Deterministic Finite Automata (DFA)

An abstract computer with extremely limited memory

Learning Objectives

By the end of this lecture, you will be able to:

- **Identify** real-world systems that can be modeled as DFAs
- **Construct** state transition diagrams for simple DFAs
- **Analyze** DFAs to determine their language
- **Design** DFAs to recognize specific languages
- **Implement** DFAs in code

Let's Start with Something Familiar

The Turnstile

Question: How many different states can a turnstile be in?

Turnstile States

The turnstile can be in one of two **states**:

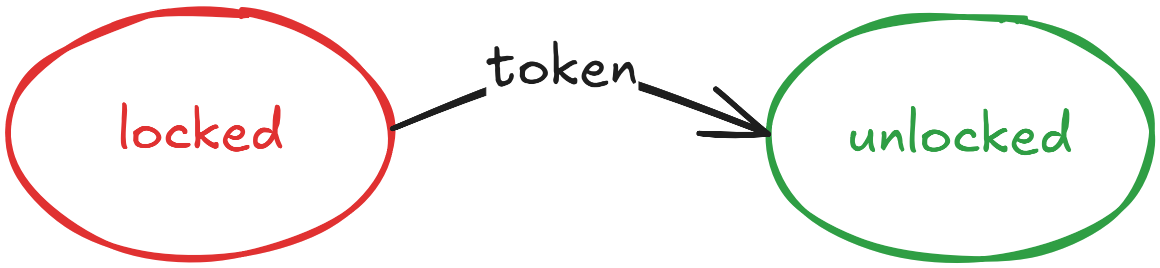
1. **Locked** - won't let you through
2. **Unlocked** - ready to let one person through

We can represent states as vertices in a graph:

State Transitions

What happens when you insert a token?

If the turnstile is **locked** and you insert a token, it **unlocks**

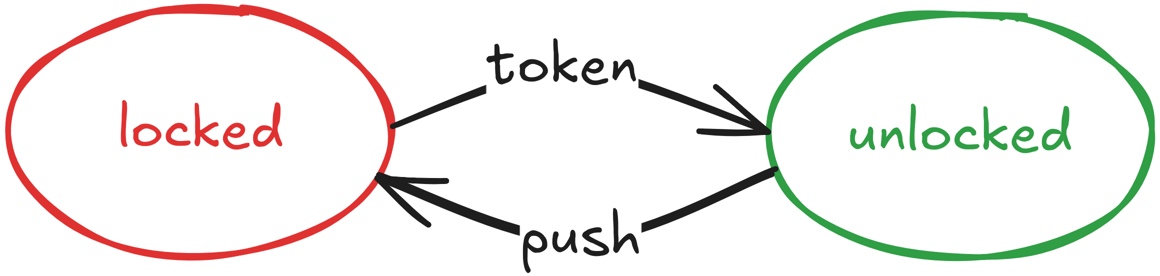


The **state-transition** is a directed edge labeled with the input/event

State Transitions (cont.)

What happens when you push?

If the turnstile is **unlocked** and you push it, it lets you pass and **locks** again

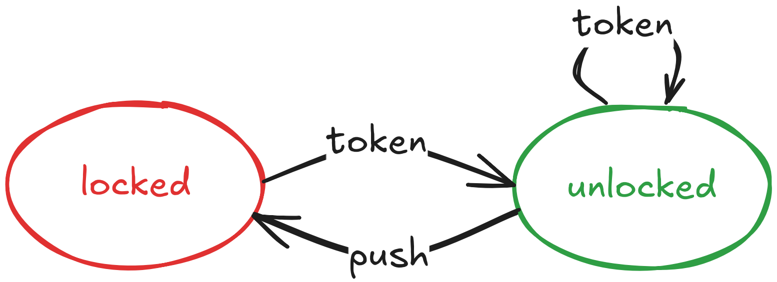


Complete the Model

What about the other cases?

