Software Testing and Quality Engineering

Course Project: General Algorithms for FSMs
Random FSM Generator

b00088564 Samir Mahmood
b00087520 Koushal Parupudi

Problem Definition:

The task is to develop a tool that can generate a large number of random FSMs following specified parameters.

Input:

The following parameters are specified through the command line:

- Number of FSMs generated
- Number of states per FSM
- Number of possible inputs
- Number of possible outputs
- Percentage of generated FSMs which are initially connected
- Percentage of generated FSMs which are complete

Output:

The tool will output a text file containing the generated FSMs. An FSM is the FSM name on line 1 followed by a list of transitions, one on each line. Each transition is in the format <Current_State,Input,Next_State,Output>. FSMs are separated by a single newline.

Related Work

Random FSM Generation

The generation of Finite State Machines (FSMs) through random methods has garnered significant attention due to its utility in various fields such as software testing, hardware design, and system modeling. Several approaches have been proposed for the random generation of FSMs, each aiming to produce models that capture essential behavioral characteristics while avoiding biases inherent in manual construction.

Brute Force Enumeration:

Early methods for FSM generation often relied on exhaustive enumeration of all possible state-transition combinations within predefined constraints. While straightforward, this approach becomes computationally infeasible for FSMs with large state spaces, limiting its scalability.

Random Walk and Monte Carlo Methods:

Random walk-based techniques simulate FSM construction by iteratively selecting random transitions from an initial state, evolving the machine until a termination condition is met. Monte Carlo methods enhance this process by incorporating probabilistic criteria to guide state and transition selection. These approaches offer simplicity and efficiency but may struggle to produce diverse FSM structures without additional heuristics.

Genetic Algorithms and Evolutionary Techniques:

Inspired by natural selection principles, genetic algorithms and evolutionary techniques iteratively refine populations of FSMs through mutation, crossover, and selection operations. These methods leverage fitness functions to assess the quality of generated models, encouraging the emergence of FSMs exhibiting desired properties such as fault coverage, input-output behavior, or structural complexity.

Constraint-Based Generation:

Contrary to purely random methods, constraint-based approaches impose specific criteria or constraints on the generated FSMs, ensuring adherence to predefined specifications. Techniques such as constraint satisfaction and constraint solving algorithms enable the systematic exploration of FSM design spaces while guaranteeing desired properties such as determinism, minimality, or reachability.

Evaluation of Randomly Generated FSMs

The evaluation of randomly generated FSMs is essential to assess their quality, relevance, and suitability for intended applications. Various metrics and criteria are employed to quantify the effectiveness and fidelity of generated models:

State Space Coverage:

Measuring the proportion of reachable states within the FSM provides insights into its comprehensiveness and ability to capture system behavior under different inputs and conditions.

Transition Coverage:

Analyzing the coverage of possible state transitions evaluates the connectivity and dynamism of the generated FSM, ensuring adequate exploration of state-transition space.

Structural Properties:

Metrics such as state count, transition count, and average transition density offer quantitative assessments of FSM complexity, aiding in the identification of overly simplistic or overly complex models.

Functional Properties:

Validation against functional requirements and specifications verifies whether the FSM accurately represents desired system behaviors and constraints, validating its relevance for intended applications.

Performance:

Evaluation of computational resources required for FSM generation, simulation, and analysis helps assess the scalability and efficiency of random generation techniques, guiding their applicability to real-world scenarios.

Comparison with Reference Models:
Comparative analysis against reference FSMs or manually constructed models serves as a benchmark for assessing the effectiveness and diversity of random generation methods, highlighting strengths, limitations, and areas for improvement.

In summary, the evaluation of randomly generated FSMs encompasses a diverse array of metrics and criteria, aiming to ensure the production of high-quality models that faithfully represent system behaviors while balancing considerations of efficiency, scalability, and relevance to practical applications.

Methodology:

Input Parameters: The algorithm takes in the following parameters:

num_states: The number of states in the finite state machine (FSM).

num_outputs: The number of possible outputs for each transition.

num_inputs: The number of possible inputs to the FSM.

initial_connected: A boolean indicating whether the FSM should be initially connected (all states reachable from the initial state).

complete_input: A boolean indicating whether the FSM should be complete (all states have transitions for all inputs).

