Theory of Computation

BCS1110

Dr. Ashish Sai

🗓️ TOC - Lecture 2

💻 bcs1110.ashish.nl

Plan for Today

- Recap from TOC Lecture 1
- Tabular DFAs
- Regular Languages
- NFAs
- Designing NFAs
- (*if time permits*) Tutorial Questions

Recap From Last Time

Old MacDonald Had a Symbol, 🎶 Σ-eye-ε-ey∈, Oh! 🎶

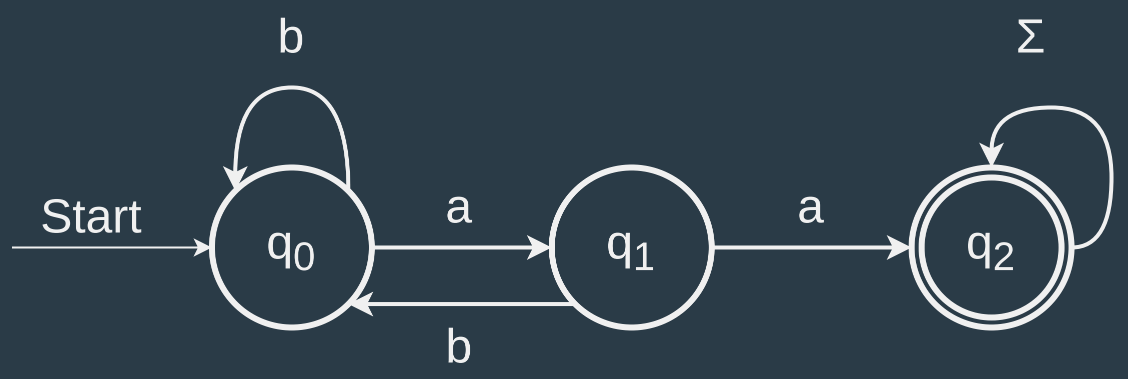
- Here's a quick guide to remembering which is which:
  - Typically, we use the symbol **Σ** (sigma) to refer to an *alphabet*
  - The *empty string* is length 0 and is denoted **ε** (epsilon)
  - In set theory, use **∈** to say "is an *element of*"
  - In set theory, use **⊆** to say "is a *subset of*"

DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton

Recognizing Languages with DFAs

L = { w ∈ {a,b} * | w contains aa as a substring }



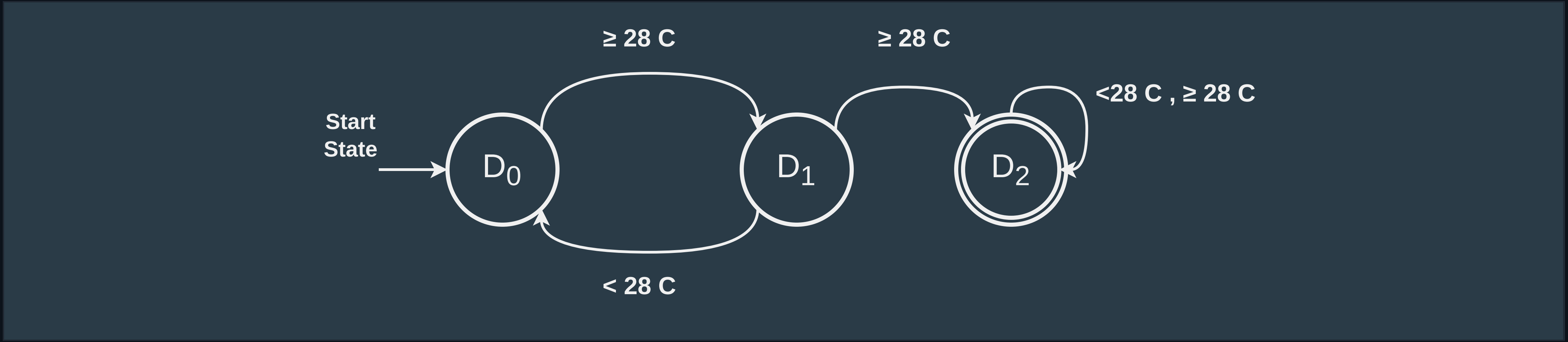
DFAs

- A DFA is defined relative to some alphabet Σ (sigma)
- For each state in the DFA, there must be **exactly one** transition defined for each symbol in Σ
    - This is the “deterministic” part of DFA
- There is a unique start state
- There are zero or more accepting states

Tabular DFAs

Part 1/4

Deterministic Finite Automaton (Formal Definition)



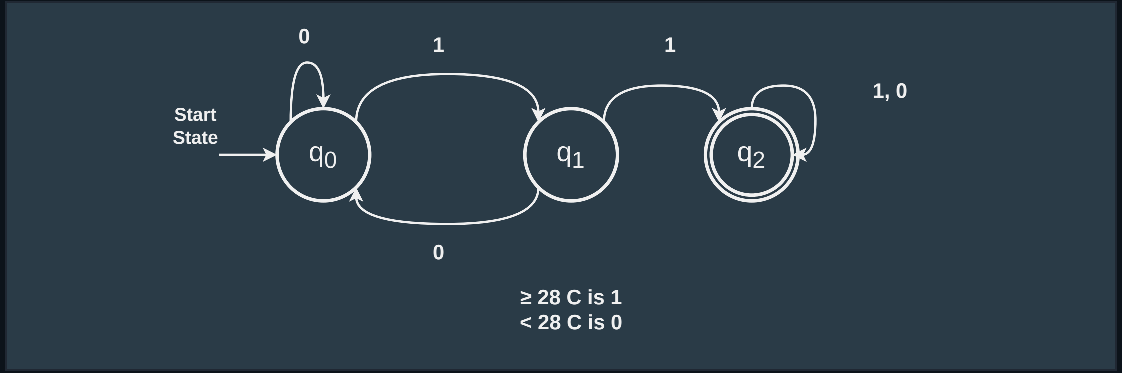
- Input: String of weather data
- 🇬🇧 Heatwave: temperature ≥ 28 C for **2 consecutive days**

DFA Definition

D = (Q, Σ, δ, q_0, F)

- Q is the set of states [Q = { q_0, q_1, q_2 }]
- Σ is the alphabet [Σ = {1,0}]
- δ is the transition function
- q_0 is the start state
- F is an accepting state [F = { q_3 }]

Transition Function:

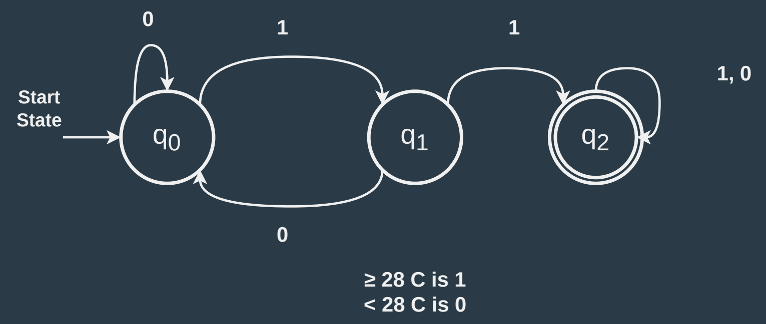


DFA Definition

D = (Q, Σ, δ, q_0, F)

- Q is the set of states [Q = { q_0, q_1, q_2 }]
- Σ is the alphabet [Σ = {1,0}]
- δ is the transition function
- q_0 is the start state
- F is an accepting state [F = { q_3 }]

Transition Function:



	1	0
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_2	q_2

Which table best represents the transitions for the following DFA?