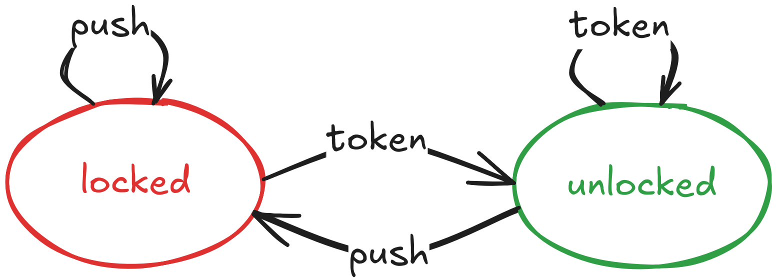
Token while unlocked:

- Stays unlocked



Push while locked:

- Stays locked



State Transition Table

Another way to represent the same information:

	Token	**Push**
Locked	Unlocked	Locked
Unlocked	Unlocked	Locked

Key Insight: The state-transition graph/table specifies a DFA that controls the turnstile

- Think of it as a single-bit computer storing the current state
- Contains circuitry to transition based on inputs

Active Learning

Think-Pair-Share (2 minutes)

Think of another real-world system that has:

- A finite number of states
- Transitions based on inputs

Share with your neighbor, then we'll discuss!

DFAs Are Everywhere!

Physical Systems

- **Elevators** - floors and button presses
- **Vending machines** - money inserted and product selection
- **Traffic lights** - timing and sensor inputs

Non-Physical Systems

- **Network protocols** - TCP three-way handshake
- **Text parsing** - finding patterns in strings
- **User input validation** - checking format correctness
- **Workflow management** - approval processes

Abstract DFA Definition

A DFA consists of:

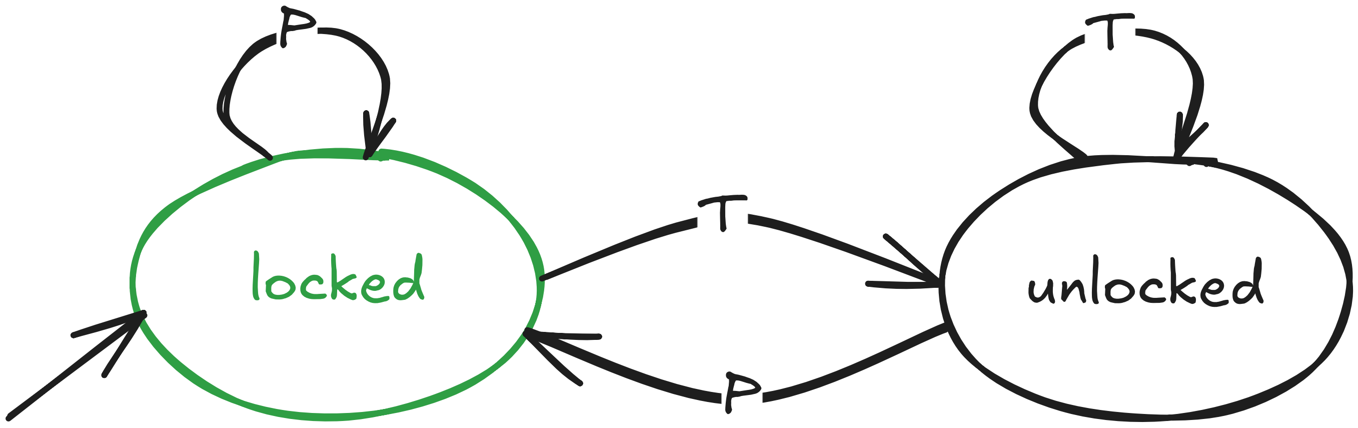
1. **States** (finite set)
    - One **start state** (initial state)
    - One or more **accept states** (final states)
2. **Alphabet** - finite set of input symbols
3. **Transitions** - rules for changing states based on input
    - Process input left to right
    - If in accept state after all input → string is **accepted**
    - Set of all accepted strings = **language** of the DFA

