Question 1:

1. Context:

The chosen application is the recognition of airport codes. Airport codes are three-letter abbreviations used to identify airports worldwide, such as JFK for John F. Kennedy International Airport in New York or DFW for Dallas fort worth international airport. I chose this application because airport codes play a vital role in the aviation industry for ticketing, baggage handling, flight tracking, and communication between airlines and airports. Having an automated system to recognize valid airport codes can be beneficial in various airline operations, including reservation systems, flight management, and logistics.

2. Alphabet:

The alphabet for this example consists of uppercase letters from A to Z. It includes all the letters used in airport codes.

3. Description of the set of strings:

The set of strings for this example consists of three uppercase letters from A to Z. Each string in this set represents a valid airport code. This set is important because it allows us to validate and recognize valid airport codes in various applications. By ensuring that the input follows the pattern of three uppercase letters, we can quickly identify and process airport codes accurately.

4. Example strings:

- String in the set: "LAX"
- String not in the set: "123"

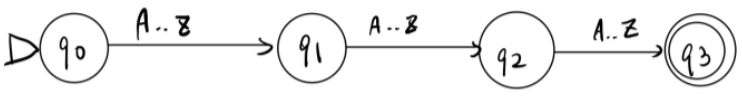
Explanation:

The string "LAX" is an example of a valid airport code and belongs to the set of recognized strings. It consists of three uppercase letters from A to Z, which is the pattern we are looking for.

On the other hand, the string "123" is not a valid airport code and does not belong to the set of recognized strings. It contains numeric characters, which are not part of the defined alphabet for this language.

5. Classification and Machine Construction:

The language of recognizing airport codes can be classified as a regular language. We can prove this by constructing a deterministic finite automaton (DFA) that recognizes this language.



DFA for Recognizing Airport Codes:

- States: q0, q1, q2, q3
- Alphabet: {A…Z}, letters from A to Z
- Initial State: q0

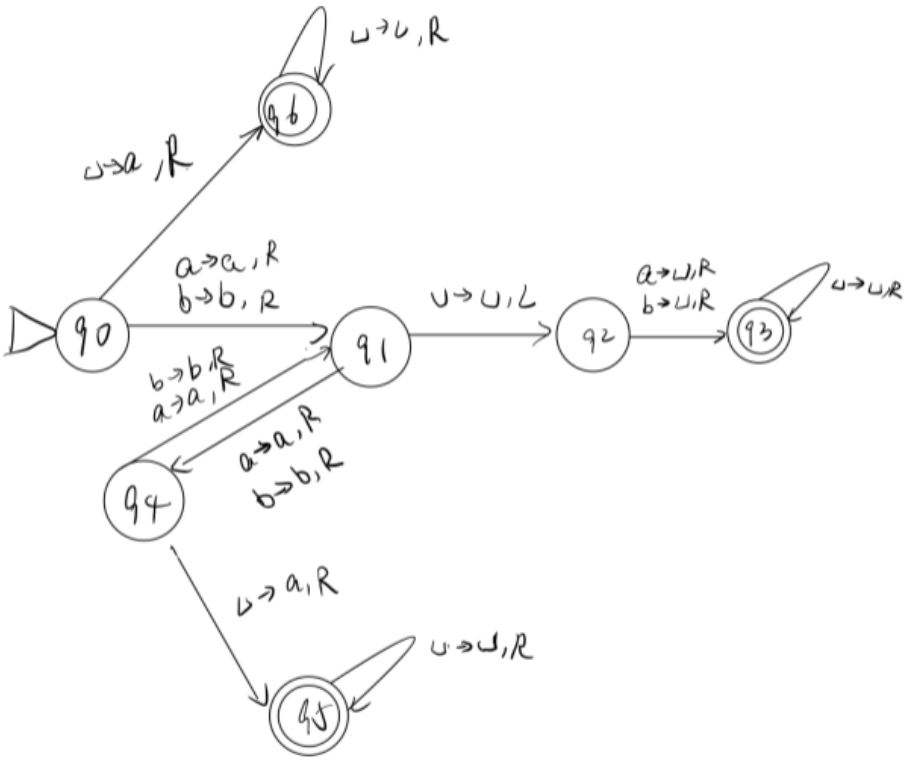
- Accepting State: The accepting state is q3. If the DFA reaches the accepting state after reading three uppercase letters, the input string is recognized as a valid airport code.
- Transitions: The DFA will have transitions based on the current state and the input symbol (uppercase letter) being read.

By constructing such a DFA, we can validate input strings efficiently and accurately determine whether they represent valid airport codes.

