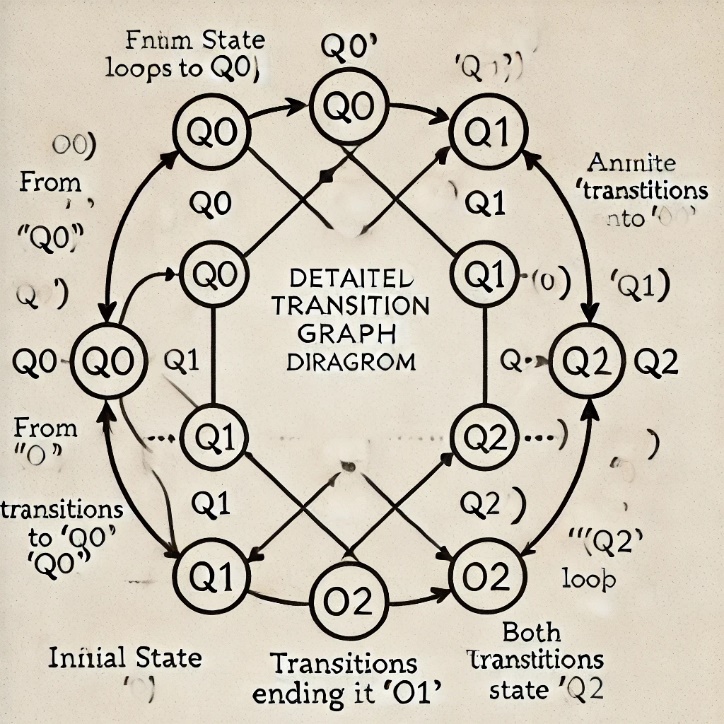
**Week 7: Transition Graphs and Kleene’s Theorem**

**Definition of Terms**

* **Transition Graphs:** Visual representations of states and transitions in automata, used to illustrate how an automaton moves between states based on inputs.
* **Kleene’s Theorem:** Establishes that any language recognized by a finite automaton can also be represented using regular expressions, bridging automata and regular expressions.

**Illustrative Diagram**

* Simple transition graph of a finite automaton accepting a binary string (e.g., 0s and 1s ending with “01”).



**Fig Week 7**: Transition graph for a finite automaton that accepts binary numbers ending in "01".

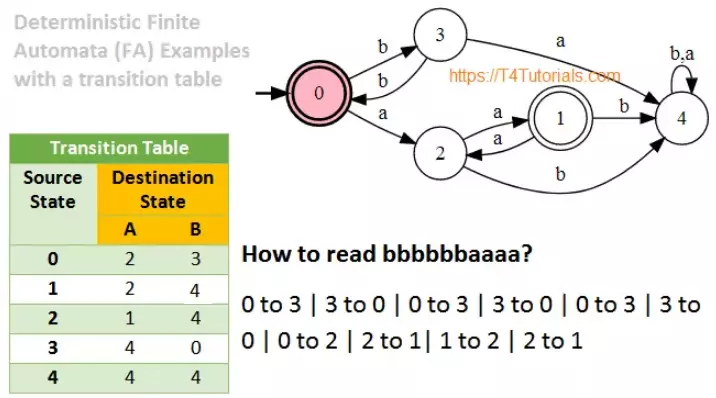


Illustration figure.

**Real-Life Example**

* **Transition Graph:** Think of a GPS app that changes its route based on traffic updates (inputs) and location (states). Each state transition reflects a change in the suggested path.
* **Kleene’s Theorem:** Similar to shortcuts in text processing software that recognize specific patterns (e.g., typing "btw" expands to "by the way").

**Applications**

* **Programming Language Compilers:** Use automata and transition graphs for syntax analysis.
* **Chatbots:** Transition graphs model state changes in conversations.

**Career Connection**

* **Compiler Engineer:** Designs systems that utilize transition graphs for language parsing.
* **Chatbot Developer:** Uses transition graphs to design conversational flows and responses.