Applying Theory to Turnstile

Let's formalize our turnstile:

1. **States:** {locked, unlocked}
    - Start state: locked
    - Accept states: {locked}
2. **Alphabet:** {T, P}
    - T = token insertion
    - P = push
3. **Transitions:** As shown in our diagram

Complete Turnstile DFA

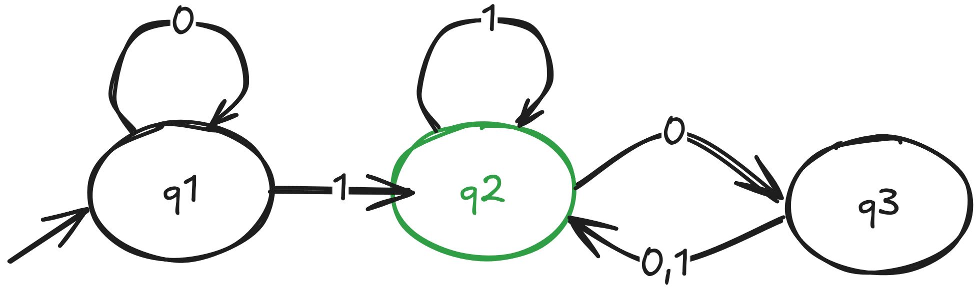


Language: All sequences that return turnstile to locked state

- Any number of P's (stay locked)
- Followed by: T (unlock) → any T's → P (lock)
- Repeat as needed

🎯

Practice Problem 1

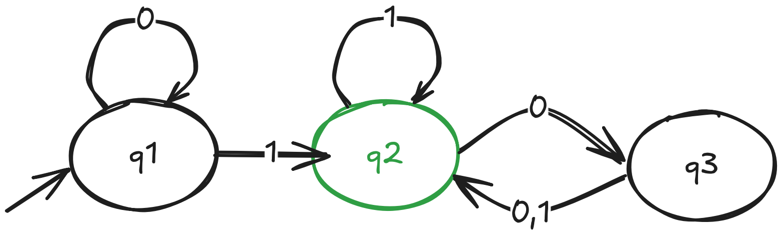


Questions:

1. What is the alphabet?
2. What strings does this DFA accept?

(Work individually for 2 minutes, then discuss)

Solution to Practice 1



1. **Alphabet:** {0, 1}
2. **Language:** Binary strings with:
   - At least one 1, AND
   - Even number of 0s after the last 1

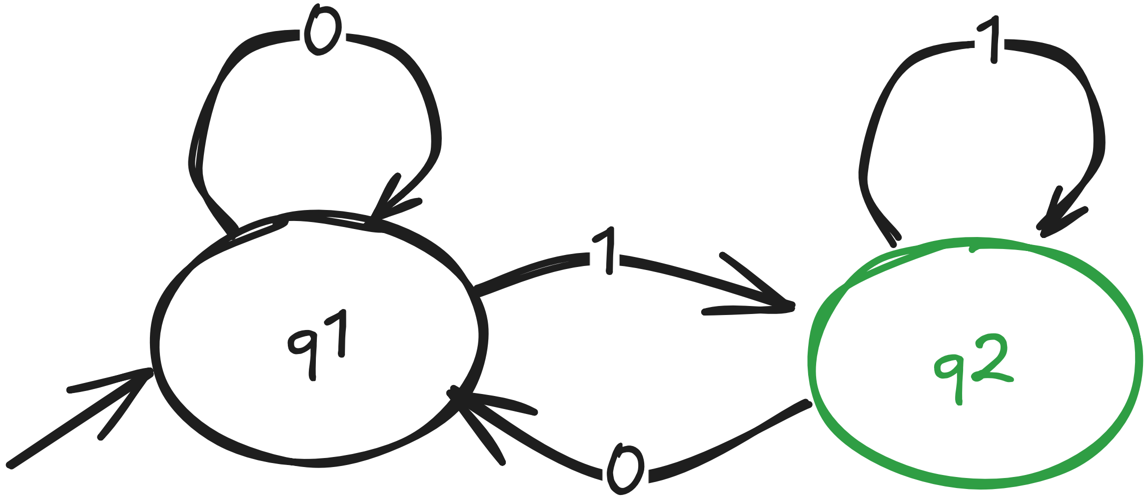
Design Challenge 1

Your Turn!