Question 2:
Function : For a string in {a, b}* of length n, if n is odd, then the output has the last character removed; if n is even, then the output is the input string with an ‘a’ append at last.

1.
We define the turing machine as follows: If the length of the input string is odd, remove the last character, then accept. If the length of the input string is even, append an ‘a’ at the end and accept.

2.



3.

```python
class TuringMachine:
    def __init__(self, tape):
        self.tape = list(tape)  # Convert the input string to a list
        self.head = 0  # Head position
        self.state = 'q0'  # Initial state

    def move_left(self):
        if self.head == 0:
            self.tape.insert(0, '_')
        else:
            self.head -= 1

    def move_right(self):
        self.head += 1
        if self.head == len(self.tape):
            self.tape.append('_')

    def write(self, symbol):
        self.tape[self.head] = symbol

    def read(self):
        if self.head >= len(self.tape):
            return '_'
        return self.tape[self.head]
```

```
    while (self.state != 'q6') and (self.state != 'q3') and (self.state != 'q5'):
        symbol = self.read()
        print("State:", self.state)
        print("Tape: ", "".join(self.tape))
        print("Head: ", " " * self.head + "^")
        print("--------------------")

        if (self.state, symbol) in transitions:
            new_state, new_symbol, direction = transitions[(self.state, symbol)]
            self.state = new_state
            self.write(new_symbol)
            if direction == 'L':
                self.move_left()
            else:
                self.move_right()
        else:
            print("Error: No transition defined for state '{}' and symbol '{}'".format(self.state, symbol))
            return

    print("State:", self.state)
    print("Tape: ", "".join(self.tape))
    print("Head: ",  " "*(self.head) + "^")
    print("--------------------")
    print("Computation finished.")
```

```
def run(self):

    if len(self.tape) == 0:
        print("State:", "q0")
        print("Tape: ", "".join(self.tape))
        print("Head: ", " " * self.head + "^")
        print("--------------------")
        print("State:", "q6")
        print("Tape: ", "a")
        print("Head: ", " " * self.head + " ^")
        print("--------------------")
        print("Computation finished.")
        return

    transitions = {
        ('q0', ''): ('q6', 'a', 'R'),  # Transition to q6, replace '' with 'a', move right
        ('q0', 'a'): ('q1', 'a', 'R'),  # Transition to q1, replace 'a' with 'a', move right
        ('q0', 'b'): ('q1', 'b', 'R'),  # Transition to q1, replace 'b' with 'b', move right
        ('q1', 'a'): ('q4', 'a', 'R'),  # Transition to q4, replace 'a' with 'a', move right
        ('q1', 'b'): ('q4', 'b', 'R'),  # Transition to q4, replace 'b' with 'b', move right
        ('q1', ''): ('q2', '', 'L'),  # Transition to q2, replace '' with '', move left
        ('q2', 'a'): ('q3', '', 'R'),  # Transition to q3, replace 'a' with '', move right
        ('q2', 'b'): ('q3', '', 'R'),  # Transition to q3, replace 'b' with '', move right
        ('q3', ''): ('q3', '', 'R'),  # Stay in q3, replace '' with '', move right
        ('q4', 'a'): ('q1', 'a', 'R'),  # Transition to q1, replace 'a' with 'a', move right
        ('q4', 'b'): ('q1', 'b', 'R'),  # Transition to q1, replace 'b' with 'b', move right
        ('q4', ''): ('q5', 'a', 'R'),  # Transition to q5, replace '' with 'a', move right
        ('q5', ''): ('q5', '', 'R'),  # Stay in q5, replace '' with '', move right
        ('q6', ''): ('q6', '', 'R'),  # Stay in q6, replace '' with '', move right
    }
```

4.

```
# Example usage
input_string = "abb"


tm = TuringMachine(input_string)
tm.run()
```

```
State: q0
Tape:  abb
Head:  ^
--------------------
State: q1
Tape:  abb
Head:   ^
--------------------
State: q4
Tape:  abb
Head:    ^
--------------------
State: q1
Tape:  abb
Head:     ^
--------------------
State: q2
Tape:  abb
Head:    ^
--------------------
State: q3
Tape:  ab
Head:     ^
--------------------
Computation finished.
```

5.

```
# Example usage
input_string = "ab"


tm = TuringMachine(input_string)
tm.run()
```

```
State: q0
Tape:  ab
Head:  ^
--------------------
State: q1
Tape:  ab
Head:   ^
--------------------
State: q4
Tape:  ab
Head:    ^
--------------------
State: q5
Tape:  aba
Head:     ^
--------------------
Computation finished.

Process finished with exit code 0
```