**Weekly Task**

1. **Draw a Transition Graph:** Students create a transition graph for a simple scenario, such as validating phone numbers.
2. **Fun Real-World Challenge:** Identify real-life systems (e.g., vending machines) and represent their operation using transition graphs.

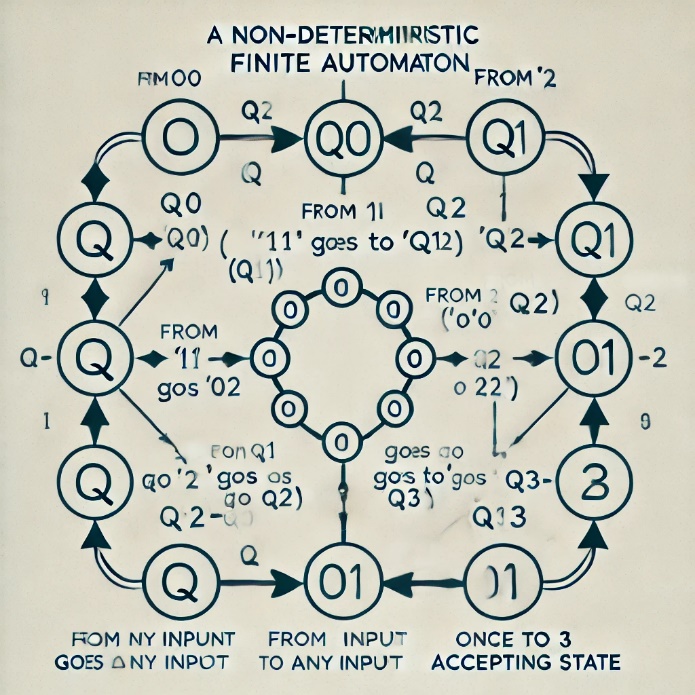
**Week 8: Non-determinism in Finite Automata**

**Definition of Terms**

* **Non-determinism:** A condition where multiple transitions for the same input may exist from a single state.
* **Non-deterministic Finite Automata (NFA):** An automaton where non-deterministic transitions allow for multiple possible outcomes from each state.

**Illustrative Diagram**

* NFA diagram showing multiple transitions for the same input symbol from a single state.

 **Fig Week 8**: Non-deterministic finite automaton (NFA) for recognizing binary strings containing the substring "101".

**Real-Life Example**

* **NFA Concept:** Think of searching for a book in multiple library sections simultaneously; you don’t follow a single, predefined path but explore various sections at once.

**Applications**

* **Pattern Matching:** NFAs are used in text editors and search algorithms, allowing multiple matches to be explored at once.
* **AI and Decision-Making Systems:** NFAs model scenarios with multiple decision paths (e.g., AI predicting different possible user actions).

**Career Connection**

* **Software Developer (Text Processing):** Uses NFAs in search functions and pattern recognition.
* **AI Engineer:** Designs systems that consider multiple possible actions or states.

**Weekly Task**

1. **Build an NFA Model:** Students model a simple NFA for searching keywords in a text.
2. **Group Challenge:** Explain a decision-making process (e.g., multiple routes to reach a destination) using an NFA model.

