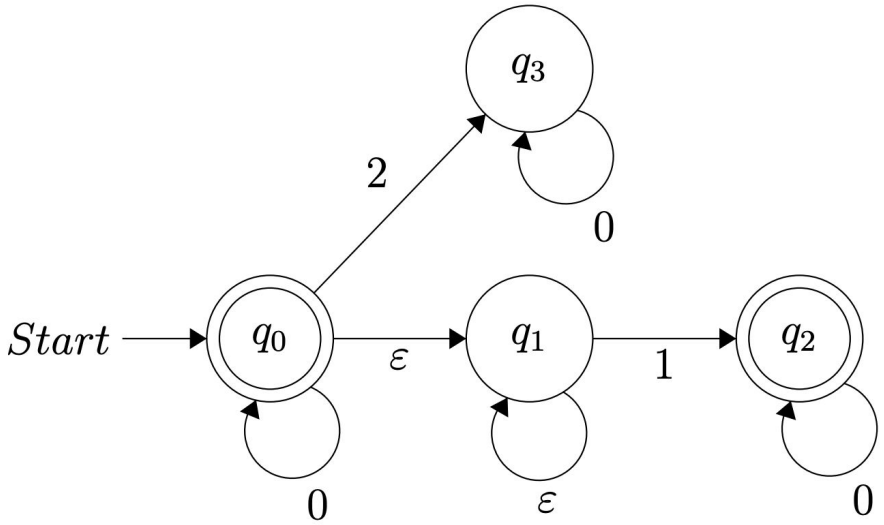
Convert the following NFA to a DFA of the same language



Task 4 - Documenting states in a table

Where can we get from the start state (q0) without reading input?

(DFA) state	0-transition	1-transition	2-transition

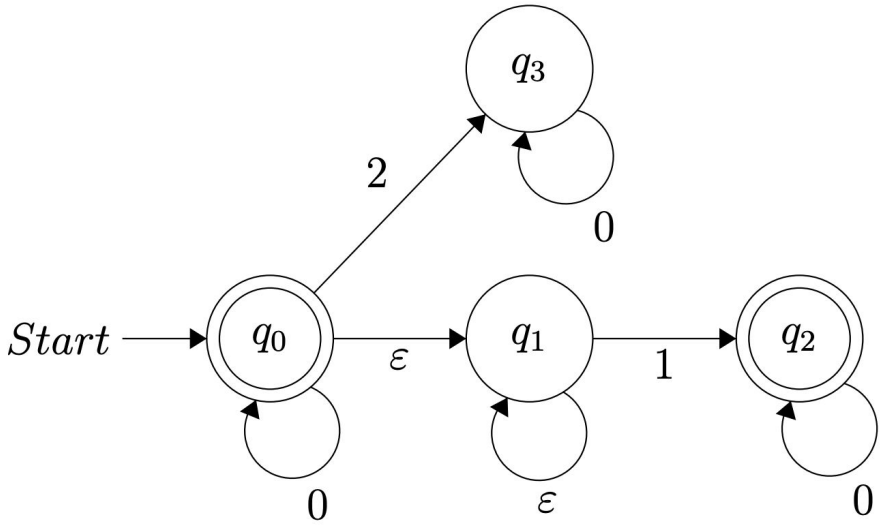


Task 4 - Documenting states in a table

Where can we get from the start state (q0) without reading input?

- q0
- take ε to q1

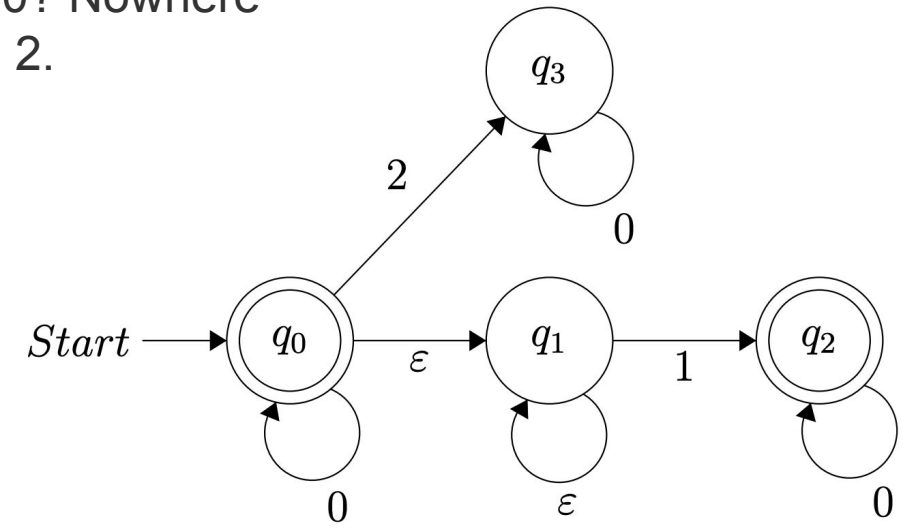
(DFA) state	0-transition	1-transition	2-transition
q0, q1			



