The 20th Asia Pacific International Conference on Information Science and Technology (APIC-IST 2025)

July 06-09, 2025, SAii Laguna Resorts, Phuket, Thailand

Outstanding Paper Award

has been awarded for the paper entitled

"Scenario-based Modeling in AI Software Validation"

by

Janghwan Kim (Hongik Univ., ROK)
Kidu Kim (TTA, ROK)
Hyun Seung Son (Mokpo National Univ., ROK)
R. Young Chul Kim (Hongik Univ., ROK)

Chang Gyoon Lim, Ph.D.

APIC-IST 2025 Conference Chair

ISSN 2093-0542




KOREAN SOCIETY FOR INTERNET INFORMATION

The 20th Asia Pacific International Conference on Information Science and Technology

(APIC-IST 2025)

July 06-09, 2025, SAii Laguna Resorts, Phuket, Thailand

http://www.apicist.org

Proceedings of APIC-IST 2025

| Organized by |

Korean Society for Internet Information (KSII)

Contents

Scenario-based Modeling in AI Software Validation

Janghwan Kim[1], Kidu Kim[2], Hyun Seung Son[3], and R. Young Chul Kim[1*]
[1] Department of Software and Communications Engineering, Hongik University Seoul, South Korea
[2] AI Infrastructure Team, Telecommunications Technology Association, Seoul, South Korea
[3] Department of Computer Engineering, Mokpo National University, Mokpo, South Korea,
[e-mail: lentoconstante@hongik.ac.kr, kdkim@tta.or.kr, hson@mnu.ac.kr, bob@hongik.ac.kr]
*Corresponding author: R. Young Chul Kim

Abstract

Recently, there has been a significant interest in AI and software in both academic and industrial areas. However, in this moment, no one is concerned with validating AI software, such as reinforcement learning models, which may be particularly challenging. To address this issue, we propose our Scenario-based Modeling in AI Software Validation mechanism, which incorporates concepts from state machine diagrams, Markov Decision Processes (MDPs), stochastic state decision processes, and scenario-based integration testing. This mechanism enhances probabilistic modeling, which defines all possible scenarios on AI software and explicitly represents probability and reward values on the edges and nodes. Then we convert the scenario-based tree with it to adapt the priority of all possible scenario paths. Finally, we generate sequential event flows, such as test scripts, and validate the AI model with them. Through this, it may offer an effective method to systematically validate all available scenarios for AI Software and increase testing productivity. Although there is currently a significant manual effort and resource constraint at one step of scenario definition and transformation, it is expected that we will improve the validation process through the development of automation techniques and new coverage measurements.

Keywords: AI Software Validation, Scenario-based Modeling, Markov Decision Process

1. Introduction

In the field of software engineering, the importance of AI technologies, such as "AI for SE," has recently been emphasized. Research on applying AI technologies, either partially or entirely, beyond traditional manual testing methods, is actively underway in the software verification stage [1]. However, AI scientists tend to focus primarily on improving model verification performance and accuracy. Research on the Validation of AI software in the field of software engineering remains insufficient [2]. In particular, while supervised learning is easy to verify because the dataset includes answers, reinforcement learning faces a fundamental challenge in identifying optimal inputs and confirming results, making the validation of software with reinforcement learning models essential [3]. The premise of traditional software testing methodologies is the consistency of expected results for the same input. However, reinforcement learning-based AI software is challenging to validate with these existing methods because the model's decision factors change dynamically [4].

To address these limitations, we propose a Scenario-based Testing method that utilizes scenario graphs to effectively verify the safety, reliability, and accuracy of AI-based software

This research was conducted with the support of the Korea Creative Content Agency (Project Name: Artificial Intelligence-Based Interactive Multimodal Interactive Storytelling 3D Scene Authoring Technology Development, Project Number: RS-2023-00227917, Contribution Rate: 100%) and the Korea Research Foundation's four, Brain Korea 21 (Project Name: Ultra-Distributed Autonomous Computing Service Technology Research Team, Project Number: 202003520005).

equipped with reinforcement learning models from a software engineering perspective. Scenario-based testing is expected to comprehensively consider various situations and interactions that can occur in complex and dynamic reinforcement learning environments, thereby enabling the systematic detection and verification of unpredictable behaviors in AI software.

