DFA Library in Python

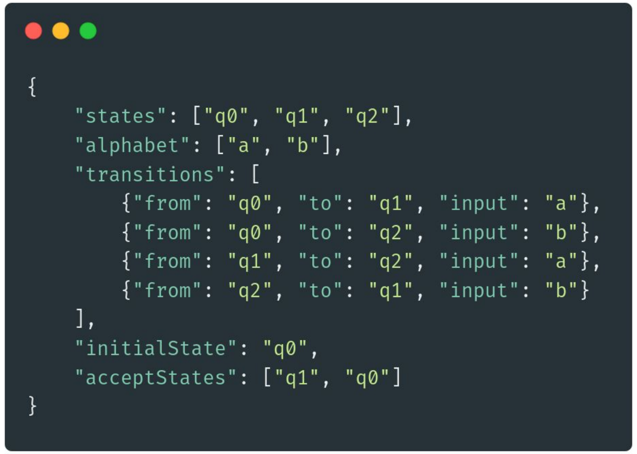
Curri Mattia

DFA Definition

A **Deterministic Finite Automaton (DFA)** is a mathematical model used to recognize patterns in sequences of inputs. It consists of a finite set of states, one of which is designated as the starting state, and a set of rules that determine how the automaton transitions from one state to another based on the input. Some states are marked as accepting states, meaning that if the automaton stops in one of these states after processing an input sequence, the input is considered accepted. Otherwise, it is rejected. A DFA ensures that for each state and input, there is exactly one possible next state.

DFA: json format

states: the set of states of the DFA
alphabet: the set of acceptable input symbols
transitions: the set of transitions. A transition is described as follows:
- from: current state
- to: next state
- input: input that triggers the transition
initialState: starting state
acceptStates: set of acceptance



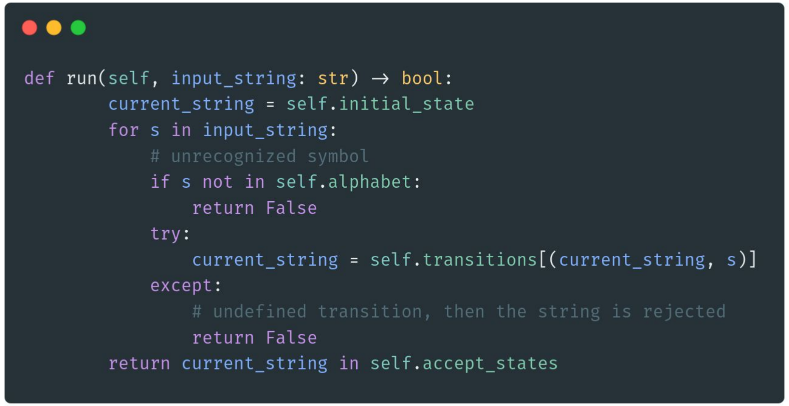
```
{
    "states": ["q0", "q1", "q2"],
    "alphabet": ["a", "b"],
    "transitions": [
        {"from": "q0", "to": "q1", "input": "a"},
        {"from": "q0", "to": "q2", "input": "b"},
        {"from": "q1", "to": "q2", "input": "a"},
        {"from": "q2", "to": "q1", "input": "b"}
    ],
    "initialState": "q0",
    "acceptStates": ["q1", "q0"]
}
```

Features: run

Processes an input string using the DFA and returns True if the string is accepted, False otherwise.

Args:
input_string (str): The input string to be processed.

Returns:
bool: True if the input string is accepted by the DFA, False otherwise



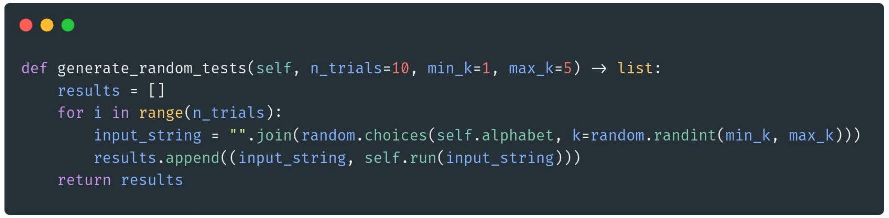
```python
def run(self, input_string: str) → bool:
        current_string = self.initial_state
        for s in input_string:
            # unrecognized symbol
            if s not in self.alphabet:
                return False
            try:
                current_string = self.transitions[(current_string, s)]
            except:
                # undefined transition, then the string is rejected
                return False
        return current_string in self.accept_states
```

Features: random tests

Generates random input strings and tests them against the DFA.

Args:
- n_trials (int): The number of random input strings to generate and test.
- min_k (int): The minimum length of the generated input strings.
- max_k (int): The maximum length of the generated input strings.