Task 4 - Documenting states in a table

From q0, where can we get reading in 0? q0 and q1
From q1, where can we get reading in 0? Nowhere
Repeat for reading in 1, and reading in 2.
Then fill in the state table.

(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3

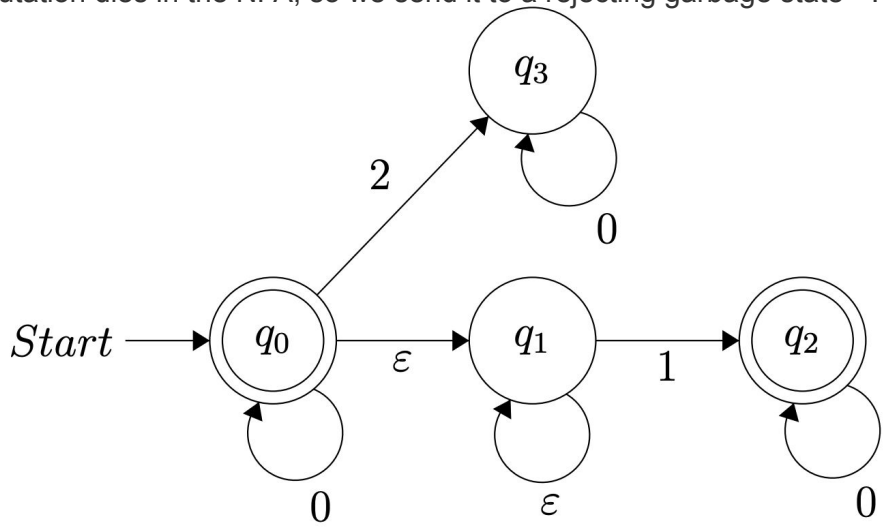


Task 4 - Documenting states in a table

We need to define the other states (q2, q3) in the table now.

- When there's no defined transition for an input, the computation dies in the NFA, so we send it to a rejecting garbage state ∅.

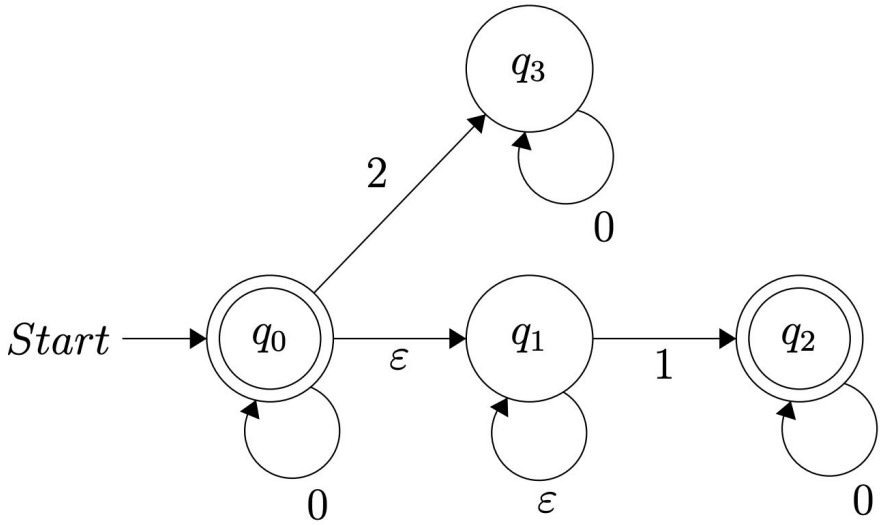
(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3			



Task 4 - Documenting states in a table

We need to define the other states (q2 and q3) in the table now.