Chapter 2 discusses related research on AI software and testing methods. Chapter 3 introduces the scenario-based modeling mechanism for Scenario-based modeling in AI Software Validation, and Chapter 4 discusses conclusions and future research.

2. Related Works

2.1 State Machine Diagram

Mealy and Moore-based State Machine Diagrams (SMD) are modeling techniques that visually represent the possible states of a system or object and the transitions between those states.

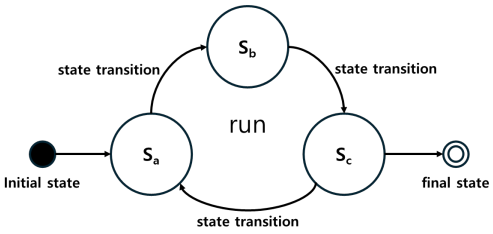


Fig. 1. Mealy & Moore State Machine Diagram

Mealy & Moore machines, defined as $M=(S,\Sigma,O,\delta,\lambda,s_0)$, share commonalities but differ in their output functions as follows:

(1) $\lambda: S\times\Sigma\rightarrow O$ (depends on current state & input)

(2) $\lambda: S\rightarrow O$ (depends only on the current state)

As shown in **Fig. 1**, SMDs demonstrate how a system changes states and responds when a specific event occurs, serving as an essential tool for analyzing and designing software system behavior [5]. In traditional software validation, SMDs have been used to explore all possible paths of a system and derive test cases for verifying expected behaviors in each state and transition [6].

2.2 Markov Decision Process

The Markov Decision Process (MDP) is a framework for mathematically modeling sequential decision-making problems, serving as the theoretical foundation for reinforcement learning. An MDP consists of five core elements: a set of states (S), a set of actions (A), transition probabilities (P), rewards (R), and a discount factor (γ). **Fig. 2** illustrates the Markov Decision Process [7].

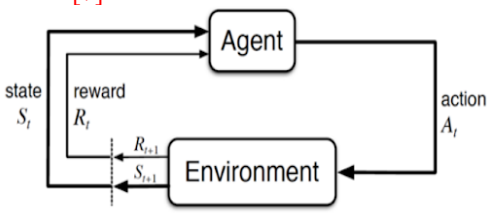


Fig. 2. Markov Decision Process (MDP)

(3) $MDP = (S, A, P, R, \gamma),\ s\in S,\ a\in A,\ (0\leq\gamma\leq 1),$

(4) $P(s'|s,a)=P[St+1=s'|St=s,At=a]$

(5) $R(s,a)=E[Rt+1|St=s,At=a]$

When an agent takes an action (At) in a specific state (St), the environment transitions to a new state (S_{t+1}) and provides a corresponding reward (R_{t+1}) to the agent. At this time, the next state (S_{t+1}) depends only on the current state (S_t) and action (A_t), following the Markov Property, which states it is independent of past states. The discount factor (γ) represents the present value of future rewards and has a value between 0 and 1. This value determines whether the agent places a greater emphasis on immediate rewards or future rewards. As γ approaches 0, the agent focuses more on immediate rewards, while as it approaches 1, it makes decisions considering future rewards from a long-term perspective.

2.3 Stochastic State Decision Process

The Stochastic State Decision Process (SSDP) is an approach that analyzes and models how a system makes decisions among various states, incorporating stochastic elements. It consists of five components: a finite set of events (Σ), a finite set of states (S), a state transition function (g), an initial state (S0), and a set of final states (F).

Σ: A finite set of events. S: A finite set of states.

g: A state transition function, g: S×Σ→ff(E), where ff is a

random function, S=∬(Ψ, Gc) and E⊆E, and E is a set of events.