Partialness: A floating point input that dictates the percentage of partialness in the generated FSM (range = [0,1])

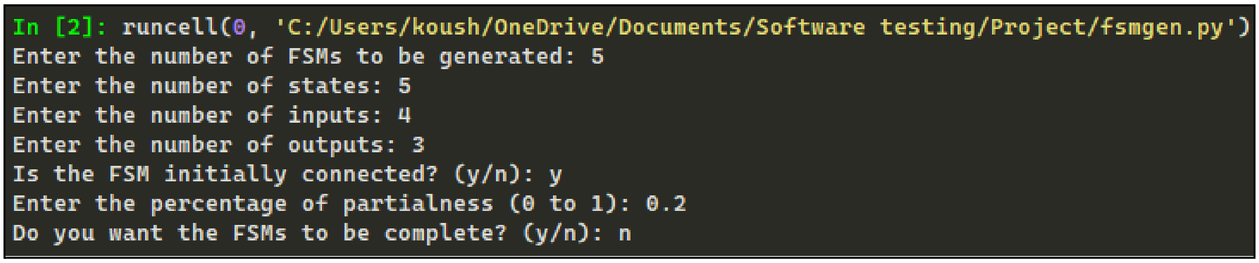
Algorithm:

1. Initialize FSM: Create an empty dictionary FSM to represent the FSM.
2. Generate Transitions: Loop through each state in the FSM and for each state, loop through each possible input. For each input, randomly generate a next state and an output, and store them in the FSM dictionary.
3. Check Initial Connectedness (if required): If initial_connected is True, perform a Depth-First Search (DFS) traversal starting from the initial state (state 0) to check if all states are reachable. If any state is unreachable, return False indicating failure.
4. Enforce Completeness (if required): If complete is True, ensure that every state has transitions for every possible input. If any input is missing for a state, randomly generate a next state and an output for that input and add it to the FSM dictionary.

The algorithm randomly generates transitions between states for each possible input, optionally ensures initial connectedness, and optionally enforces completeness of the FSM.

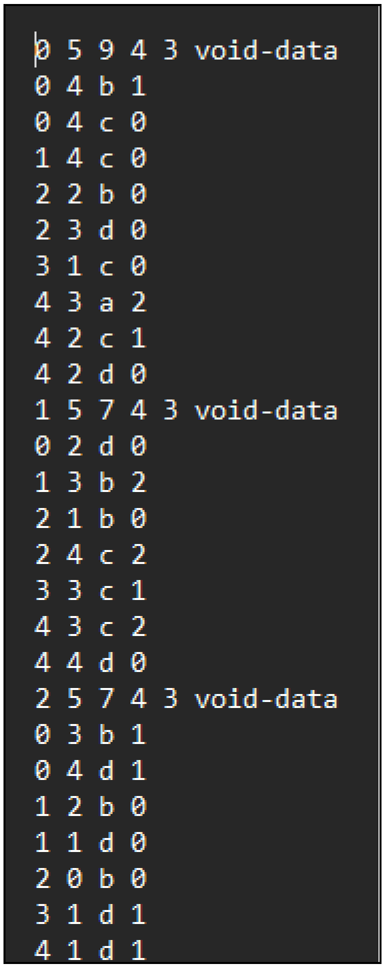
Result Mapping and Output:

Generated FSMs are output to a local text file called “generated_fsms.txt”.



```
In [2]: runcell(0, 'C:/Users/koush/OneDrive/Documents/Software testing/Project/fsmgen.py')
Enter the number of FSMs to be generated: 5
Enter the number of states: 5
Enter the number of inputs: 4
Enter the number of outputs: 3
Is the FSM initially connected? (y/n): y
Enter the percentage of partialness (0 to 1): 0.2
Do you want the FSMs to be complete? (y/n): n
```

