https://github.com/cs-ubbcluj-ro/lab-work-computer-science-2024-915-Micu-AlexiaClaudia/tree/main/2-Finite-Automata/Lab3

FiniteAutomaton.cs
This class represents the automaton and the following basic components: a list of states (representing the points we transition between), an alphabet (symbols used to transition from one state to another), an initial state, and one or more final states. The class parses data from FA.txt, containing these elements and a series of lines that define transitions. Each line in FA.txt follows the format A b B.

The states, alphabet, and final states will all be stored as lists of strings, with the initial state represented as a single string. Transitions will be stored in a dictionary structured as follows: each key in the dictionary is a KeyValuePair (a type created in an earlier project) with the first element of the key being the starting state and the second being the transition symbol. Each dictionary entry's value is a HashSet containing unique destination states for that transition.

- we determine if the automaton is deterministic: if every transition maps to at most one destination state, the automaton is deterministic; otherwise, it's non-deterministic.

The FiniteAutomaton class includes these key methods:

public void ReadFromFile(string filePath): Reads and initializes data structures from FA.txt.
public bool CheckIfDeterministic(): Returns a boolean indicating whether the automaton is deterministic.
Getters for accessing key elements, such as:
public List<string> GetStates()
public string GetInitialState()
public List<string> GetAlphabet()
public List<string> GetFinalStates()
public Dictionary<KeyValuePair<object, object>, HashSet<string>> GetTransitions()
public string WriteTransitions(): Outputs a list of all transitions.
public bool CheckSequence(string sequence): Determines if a given sequence (containing only alphabet symbols) is valid. Starting from the initial state, it verifies each symbol in the sequence has a corresponding transition and updates the current state accordingly. If the end state after parsing the sequence is a final state, the sequence is valid.

FA.txt uses EBNF format and contains the following definitions:

state = “M” | “N” | “O” | “P”;
initial state = “M”;
alphabet = “a” | “b” | “c” | “d”;
final state = “P”;
transition = state, “ ”, alphabet, “ ”, state;
transitions = transition, {transition};