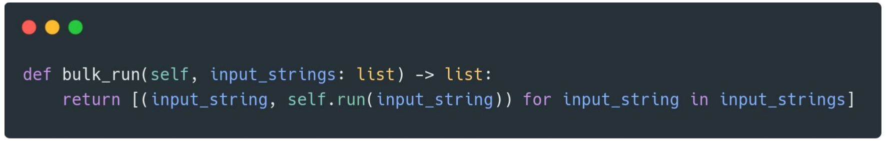
Returns:
- list: A list of tuples containing the input strings and their corresponding results (True if accepted, False otherwise).



```python
def generate_random_tests(self, n_trials=10, min_k=1, max_k=5) -> list:
    results = []
    for i in range(n_trials):
        input_string = "".join(random.choices(self.alphabet, k=random.randint(min_k, max_k)))
        results.append((input_string, self.run(input_string)))
    return results
```

Feature: bulk run

Runs a list of input strings through the DFA and returns a list of results.
Args:
- input_strings (list): A list of input strings to be processed.
Returns:
- list: A list of tuples containing the input strings and their corresponding results (True if accepted, False otherwise).

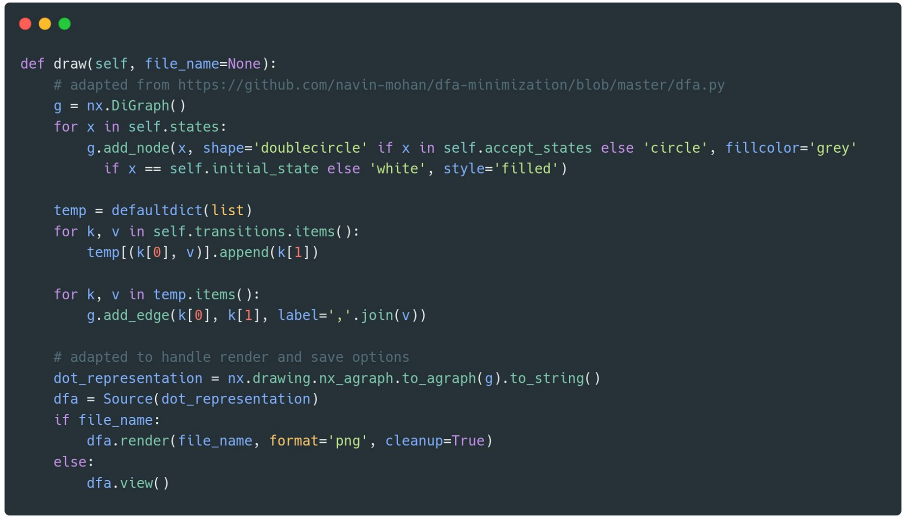


```python
def bulk_run(self, input_strings: list) -> list:
    return [(input_string, self.run(input_string)) for input_string in input_strings]
```

Feature: draw

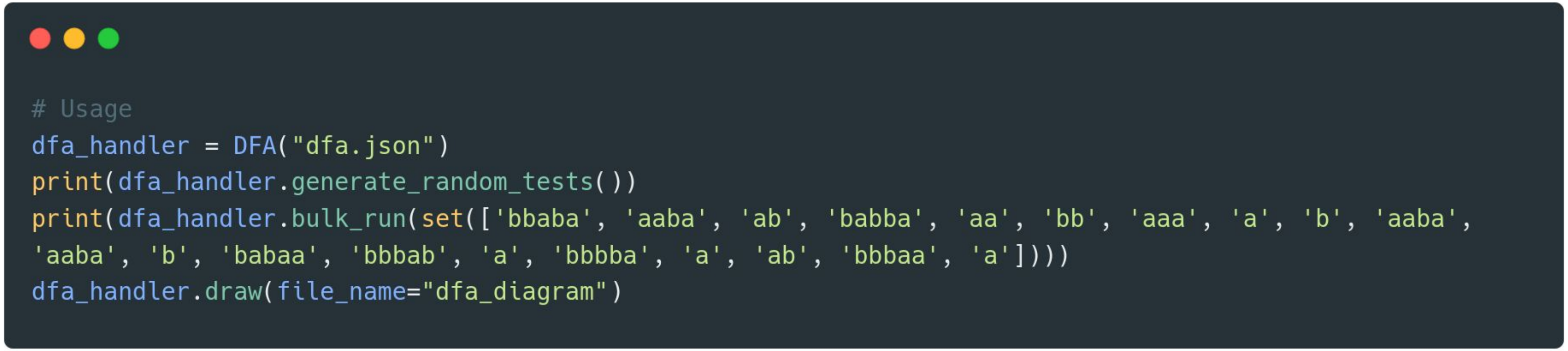
Draws a diagram of the DFA. Optionally saves the diagram to a file.