F: A set of final states, F ⊆S. S0: An initial state, s0 ∈S.

Fig. 3 is an example of a Stochastic-based State Diagram. Each state transition is defined in the form of [E/Gc/A/P], including a probability weight value (P) in addition to the existing event (E), guard condition (Gc), and action (A), which accurately reflects the system's probabilistic characteristics and enhances simulation accuracy. Furthermore, it improves the dynamic characteristics and uncertainty in embedded software system modeling [8].

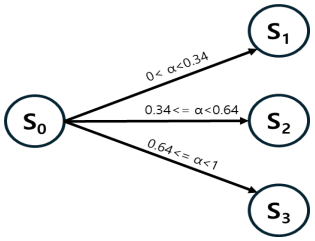


Fig. 3. Stochastic-based State Diagram

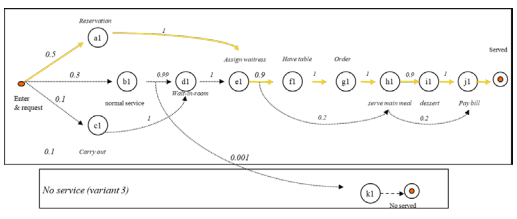
2.4 Scenario-based Integration Testing

In object-oriented software development, the adaptive use case methodology for improving the efficiency of scenario-based integration testing integrates the software design, development, and testing processes through a series of algorithmic transformations, generating test plans at the design stage [9].

Index	Action Unit Name
A1	Action 1
B1	Action 2
C1	Action 3
D1	Action 4
⋮	⋮

Use Case Scenarios \ Action Units		A1	B1	C1	D1	...
A Scenario	Scenario 1	1	3	4	5	...
	Scenario 2	1		2	3	...
	Scenario 3		1		2	...

Fig. 4. Use Case Action Matrix



Weighted value: 1 1 1 1 1 1 1 1 1 1 1 = 11 $\prod \frac{Si*Pi}{P}$

	a1	b1	c1	d1	e1	f1	g1	h1	i1	j1	k1	Weighted value created	Total amount of probability of occurrence
Main path (Reserv.)	1				2	3	4	5	6	7		7	0.5*1 * 0.9*1 * 0.9*1 =0.405
Variant 1 (normal)		1		2	3	4	5	6	7	8		8	0.3*1*0.9*1 * 0.9* 1 =0.243
Variant 2 (Carryout)			1	2	3			4		5		5	0.1*1 * 0.1*1 * 0.1*1=0.001
Variant 3 (no service)		1									2	2	0.3*1 * 0.001*1 =0.0003

Fig. 5. Action Matrix for Test Plan

The Use Case Action Matrix explains executable scenarios composed of 'action units'. This clarifies the execution path by numerically indicating specific operational steps within a scenario. **Fig. 4** is an example of a Use Case Action Matrix. Based on the list extracted from **Fig. 4**, an Action Matrix is created, and the Graph is verified and improved. Additionally, applying Musa's operational profile concept to optimize the scenario execution order enhances test productivity by utilizing software test metrics and effectively provides test prioritization at the design stage [10].

3. Scenario-based modeling in AI Software Validation

This chapter introduces the Scenario-Based Model (SBM) for validating AI software and proposes a 6-step procedure for a scenario-based testing methodology. For explanatory purposes, the method is described based on an AI system utilizing a reinforcement learning model.