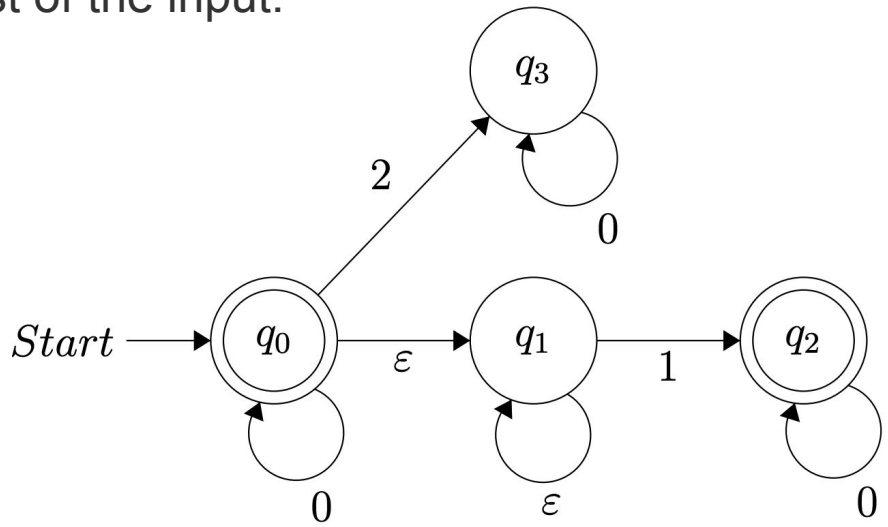
(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3	q3	∅	∅



Task 4 - Documenting states in a table

The only state we have left to define is the “garbage” state. This one always is rejecting regardless of the rest of the input.

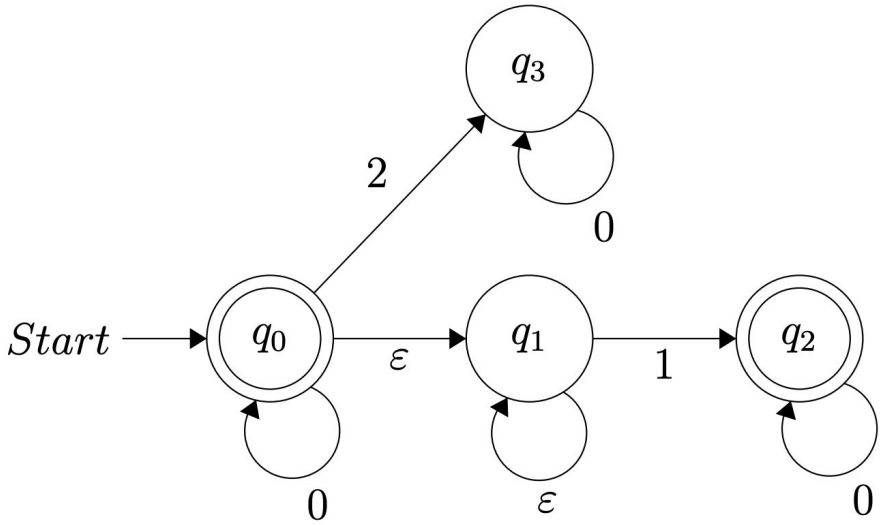
(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3	q3	∅	∅
∅	∅	∅	∅



Task 4 - Documenting states in a table

Which states are accepting? (The rest are rejecting)

(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3	q3	∅	∅
∅	∅	∅	∅

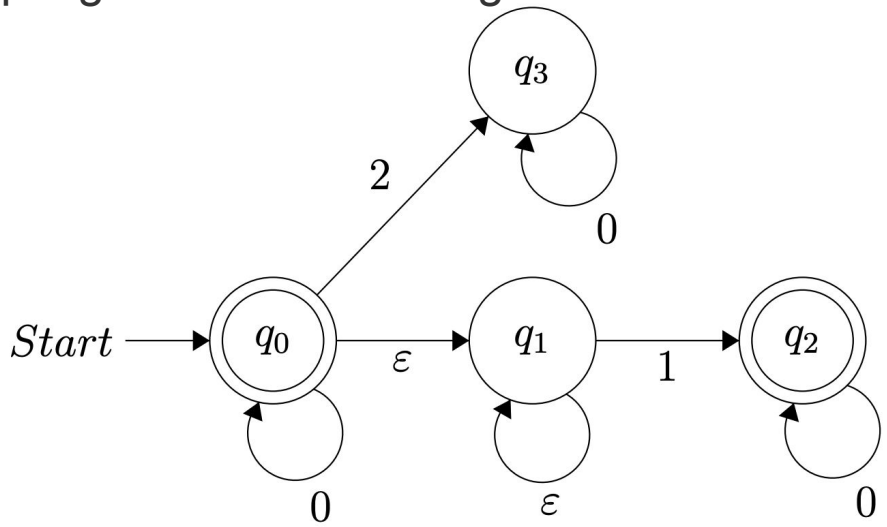


Task 4 - Documenting states in a table

Which states are accepting? (The rest are rejecting)

- All DFA states that include an accepting state from the original NFA.