Design a DFA that recognizes binary strings that **end in 1**

(Work in pairs for 3 minutes)

Solution: Strings Ending in 1



Key insight:

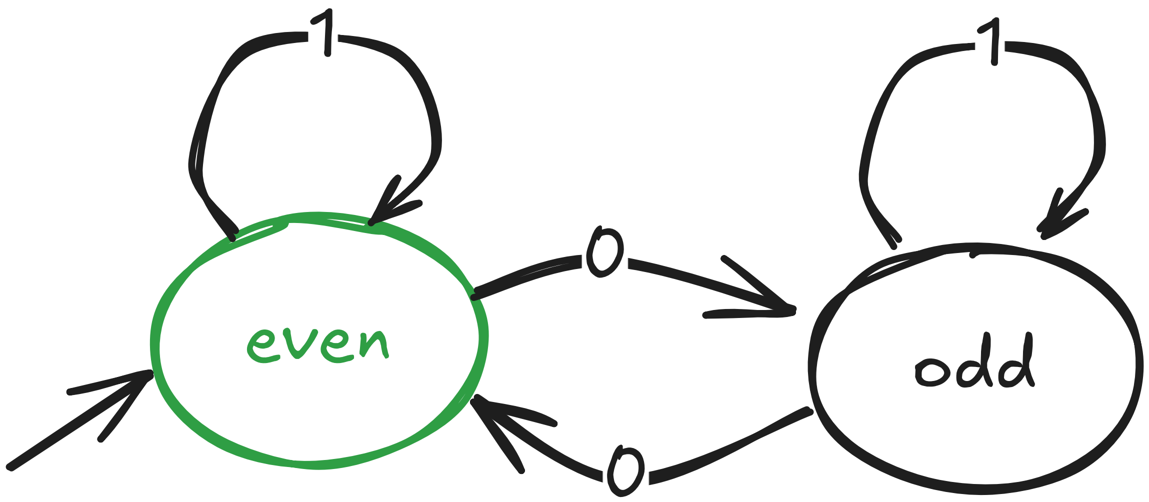
- q_0: Haven't seen a 1 yet OR just saw a 0
- q_1: Just saw a 1 (accept state)

Design Challenge 2

Design a DFA for binary strings with an **even number of 0s** (including no 0s and the empty string)

(Work in pairs for 3 minutes)

Solution: Even Number of 0s



Key insight:

- q_0: Even number of 0s seen (accept)
- q_1: Odd number of 0s seen
- 1s don't affect the count!