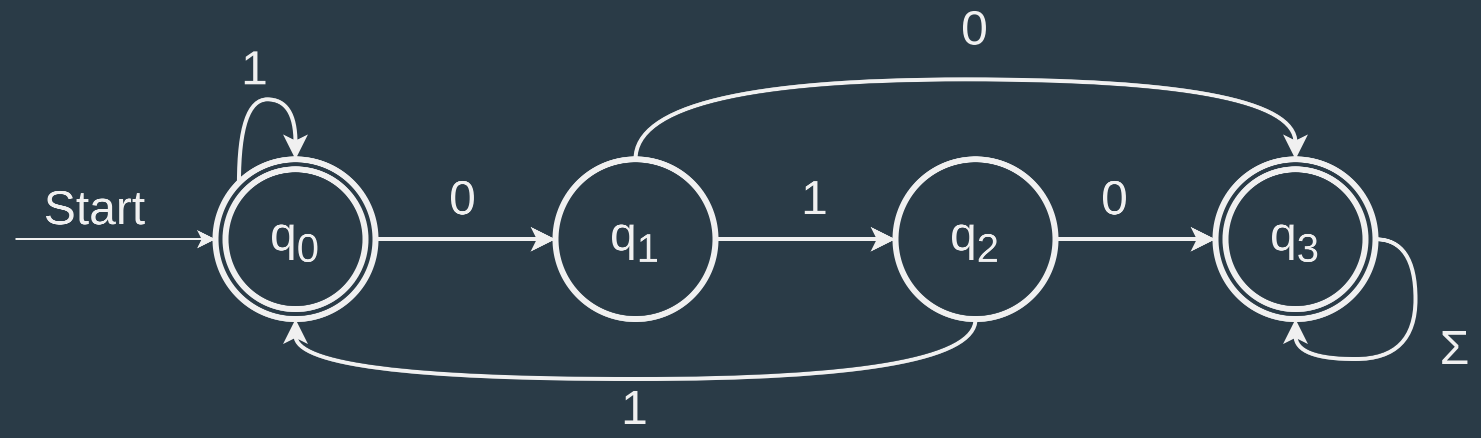


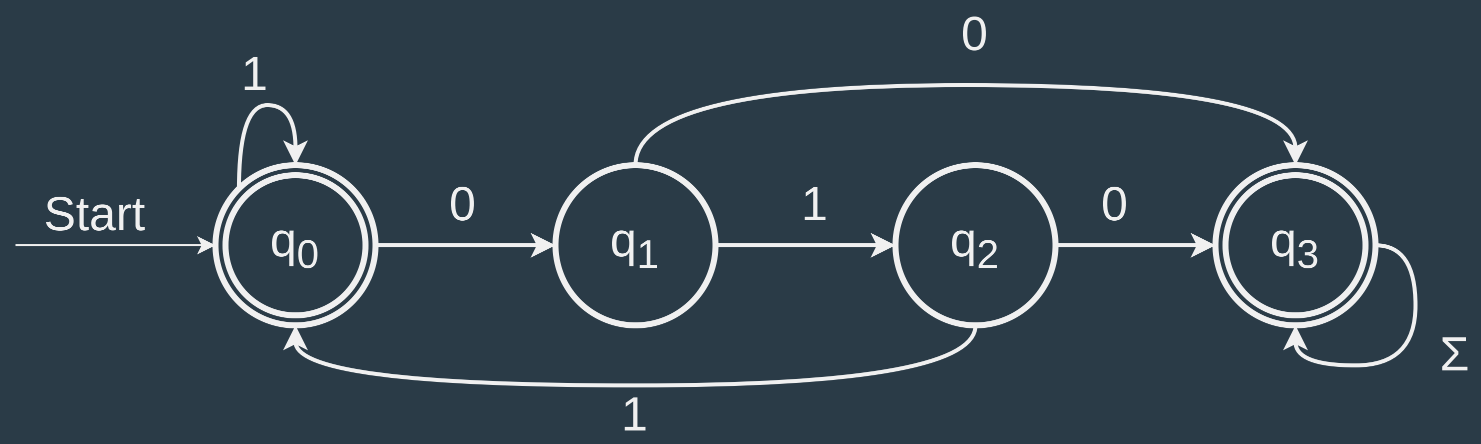
Table A

	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

Table B

	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_3
q_3	q_3	q_3

Tabular DFAs



	0	**1**
$^*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$^*q_3$	q_3	q_3

- Stars indicate accepting states
- First row is the start state

Code Demo


When I wrote this code,
only god & I understood what it did.
Now... only god knows.

```java
public class DFASimulator {

  private static final int kNumStates = 4; // 4 states based on the table
  private static final int kNumSymbols = 2; // 2 symbols (0 and 1) based on the table

  private static final int[][] kTransitionTable = {
    {1, 0},
    {3, 2},
    {3, 0},
    {3, 3}
  };

  private static final boolean[] kAcceptTable = {
    true,
    false,
    false,
    true
  };

  public static boolean simulateDFA(String input) {
    int state = 0;
    char[] inputArray = input.toCharArray();
    for (int i = 0; i < inputArray.length; i++) {
      char ch = inputArray[i];
      if (ch != '0' && ch != '1') {
        throw new IllegalArgumentException("Invalid input symbol: " + ch);
      }
      state = kTransitionTable[state][ch - '0'];
    }
    return kAcceptTable[state];
  }

  public static void main(String[] args) {
    String testInput = "1011"; // Example input
    boolean isAccepted = simulateDFA(testInput);
    System.out.println("The input " + testInput + " is " +
      (isAccepted ? "accepted" : "rejected") + " by the DFA.");
  }
}
```

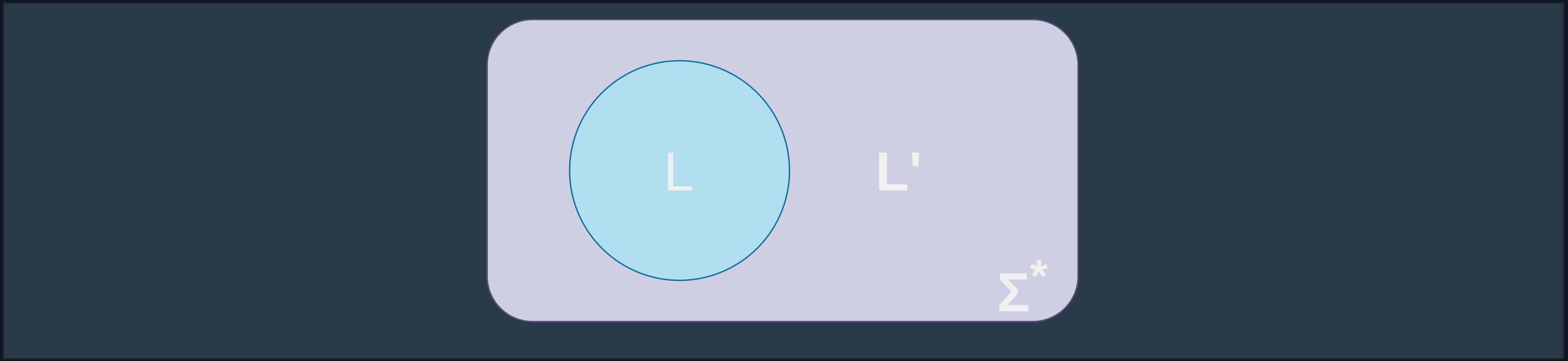
The Regular Languages

Part 2/4

- A language L is called a **regular language** if there exists a *DFA D* such that $L(D) = L$
- If L is a language and *L(D) = L*, we say that D **recognises** the language L