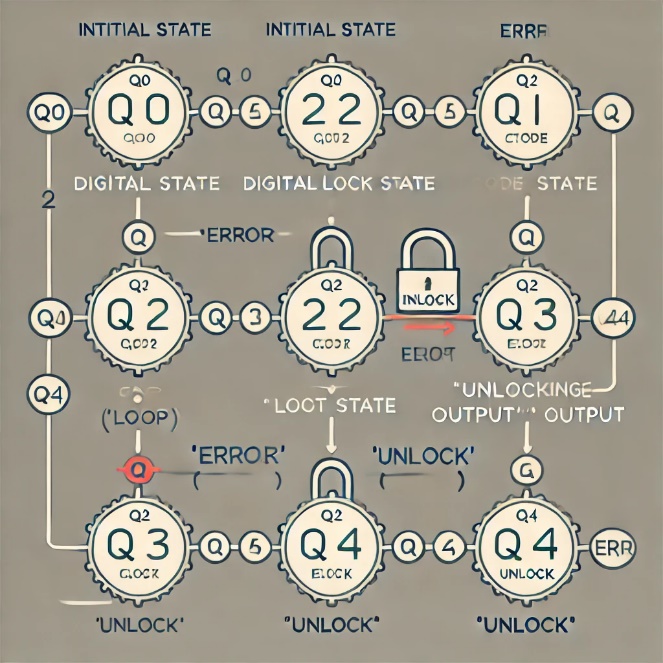
**Week 9: Finite Automata with Output**

**Definition of Terms**

* **Finite Automata with Output:** A finite automaton that produces output based on states or transitions, as seen in Moore or Mealy machines.

**Illustrative Diagram**

* Mealy and Moore machines showing inputs, state transitions, and resulting outputs.



**Fig Week 9**: Moore machine simulating a digital lock system that unlocks with the code "1234".

**Real-Life Example**

* **Finite Automata with Output:** Digital lock system where a specific code (input sequence) leads to a lock opening (output).

**Applications**

* **Digital Circuit Design:** Finite automata are used in designing circuits that produce specific outputs based on input sequences.
* **Automated Customer Service:** Chatbots use automata with output to respond with specific phrases based on input.

**Career Connection**

* **Embedded Systems Engineer:** Uses finite automata in designing devices that respond to inputs (e.g., keyboards).
* **UX/UI Developer:** Uses automata with output for interactive systems that respond to user actions.

**Weekly Task**

1. **Design an Automaton with Output:** Students design a simple finite automaton that outputs “access granted” or “denied” based on input codes.
2. **Practical Coding Challenge:** Implement a basic Mealy or Moore machine in code to simulate a simple password check system.

Expounded notes:

**Week 7: Transition Graphs and Kleene’s Theorem**

**In-Depth Explanation**

1. **Transition Graphs**
   * A **transition graph** is a visual way of representing the states and transitions of a finite automaton. In these graphs:
     + **States** are represented as circles, labeled with names like q0, q1, etc.
     + **Transitions** are arrows between states, labeled with the input symbols that trigger the transitions.
     + An **initial state** is marked by an arrow pointing toward it with no originating state.
     + **Accepting (or final) states** are usually marked with a double circle.
2. **Kleene’s Theorem**
   * Kleene’s Theorem connects **finite automata** and **regular expressions**, proving that any language recognized by a finite automaton can also be expressed using a regular expression.
   * It introduces three basic operations to construct complex expressions from simple ones:
     + **Concatenation** (joining sequences of symbols),
     + **Union** (using the “|” symbol to represent choice between symbols or strings), and
     + **Kleene Star** (allowing repetition of sequences).
   * This theorem helps create efficient **regular expressions** for patterns recognized by finite automata.

**Detailed Example**

* Consider a **simple transition graph** for a finite automaton that accepts binary numbers ending in “01”:
  + **States**: q0 (start), q1, q2 (accepting).
  + **Transitions**:
    - From q0, 0 → q0 and 1 → q1.
    - From q1, 0 → q2.
    - From q2, 0 or 1 → q2 (staying in q2 if the binary ends in "01").
* This automaton can be expressed with a regular expression as (0|1)\*01, thanks to Kleene’s Theorem.

**Real-Life Application Examples**

* **Vending Machines**: Each button press (input) triggers a state change, leading eventually to an outcome (e.g., dispensing an item). This interaction can be visualized using transition graphs.
* **Text Editing Shortcuts**: Regular expressions developed through Kleene’s Theorem are essential in word processors for tasks like finding and replacing patterns across documents.

**Career Connections**

* **Compiler Engineers**: Use automata and transition graphs to help parse code in compilers, translating high-level code to machine language.
* **Cybersecurity Analysts**: Detect attack patterns in network logs by using regular expressions based on automata for efficient scanning and pattern matching.