Args:
- file_path (str): The file path to save the diagram. If None, the diagram will be displayed but not saved.



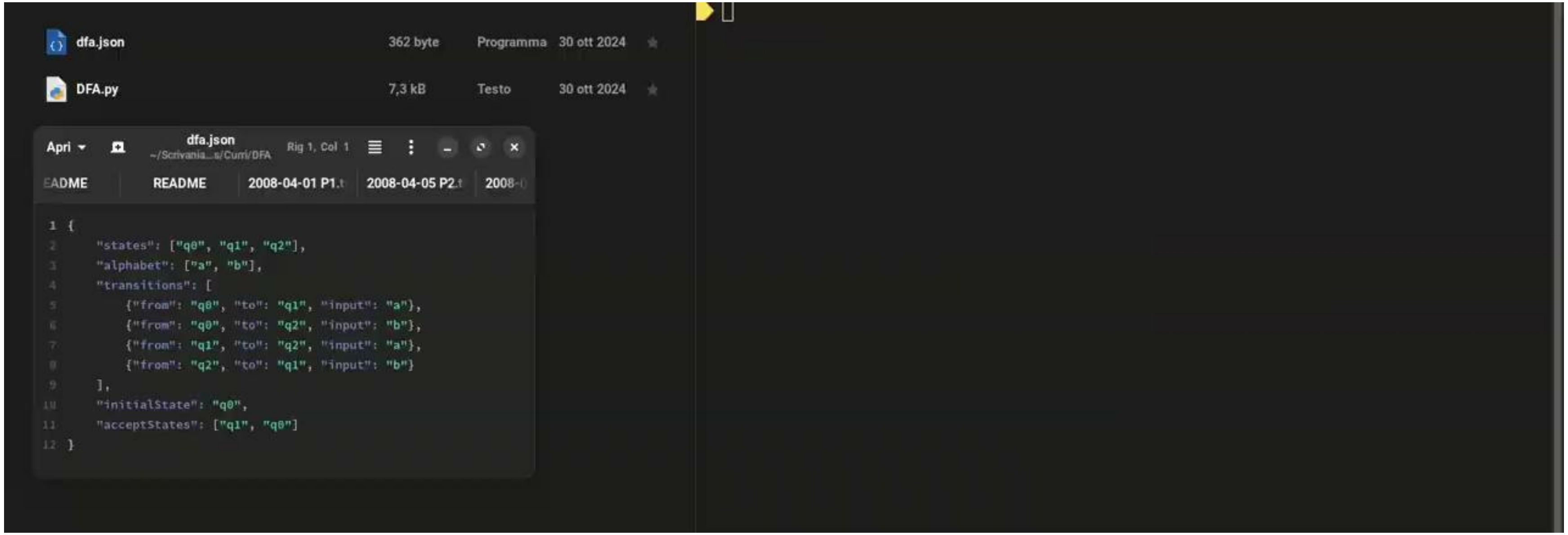
```python
def draw(self, file_name=None):
    # adapted from https://github.com/navin-mohan/dfa-minimization/blob/master/dfa.py
    g = nx.DiGraph()
    for x in self.states:
        g.add_node(x, shape='doublecircle' if x in self.accept_states else 'circle', fillcolor='grey'
          if x == self.initial_state else 'white', style='filled')

    temp = defaultdict(list)
    for k, v in self.transitions.items():
        temp[(k[0], v)].append(k[1])

    for k, v in temp.items():
        g.add_edge(k[0], k[1], label=','.join(v))

    # adapted to handle render and save options
    dot_representation = nx.drawing.nx_agraph.to_agraph(g).to_string()
    dfa = Source(dot_representation)
    if file_name:
        dfa.render(file_name, format='png', cleanup=True)
    else:
        dfa.view()
```

Usage example



```python
# Usage
dfa_handler = DFA("dfa.json")
print(dfa_handler.generate_random_tests())
print(dfa_handler.bulk_run(set(['bbaba', 'aaba', 'ab', 'babba', 'aa', 'bb', 'aaa', 'a', 'b', 'aaba',
'aaba', 'b', 'babaa', 'bbbab', 'a', 'bbbba', 'a', 'ab', 'bbbaa', 'a'])))
dfa_handler.draw(file_name="dfa_diagram")
```

Usage Example



```
{
    "states": ["q0", "q1", "q2"],
    "alphabet": ["a", "b"],
    "transitions": [
        {"from": "q0", "to": "q1", "input": "a"},
        {"from": "q0", "to": "q2", "input": "b"},
        {"from": "q1", "to": "q2", "input": "a"},
        {"from": "q2", "to": "q1", "input": "b"}
    ],
    "initialState": "q0",
    "acceptStates": ["q1", "q0"]
}
```