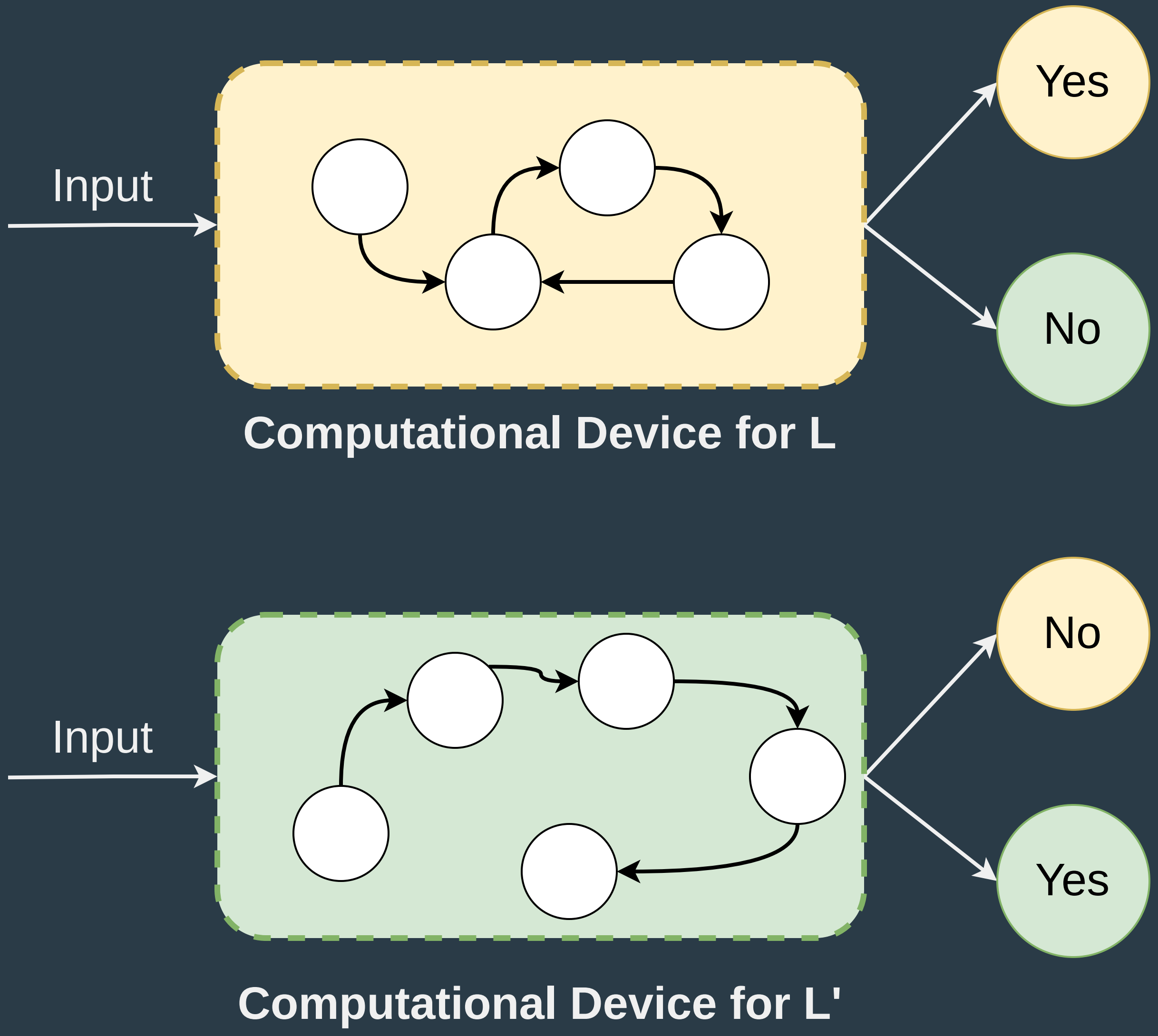
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted **L′**) is the language of all strings in Σ^* that aren't in L
- Formally: $L' = \Sigma^* - L$



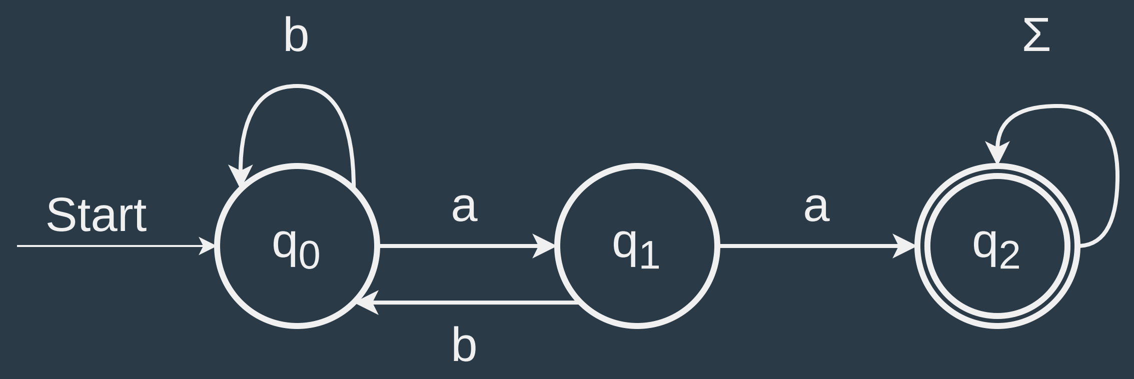
Complements of Regular Languages

- A **regular language** is a language accepted by some DFA
- **Question**: If L is a regular language, is L' necessarily a regular language?
- If yes → if there is a DFA for L, there must be a DFA for L′
- If no → some L can be accepted by a DFA, but L' cannot