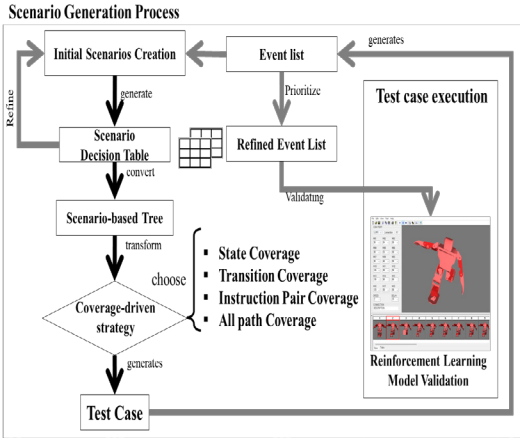


Fig. 6. Scenario-based Test Case Generation Mechanism for validating AI Software

First, diverse operating environment scenarios for validating AI software are defined, as shown in **Fig. 6**. Next, these scenarios are refined using a decision table to clarify logical branching points and condition-action relationships. The refined scenarios are then transformed into a tree structure to visualize complex decision-making flows. In this tree structure, the 'states' of the reinforcement learning agent become nodes, and 'actions' become edges. Subsequently, test cases are generated based on this tree structure and

probability values, and the generated test cases are converted into sequential event lists executable in a simulation environment. Finally, the generated sequential event list is applied to the actual simulation environment to confirm whether the reinforcement learning model selects the correct action unit under specific conditions. Furthermore, the AI software equipped with the reinforcement learning model is verified by checking whether the scenario leads to an expected 'successful termination' or an undesirable outcome (failure).

3.1 Scenario-Based Model (SBM)

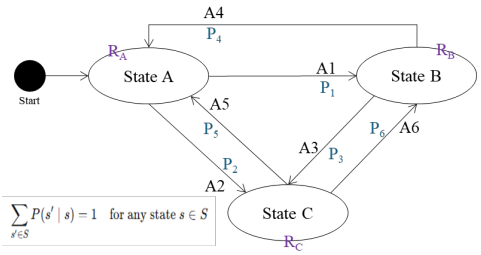


Fig. 7. Scenario-Based Model (SBM)

(6) $SBM=\{Scenarios\}, Scenarios = \{S,A,P,R,\gamma\}$

(7) $s \in S, S = \{s_0, s_1, \ldots, s_n, \ldots, s_{goal}, s_{failure}\}$//state

(8) $S_{failure} = \neg reach\ S_{goals}$, where $\gamma = -1$

(9) $a \in A, A = \{a_0, a_1, \ldots, a_{n+1}, a_n\}$ //action

(10) $P(s'/s, a) = P[S_{t+1}=s'/S_t=s, A_t=a]$

(11) $R(s, a) = E[R_{t+1}/S_t=s, A_t=a]$

(12) γ is an integer such that $(-1 \leq \gamma \leq 1)$ and $\gamma \neq 0$.

(13) $G_{A \to B} = E[Reward\ for\ State_{A \to B}] = (P_{AB}) \cdot (R_A + \gamma R_B)$

The SBM for the validation of AI software is defined as *Eq. (6)*. In **Fig. 7**, S is the set of states, structured as shown in *Eq. (7)*, and it includes initial states ($S_{initial}$), intermediate states ($S_{intermediate}$), goal states (S_{goals}), and failure states (S_{fails}). A represents the set of actions that AI software can perform in a specific state and is expressed as in *Eq. (9)*. *Eq. (10)* signifies the state-action probability that the next state will be s′ when action a is performed in the current state s. *Eq. (11)* denotes the expected *(E)* immediate reward or penalty when action a is performed in state s; specifically, positive rewards (+) are assigned to actions leading to Sgoals, and negative rewards *(penalties, -)* are assigned to actions leading to S_{fails}. Lastly, γ is the Discount Factor, which is an integer such that $(-1 \leq \gamma \leq 1)$ and $\gamma \neq 0$, as expressed in *Eq. (12)*. This value indicates how much importance is given to the present value of future rewards. It is set as an integer, including negative values, based on SBM's specific purpose (failure) to emphasize or avoid certain outcomes. In a scenario, failure is defined as the AI software never reaching the goal state (S_{goals}), which can be expressed as in *Eq. (8)*. Finally, the expected reward for the path E(State A → State B) is represented as in *Eq. (13)*.