**Engaging Task**

* **Task 1: Transition Graph Challenge**
  + Design a transition graph for a real-world scenario, such as a parking garage ticketing system, where input events include ticket issue, payment, and exit.
* **Fun Activity: Movie Dialogue Pattern**
  + Ask students to create a transition graph representing a sequence in movie dialogues, e.g., tracking if certain keywords lead to a climactic response.

**Week 8: Non-Determinism in Finite Automata**

**In-Depth Explanation**

1. **Non-Determinism**
   * Non-determinism in automata means that, given a specific input symbol, the automaton may have multiple possible next states. Unlike deterministic automata, non-deterministic finite automata (NFA) can "choose" between multiple transitions from the same state.
   * For an NFA to accept an input string, at least one of its possible paths must lead to an accepting state. This makes NFAs highly versatile in scenarios where exploration of multiple paths or outcomes is required.
2. **NFA Construction**
   * An NFA consists of:
     + **A set of states**,
     + **An initial state**,
     + **A set of accepting states**,
     + **An alphabet of input symbols**, and
     + **A transition function** allowing for multiple possible next states.

**Detailed Example**

* Consider an **NFA** that recognizes strings containing the substring “101”:
  + **States**: q0 (start), q1, q2, q3 (accepting).
  + Transitions:
    - From q0, 1 → q1.
    - From q1, 0 → q2.
    - From q2, 1 → q3 (accepting).
  + The NFA allows for different paths, making it easy to match “101” even if extra symbols exist at the beginning or end of the string.

**Real-Life Applications**

* **Web Search Engines**: When a user searches for “AI developments,” the search engine uses NFAs to quickly find relevant documents that contain any of the terms or related patterns.
* **Robotics**: NFAs are useful in designing decision-making algorithms for autonomous robots that explore different paths based on sensor inputs.

**Career Connections**

* **Data Scientist**: NFAs are essential in text processing and pattern matching, helping data scientists handle large volumes of unstructured data.
* **Game Developer**: Uses NFAs for designing game scenarios where characters or objects follow multiple possible action paths based on player inputs.

**Engaging Task**

* **Task 1: NFA Design Challenge**
  + Students create an NFA that recognizes their name spelled out in letters, with alternate paths for possible typos.
* **Real-World Decision Paths**
  + Ask students to represent the decision-making process of a recommendation system (e.g., movie recommendations) using an NFA, where each state represents a genre or movie type.

**Week 9: Finite Automata with Output**

**In-Depth Explanation**

1. **Finite Automata with Output**
   * Unlike regular finite automata, which only accept or reject input strings, finite automata with output produce an output based on transitions or states.
   * Two common models:
     + **Moore Machine**: Outputs are determined solely by the state of the machine.
     + **Mealy Machine**: Outputs depend on both the state and the current input symbol.
2. **Moore vs. Mealy Machines**
   * In a **Moore Machine**, output is generated when entering a state.
   * In a **Mealy Machine**, output is generated with each transition, allowing for quicker response changes based on input.

**Detailed Example**

* Imagine a **digital lock** that only opens for a specific sequence like “1234”:
  + **States**: Each represents progress toward the correct code.
  + **Transitions**: Represent each correct digit entry.
  + **Output**: If all inputs are correct, the output is “unlock.”

**Real-Life Applications**

* **Traffic Light Systems**: Can be modeled with Moore machines to output signals (red, yellow, green) based on time intervals.
* **Vending Machines**: Use Mealy machines where each coin insert or button press leads to an output (change or product dispensed).

**Career Connections**

* **Embedded Systems Developer**: Uses finite automata with output to build control systems (e.g., washing machines, microwaves).
* **UI/UX Designer**: Incorporates automata with output to design responsive user interfaces, where the system outputs responses to specific user actions.

**Engaging Task**

* **Task 1: Moore/Mealy Machine Modeling**
  + Design a Moore or Mealy machine that simulates an automatic hand dryer, where a sensor (input) triggers the drying cycle (output) and changes based on distance.
* **Real-World Simulation**
  + Code a basic vending machine simulation that takes inputs (coin insertions) and produces outputs (products or change) as a finite automaton with output.