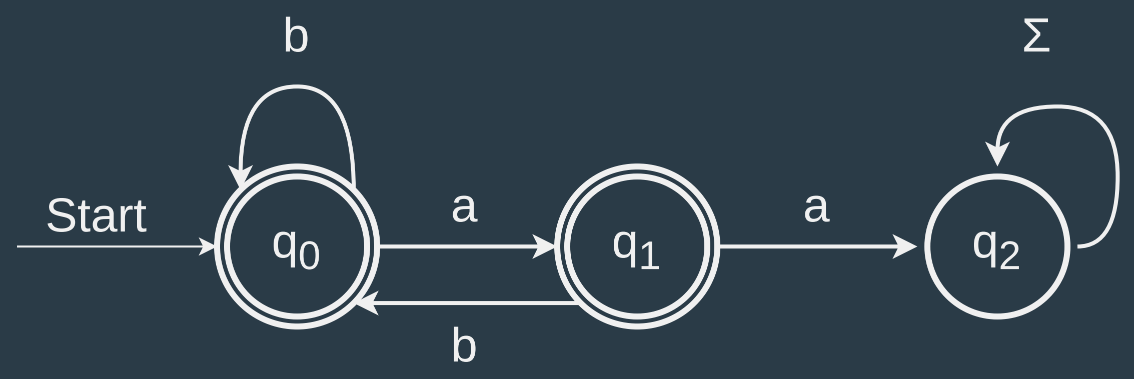

Input
Yes
No
Computational Device for L
Input
No
Yes
Computational Device for L'

Complementing Regular Languages

L = { w ∈ {a,b}* | w contains aa as a substring }



L′ = { w ∈ {a,b}* | w **does not** contain aa as a substring }

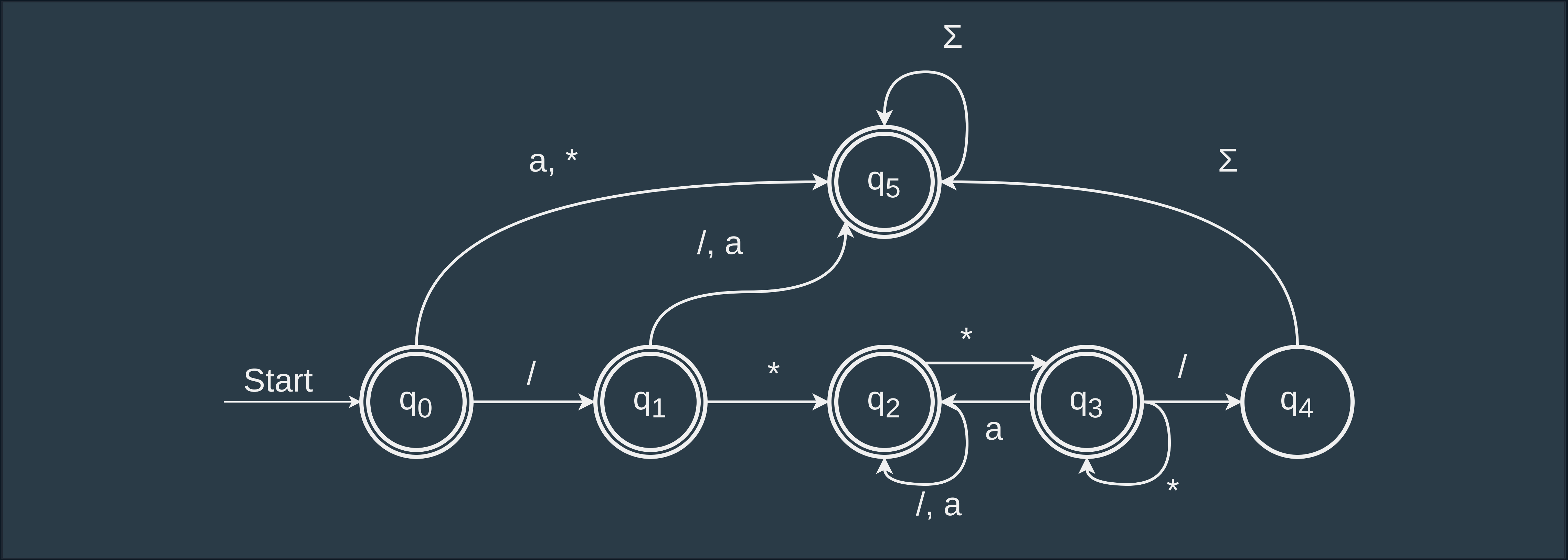


More Elaborate DFAs

L = { w ∈ {a,*,/} | w represents a (**multi-line**) Java-style comment }

More Elaborate DFAs

L′ = { w ∈ {a,*,/} | w doesn't represent a (**multi-line**) Java-style comment }



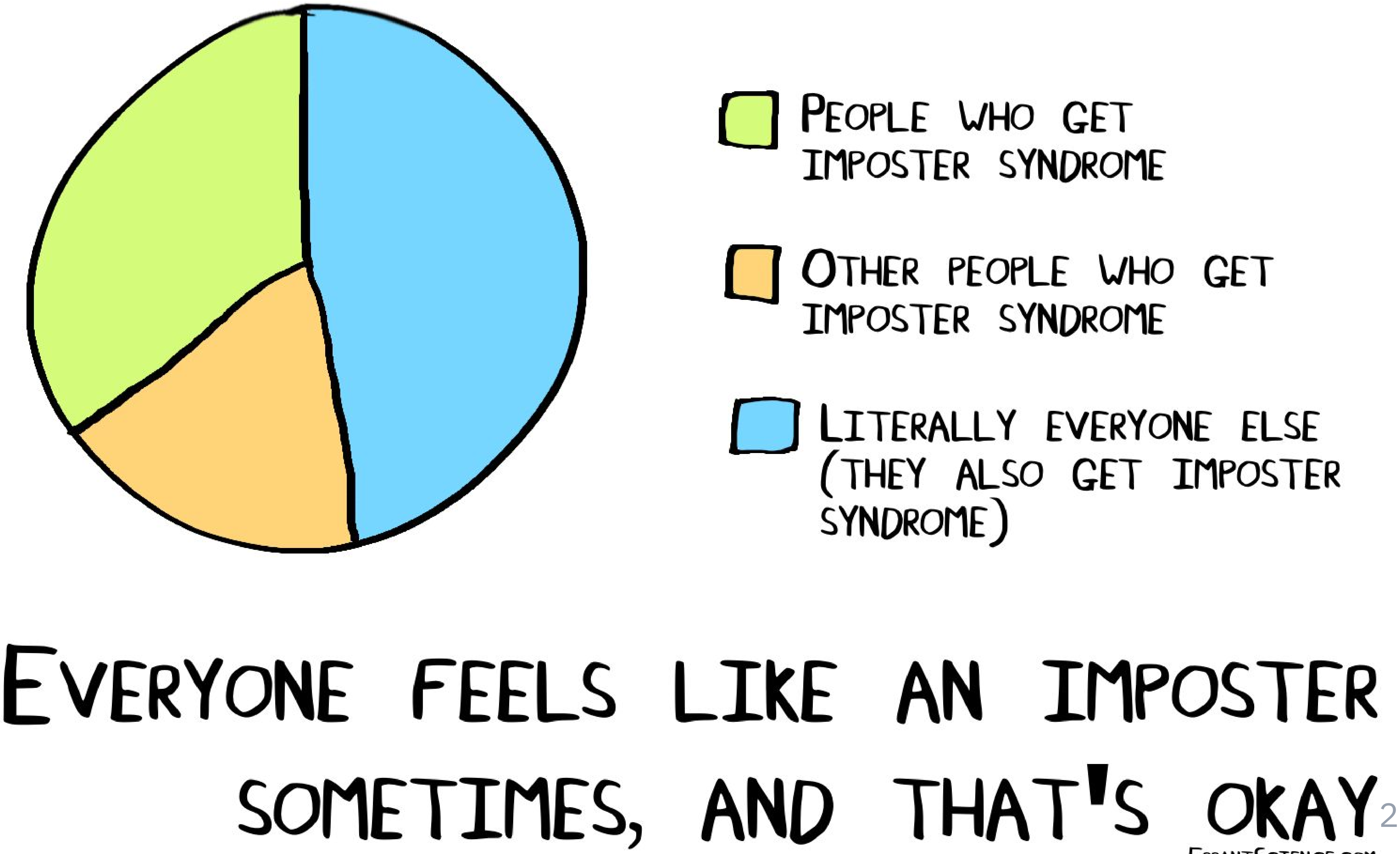
Closure Properties

- **Theorem**: If L is a regular language, then L' is also a regular language
- Thus, regular languages are **closed under complementation**

Time Out

(Not A Break)

Ever felt you weren't good enough to be in STEM? Afraid of being "found out" because you don't think you belong?