3.2 AI Software Validation Mechanism with SBM

Step 1. Creation AI Software Scenario for Testing

In Step 1, various anticipated operating environment scenarios are defined for the functional verification of AI software equipped with reinforcement learning models.

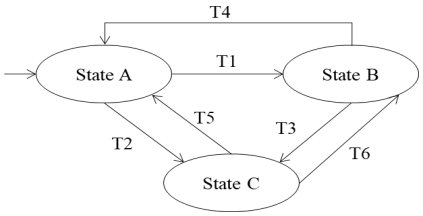


Fig. 8. Initial State Diagram

A scenario describes the process of achieving a system's functional goals and visually represents the interactions between the system and its external environment as a diagram, as shown in **Fig. 8**. Each scenario elaborates on the environment, states, and action-based system interactions necessary to achieve the software's goals.

Step 2. Refine Scenarios with Decision Table

State / Event	①State A		State B		State C	
	state	visited	state	visited	state	visited
T1	②State B	√	N/A	x	N/A	x
T2	⑤State C	√	N/A	x	N/A	x
T3	N/A	x	④State C	√	N/A	x
T4	N/A	x	⑤State A	√	N/A	x
T5	N/A	x	N/A	x	⑥State A	√
T6	N/A	x	N/A	x	⑦State B	√

Fig. 9. Initial Scenario Diagram

In Step 2, the scenarios generated in Step 1 are refined based on logical branching points and the interaction relationships of each state, utilizing a decision table as shown in **Fig. 9**. This method helps to structure the system's internal decision-making logic and define system behavior according to various conditions.

Step 3. Convert Scenario-based Tree with Graph

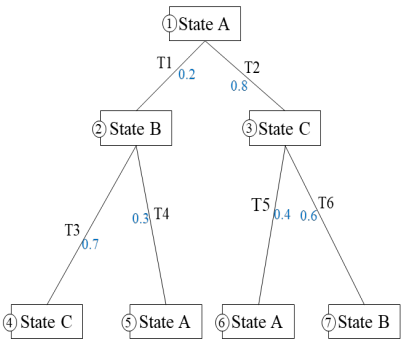
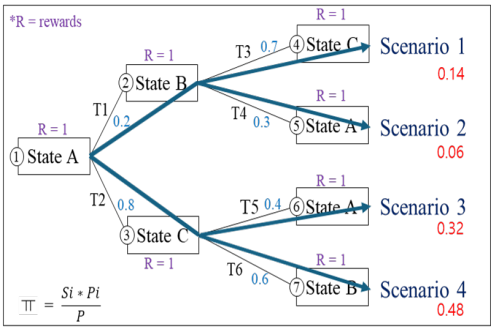


Fig. 10. Scenario Transition Tree

In Step 3, the refined scenarios are transformed into a tree structure that includes common starting points and branching points. Each node of this tree represents a 'state' of the Reinforcement Learning (RL) Agent that constitutes the scenario, and the edges represent 'actions' between the Agent's states. Additionally, as shown in **Fig. 10**, the Stochastic method is used to denote frequency-based probability values for 'action' occurrences on these edges, explicitly indicating the agent's action selection probability in uncertain environments.



	Action (Event) T1	Action (Event) T2	Action (Event) T3	Action (Event) T4	Action (Event) T5	Action (Event) T6	Total amount of probability of Occurrence
Scenario 1	1		1				0.2*1*0.7*1 = 0.14
Scenario 2	1			1			0.2*1*0.3*1 = 0.06
Scenario 3		1			1		0.8*1*0.4*1 = 0.32
Scenario 4		1				1	0.8*1*0.6*1 = 0.48

Fig. 11. Scenario-based Test Case

Fig. 11 shows a scenario-based transition tree with the expected rewards calculation for each scenario. This tree structure not only visually clarifies the various action sequences that an RL system can perform and their results, but also helps optimize the testing of RL-based AI software by allowing the identification of the importance of each action choice.