Implementing DFAs in Code

Java API Design

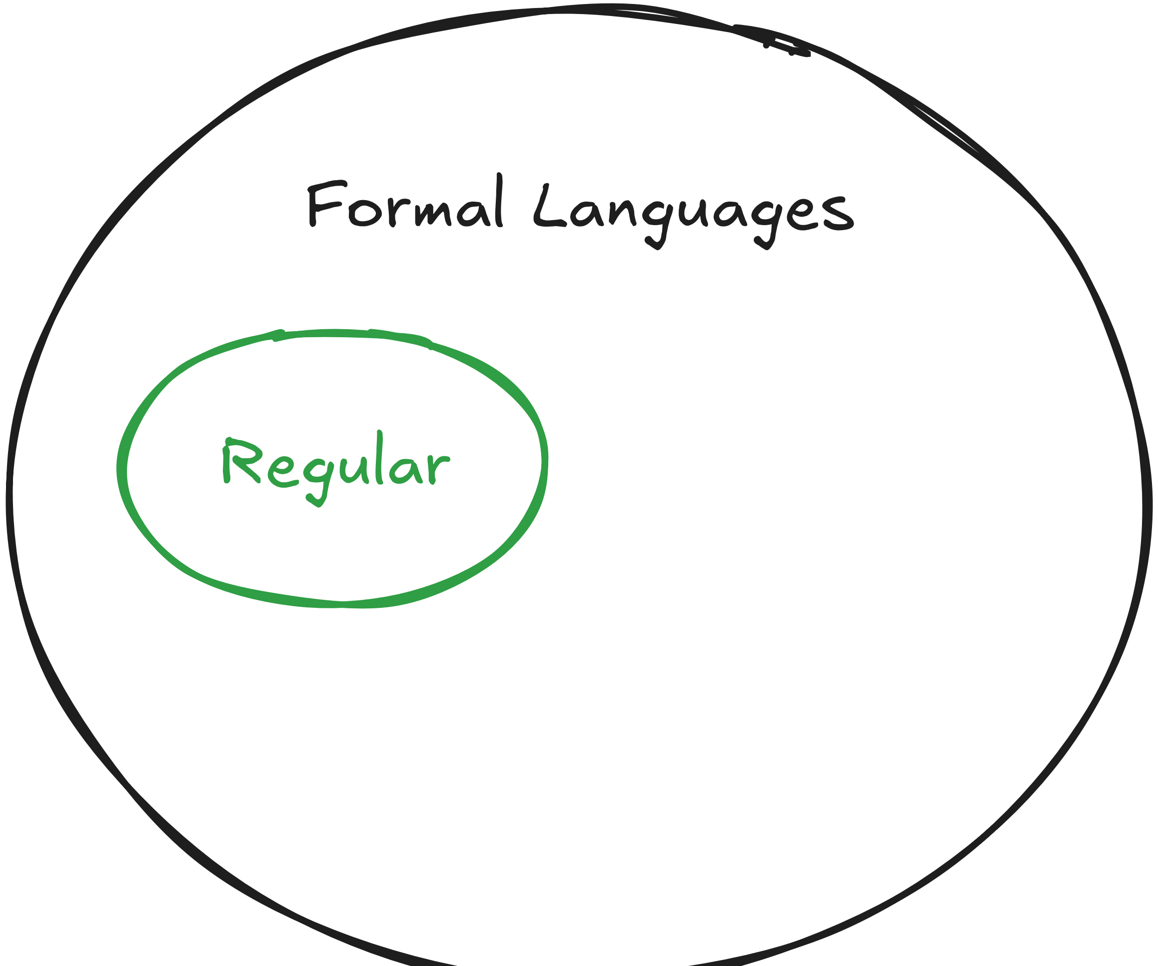
```java
public class DFA {
  public static class State {
    void addTransition(Character symbol, State to) {...}
    State getTransition(Character symbol) {...}
  }

  public void setStartState(State state) {...}
  public State getStartState() {...}
  public void addAcceptState(State state) {...}
  public Set<State> getAcceptStates() {...}

  public boolean accepts(String input) {...}
}
```

Turnstile Implementation

```
// Construct the DFA
DFA dfa = new DFA();
State locked = new State();
State unlocked = new State();

dfa.setStartState(locked);
dfa.addAcceptState(locked);

locked.addTransition('T', unlocked);
locked.addTransition('P', locked);
unlocked.addTransition('T', unlocked);
unlocked.addTransition('P', locked);

// Evaluate inputs
boolean result1 = dfa.accepts("TP");       // true
boolean result2 = dfa.accepts("PPTPTPPP"); // true
```

Regular Languages

> A formal language is called a **regular language** if some DFA recognizes it.



Summary

Key Takeaways

✅
DFAs are simple computational models with finite memory
✅
They consist of states, transitions, and an alphabet
✅
Real-world controllers often implement DFAs
✅
DFAs recognize regular languages
✅
We can implement and simulate DFAs in code