PEOPLE WHO GET IMPOSTER SYNDROME
OTHER PEOPLE WHO GET IMPOSTER SYNDROME
LITERALLY EVERYONE ELSE (THEY ALSO GET IMPOSTER SYNDROME)
EVERYONE FEELS LIKE AN IMPOSTER SOMETIMES, AND THAT'S OKAY

NFAs

Part 3/4

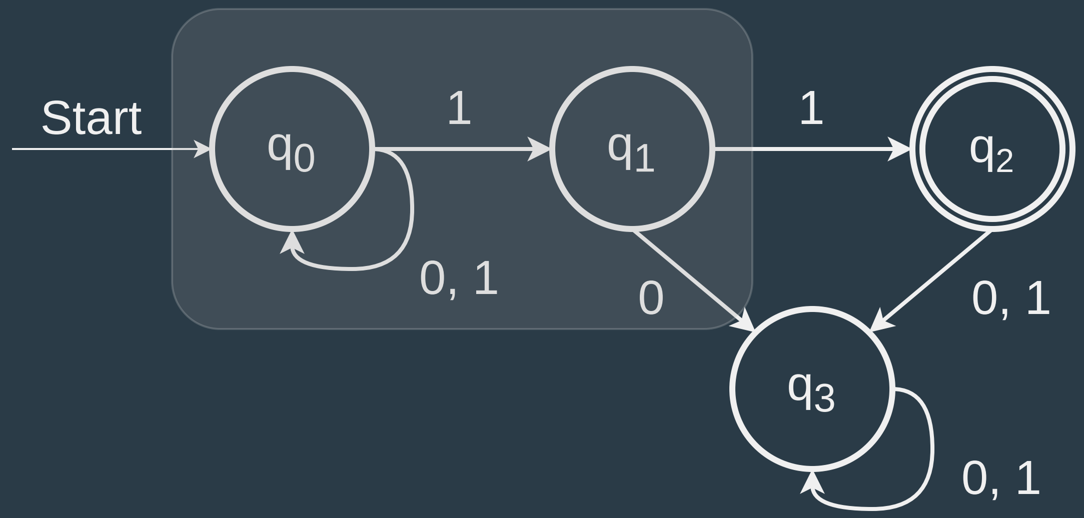
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation

(Non)determinism

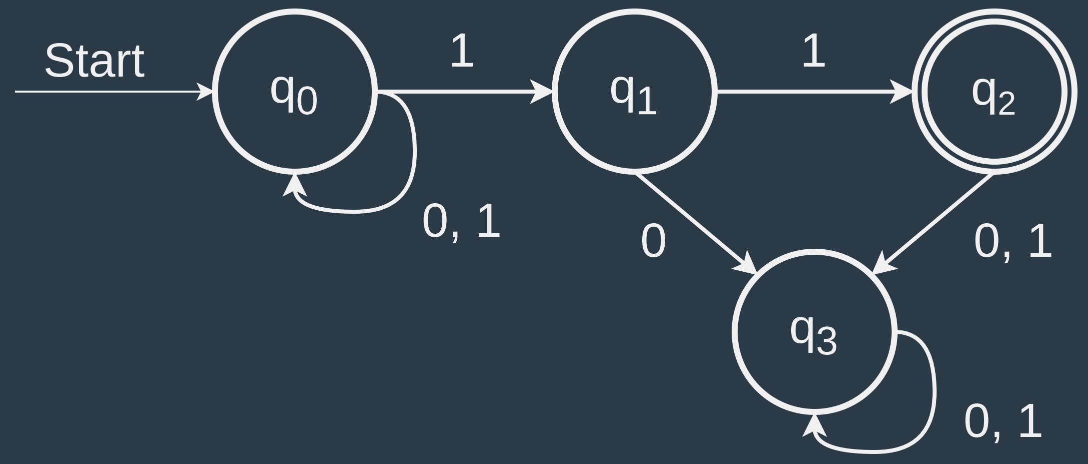
- **Deterministic**: at every point, exactly one choice
  - Accepts if that sequence leads to an accepting state
- **Nondeterministic**: machine may have multiple possible moves
  - Accepts if **any** path leads to an accepting state

A Simple NFA



> q_0 has two transitions defined on 1!

A Simple NFA



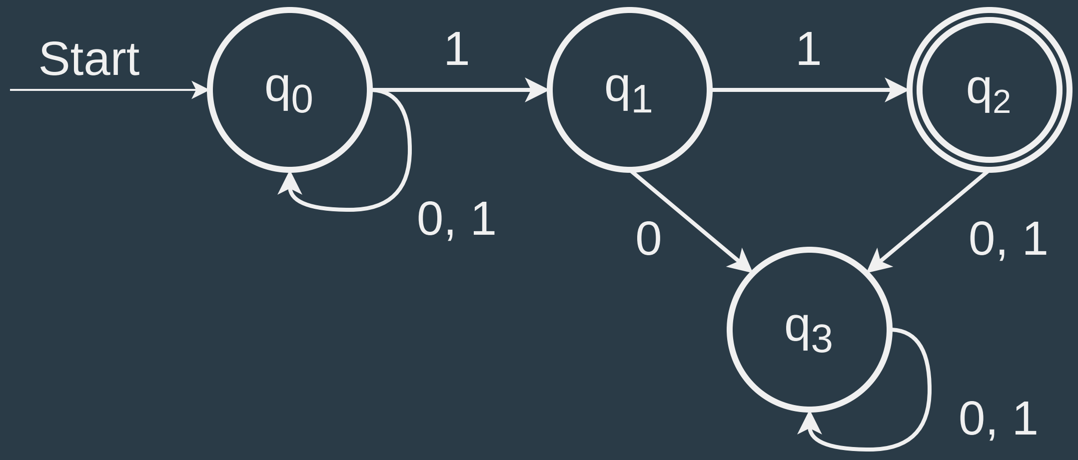
Input: 01011

Non-Deterministic Finite Automaton (Formal Definition)