Step 4. Generate Scenario-based Test Cases

Based on the tree structure transformed in Step 3 and the probability values assigned to each edge, test cases are generated, as shown in **Fig. 12**. These test cases include initial conditions, input values, and expected system behaviors and results, enabling the reproduction of a specific scenario in a simulation environment.

TCID	Initial State	Event	Action	Next State	Event	Action	End
TC1	S_0	T1	Action 1	S_2	T2	Action 2	S_4
TC2	S_1	T2	Action 2	S_3	T3	Action 3	S_5
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 12. Scenario-based Test Case

Step 5. Generate a Sequential Event List with Test Cases

Step 5 transforms the Test Cases defined in Step 4 into a 'sequential event list' executable in a simulation environment, as shown in **Fig. 13**.

No.	Test Case	Event/Action	Time(ms)	Repeat	Pass/Fail
1	TC1	Action 1	1000	1	P
2	TC1	Action 2	1000	2	P
3	TC2	Action 3	1000	1	P
4	TC3	Action 4	1000	3	F
⋮	⋮	⋮	⋮	⋮	⋮

Fig. 13. Scenario-based Event List

This list assists in test execution by thoroughly defining the occurrence of external environmental events over time and the AI Software model's response to them.

Step 6. Validate AI software with Sequential Event List

The generated sequential event list is applied to the actual simulation environment to validate the AI software, which is equipped with a

reinforcement learning model. In this process, it is confirmed whether the agent selects the correct 'action unit' under specific conditions, and whether the scenario leads to an expected 'successful termination (success)' or an 'undesirable outcome (failure)'.

4. Conclusions

This paper proposes an SBM validation mechanism for AI software to address the difficulties in AI software validation. The proposed mechanism defines an SBM method by adopting the concept of a Markov Decision Process and Scenario-based Integration testing[9], and explicitly classifies the state set into initial, intermediate, goal, and failure states. It extends the discount factor (γ) to integers to provide specialized modeling for validation. It is expected to enhance the AI software validation process by developing automation techniques that reduce manual effort and address resource constraints in scenario definition and transformation, as well as by establishing coverage measurement metrics for AI software validation. We hope that our method can contribute as a foundation for the validation of AI software.

References

[1] C. Tao, J. Gao and T. Wang, "Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices," *IEEE Access*, vol.7, pp.120164-120175, 2019. doi: 10.1109/ACCESS.2019.2937107.

[2] J. Gao, C. Tao, D. Jie and S. Lu, "Invited Paper: What is AI Software Testing? and Why," in *Proc. of 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp.27-2709, San Francisco, CA, USA, 2019. doi: 10.1109/SOSE.2019.00015.

[3] S. Jo, R. Kwon, and K. Kwon, "Safety assessment method of reinforcement learning model: Autonomous driving case study," *Journal of the Korea Information Technology Society*, vol.21, no.8, pp.165-174, 2023. doi: 10.14801/jkiit.2023.21.8.165

[4] A. J. Singh, & A. Easwaran, "Pas: Probably approximate safety verification of reinforcement learning policy using scenario optimization," in *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp.1745-1753, May 2024.

[5] Edward F. Moore, "Gedanken-experiments on Sequential Machines," *Automata Studies, Annals of Mathematical Studies*, no.34, pp.129-153, 1956.

[6] G. H. Mealy, "A Method for Synthesizing Sequential Circuits," *Bell System Tech Journal*, vol.34, 1955.

[7] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol.2, pp.331-434, 1990.

[8] S. Moon, "A Study on Modeling and Simulation for Embedded Software Systems" *(Master's Thesis)* Graduate School, Hongik University, Seoul, 2007.

[9] R. Y. Kim & C. R. Carlson, "Scenario-based integration testing for object-oriented software development." in *Proc. Eighth Asian Test Symposium (ATS'99)*, pp.283-288, IEEE, 1999.

[10] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE software*, vol.10, no.2, pp.14-32, 1993.