(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3	q3	∅	∅
∅	∅	∅	∅

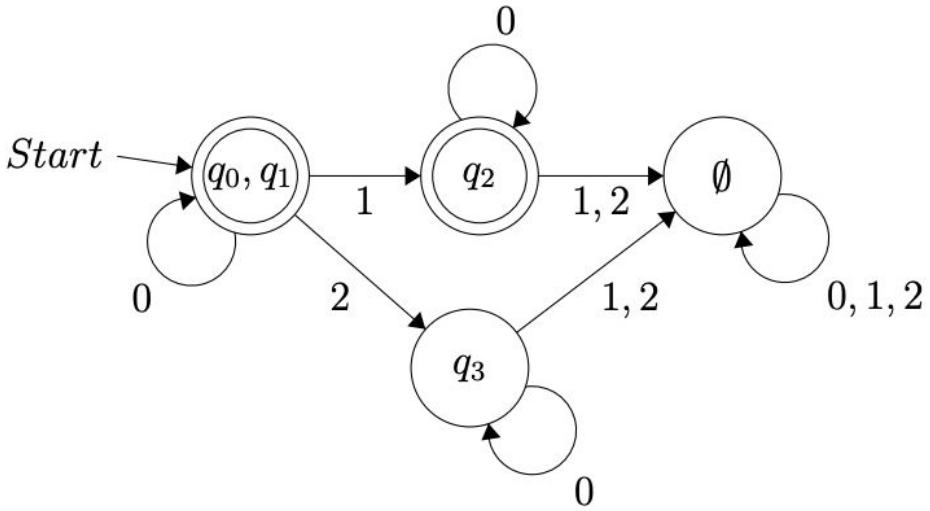


Now try drawing out the DFA using the state table.

- Don’t forget a start state!

Task 4

(DFA) state	0-transition	1-transition	2-transition
q0, q1	q0, q1	q2	q3
q2	q2	∅	∅
q3	q3	∅	∅
∅	∅	∅	∅



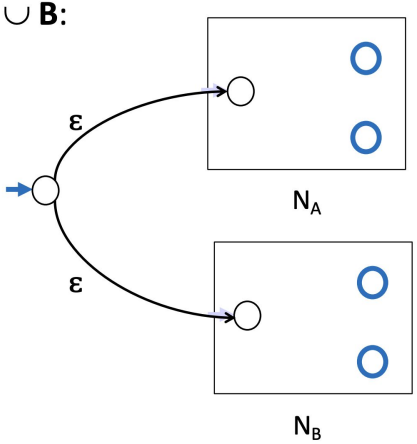
That's All, Folks!

Thanks for coming to section this week!
Any questions?

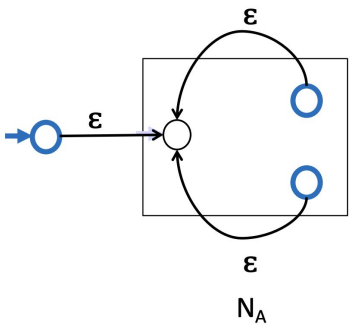
Extra Time: Regex to NFA

Regex to NFA Conversion!

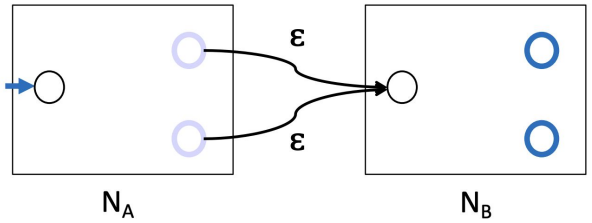
Case A ∪ B:



Case A*



Case AB:



Task 3: RE to NFA

a) Convert the regular expression "$(11 \cup (01)^*)00$" to an NFA using the algorithm from lecture. You may skip adding ε-transitions for concatenation if they are obviously unnecessary, but otherwise, you should *precisely* follow the construction from lecture.

Task 3: RE to NFA

a) Convert the regular expression "$(11 \cup (01)^*)00$" to an NFA using the algorithm from lecture. You may skip adding ε-transitions for concatenation if they are obviously unnecessary, but otherwise, you should *precisely* follow the construction from lecture.