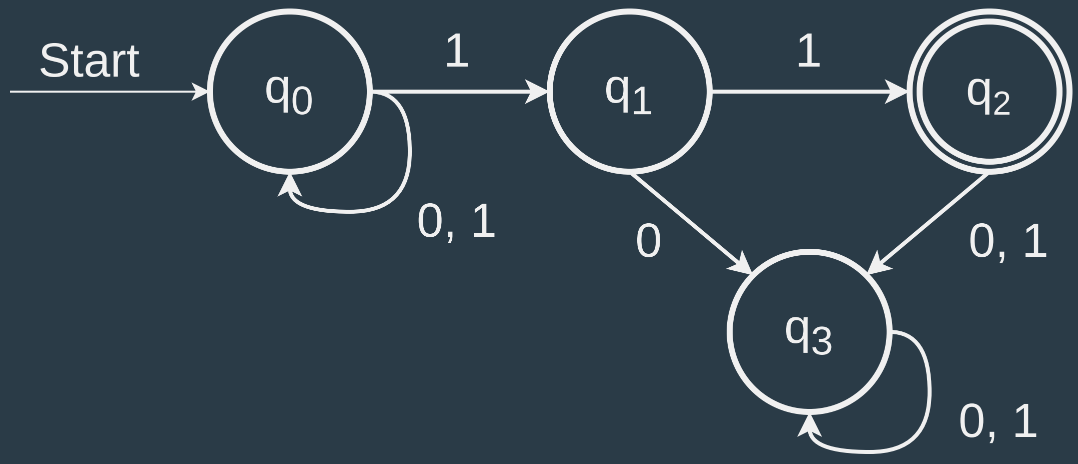
D = (Q, Σ, δ, q_0, F)

- Q = { q_0, q_1, q_2, q_3 }
- Σ = {0,1}
- δ = transition function
- q_0 = start state
- F = { q_2 }

A Simple NFA: Transition Function

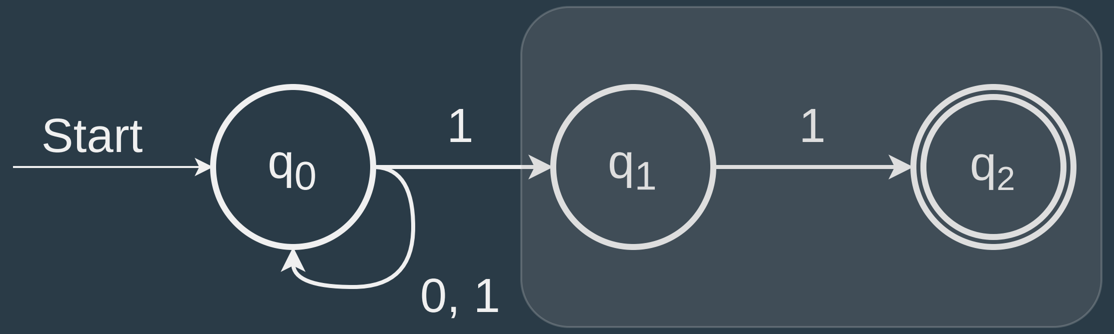


A Simple NFA: Transition Function

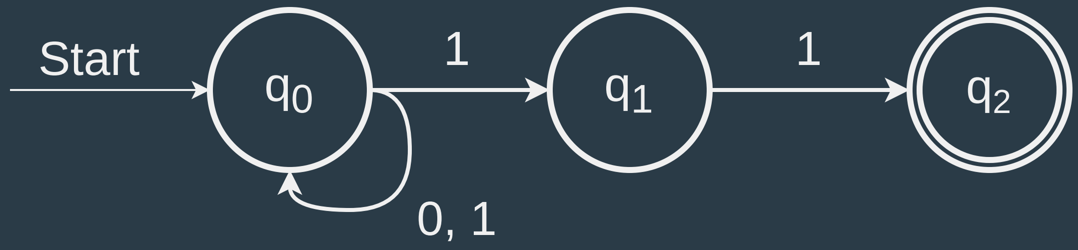


State	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_3\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_3\}$

A More Complex NFA



> If an NFA needs to make a transition when none exists, that path dies and does not accept



As with DFAs:

L(N) = { w ∈ Σ * | N accepts w }

What is the language of the NFA above?

- A) {01011}
- B) { w ∈ {0,1} * | w contains at least two 1s }
- C) { w ∈ {0,1} * | w ends with 11 }
- D) { w ∈ {0,1} * | w ends with 1 }
- E) None of these, or two or more of these

- **Answer: A and C → so E**

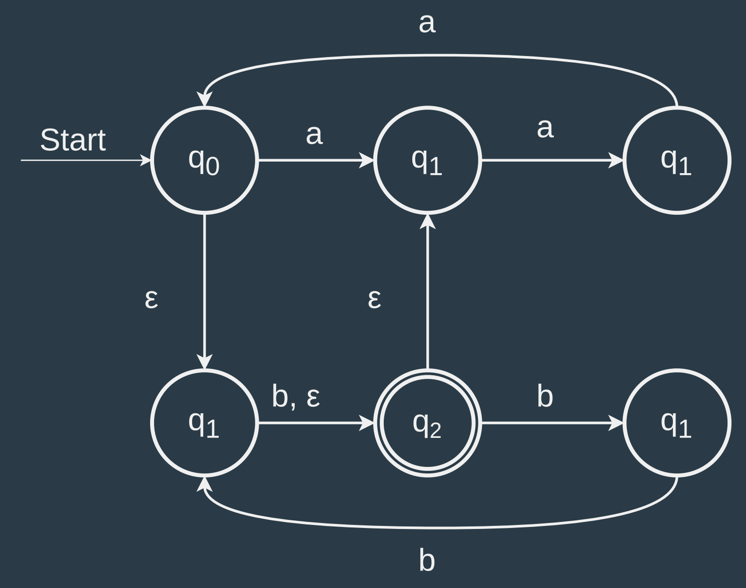
NFA Acceptance

- N accepts *w* if some path reaches an accepting state
- N rejects *w* if no path does
- Easier to prove acceptance than rejection

ε-Transitions

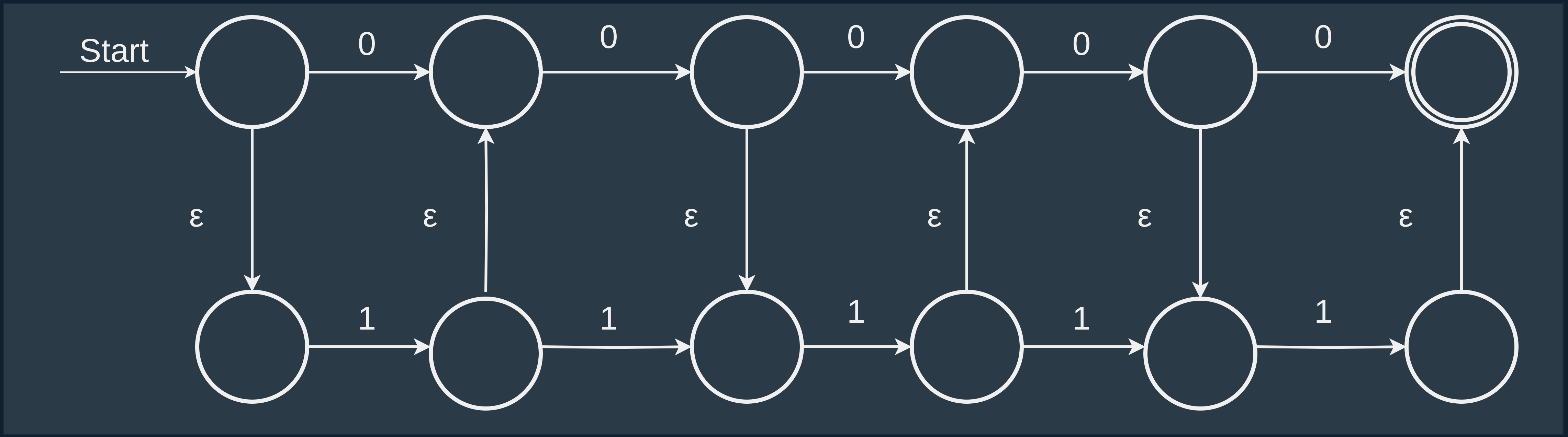
- NFAs may follow **ε-transitions** (no input consumed)
- May follow any number at any time