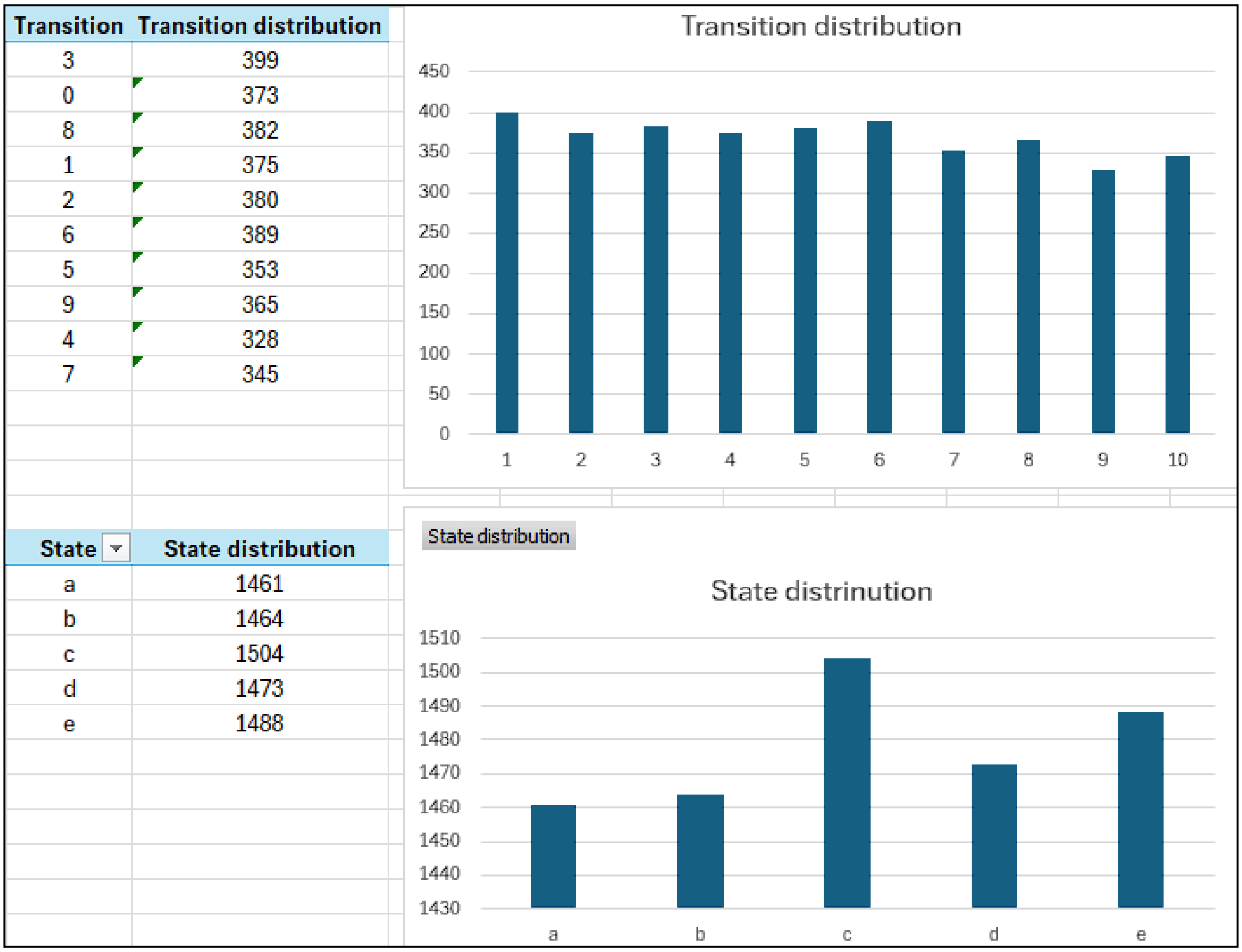
Figure 1: User defined requirements



```
0 5 9 4 3 void-data
0 4 b 1
0 4 c 0
1 4 c 0
2 2 b 0
2 3 d 0
3 1 c 0
4 3 a 2
4 2 c 1
4 2 d 0
1 5 7 4 3 void-data
0 2 d 0
1 3 b 2
2 1 b 0
2 4 c 2
3 3 c 1
4 3 c 2
4 4 d 0
2 5 7 4 3 void-data
0 3 b 1
0 4 d 1
1 2 b 0
1 1 d 0
2 0 b 0
3 1 d 1
4 1 d 1
```

Figure 2: Text file output of input, output and transitions for each state in the FSM

Evaluation of Randomness: Evaluate the randomness of the generated FSMs. This involves analyzing statistical properties such as distribution of states, transitions, and other characteristics to ensure that the generator produces diverse and unpredictable FSMs.



Transition	Transition distribution
3	399
0	373
8	382
1	375
2	380
6	389
5	353
9	365
4	328
7	345

State	State distribution
a	1461
b	1464
c	1504
d	1473
e	1488