Input: b a a b b



ε-Transitions

- NFAs are **not required** to follow ε-transitions



Suppose we run on input 10110. Which are true?

- There is at least one accepting computation
- There is at least one rejecting computation
- There is at least one dead computation
- NFA accepts 10110
- NFA rejects 10110

Designing NFAs

Part 4/4

Designing NFAs

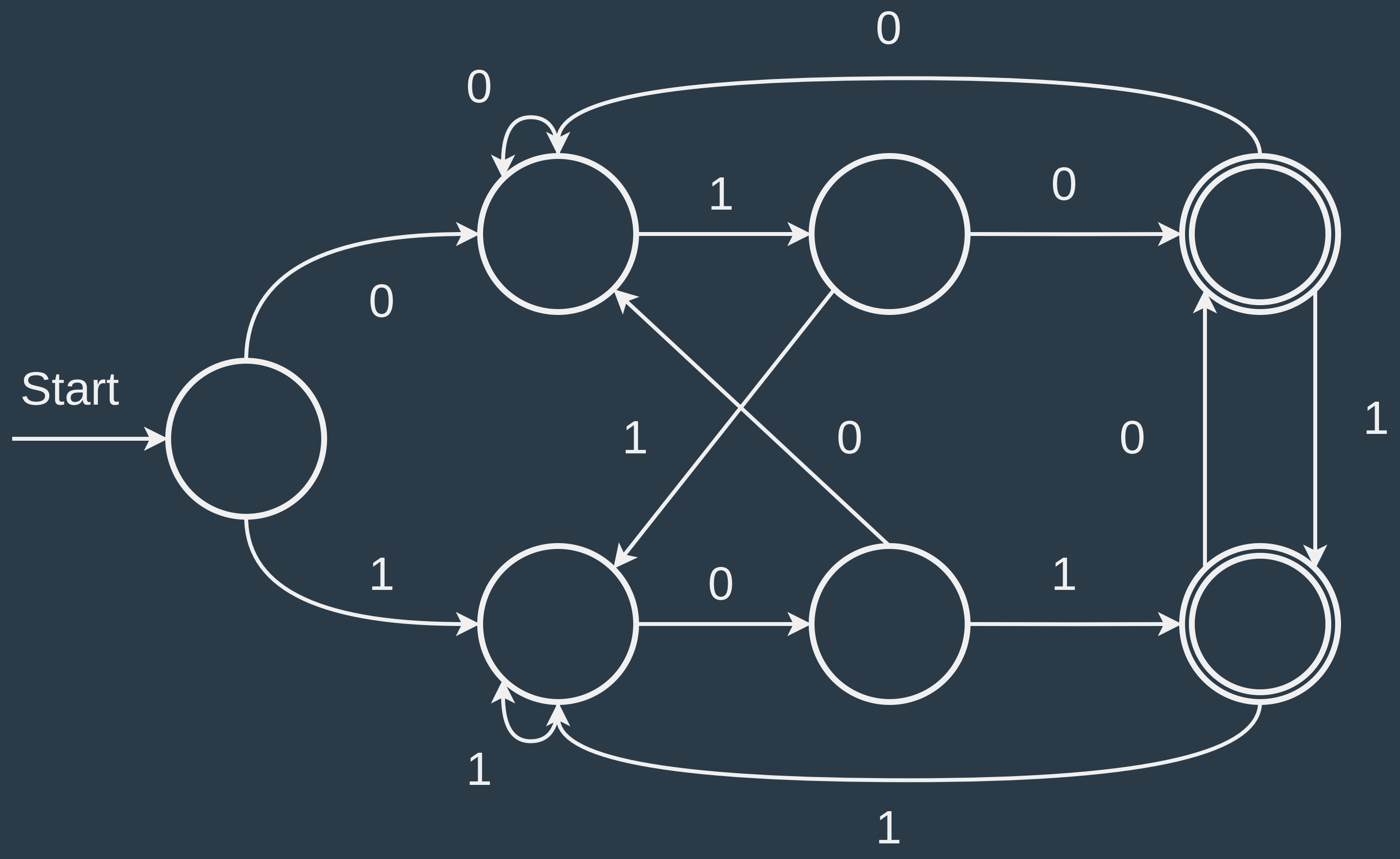
- Embrace nondeterminism
- **Guess-and-check** model:
  - Nondeterministically guess information
  - Deterministically check correctness

Guess-and-Check

$L = \{ w \in \{0,1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$

Guess-and-Check

L = { w ∈ {0,1} * | w ends in 010 or 101 }



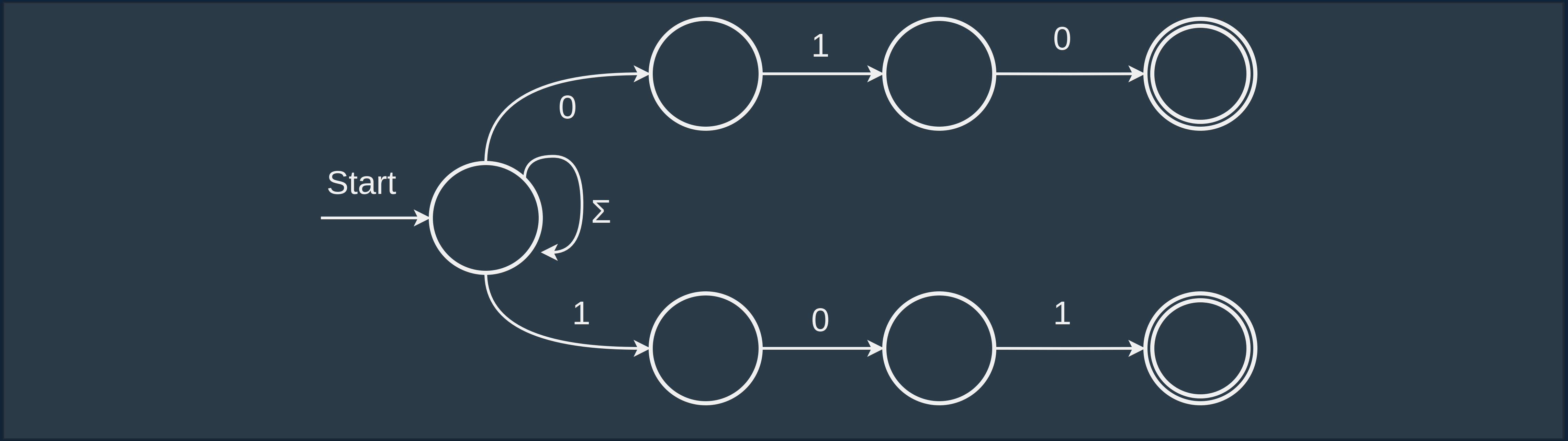
Guess-and-Check

$L = \{ w \in \{0,1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$

- Nondeterministically guess when to leave start
- Deterministically check correctness

Guess-and-Check

L = { w ∈ {0,1} * | w ends in 010 or 101 }



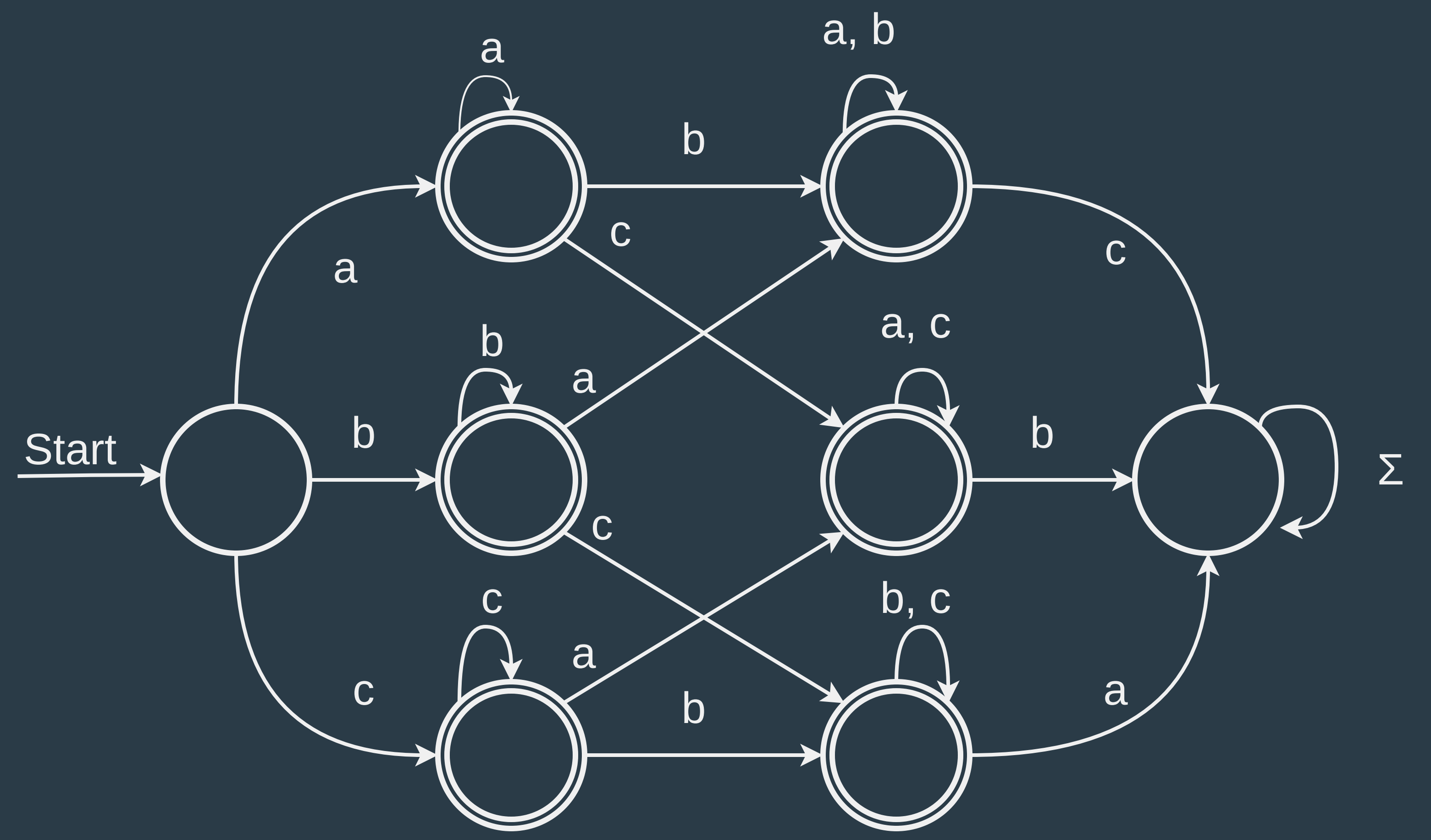
- Nondeterministically guess when to leave start
- Deterministically check correctness

Guess-and-Check

L = { w ∈ {a,b,c} * | at least one of a, b, or c is not in w }

Guess-and-Check

L = { w ∈ {a,b,c} * | at least one of a, b, or c is not in w }

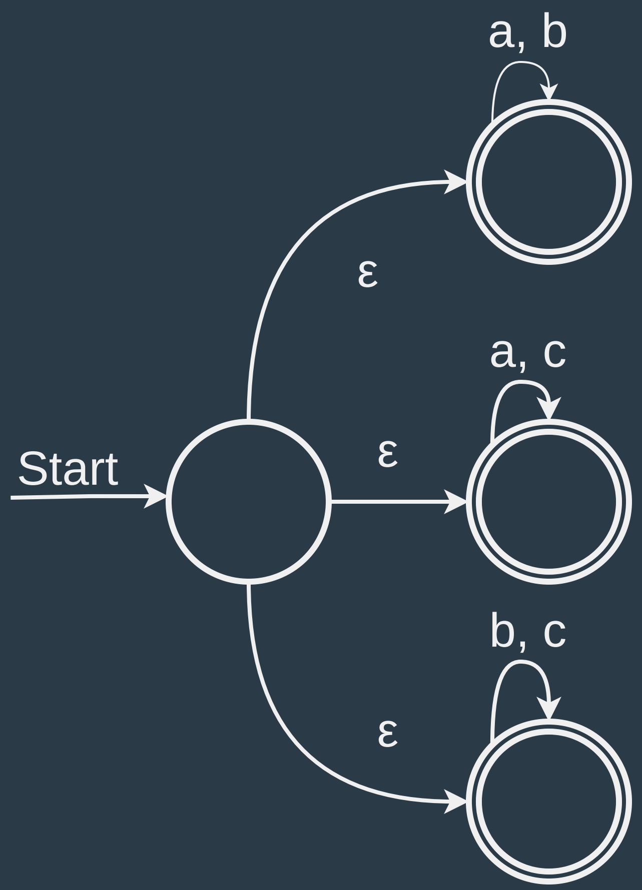


Guess-and-Check

$L = \{ w \in \{a,b,c\}^* \mid$ at least one of a, b, or c is not in w $\}$

Guess-and-Check

L = { w ∈ {a,b,c} * | at least one of a, b, or c is not in w }



NFAs and DFAs

- Any DFA is also an NFA
- So every DFA language is also an NFA language
- Question: Can every NFA language be accepted by a DFA?
- Surprisingly: **Yes!**

See you in the lab! 👋