УДК 681.5:004.4'22:004.4'242 https://doi.org/10.26160/2309-8864-2024-17-169-172

АЛГОРИТМ СИНТЕЗА ТЕСТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Гуляев С.С.
АО «Конструкторское бюро промышленной автоматики»;
Федеральный исследовательский центр «Саратовский научный центр Российской академии наук», Саратов

Ключевые слова: алгоритм синтеза тестов, формальная верификация, мобильные роботы, конечные автоматы, программное обеспечение, UML, SysML.
Аннотация. В данной статье предлагается алгоритм синтеза тестов для формальной верификации программного обеспечения. Рассматриваются ключевые этапы алгоритма, включая преобразование UML- и SysML-диаграмм в конечный автомат, определение критериев покрытия теста, генерацию тестовых сценариев и оптимизацию тестового набора. Приведен практический пример использования алгоритма для верификации системы управления мобильным роботом. Результаты экспериментов подтвердили эффективность предложенного метода, обеспечив полное покрытие состояний и переходов, минимизировав набор тестов и выявив ошибки на ранних стадиях разработки.

SOFTWARE TEST SYNTHESIS ALGORITHM

Gulyaev S.S.
SC «Design Bureau of Industrial Automatics»;
Federal Research Center «Saratov Scientific Center of the Russian Academy of Sciences», Saratov

Keywords: test synthesis algorithm, formal verification, mobile robots, finite state machines, software, UML, SysML.
Abstract. This article proposes an algorithm for synthesizing tests for formal software verification. The key stages of the algorithm are considered, including the conversion of UML and SysML diagrams into a finite state machine, the definition of test coverage criteria, the generation of test scenarios and the optimization of the test suite. A practical example of using the algorithm to verify the control system of a mobile robot is given. The experimental results confirmed the effectiveness of the proposed method, providing full coverage of states and transitions, minimizing the set of tests and identifying errors at early stages of development.

Введение. Программное обеспечение реального времени (ПО РВ) играет ключевую роль в управлении критически важными системами, такими как мобильные роботы, автономные транспортные средства и промышленные установки. Его надежность зависит от точности и корректности, поэтому необходимы строгие методы верификации и тестирования. Популярным методом является формальная верификация через математическое доказательство правильности программы. Один из ключевых подходов – моделирование поведения системы с помощью конечных автоматов, поддерживаемое инструментами вроде IBM Rhapsody с UML и SysML [1, 2].

Синтез тестов для программного обеспечения реального времени является важной частью процесса разработки и верификации таких систем. Существует множество подходов и методов, которые применяются для генерации тестов

(метод случайного тестирования, тестирование на основе моделей, генерация тестов на основе покрытия, символьное исполнение, динамическое тестирование), каждый из которых имеет свои достоинства и недостатки.

Формальная модель тестируемого ПО. Для формальной верификации программного обеспечения реального времени необходима точная и формализованная модель поведения системы. Одной из наиболее удобных и широко используемых формальных моделей являются конечные автоматы [3]. Конечные автоматы представляют систему как набор дискретных состояний и переходов между этими состояниями, вызванных внешними событиями или внутренними условиями.

После преобразования UML- или SysML-модели в конечный автомат можно приступить к формальной верификации системы. Для этого используются методы проверки моделей (model checking), которые позволяют проверить, удовлетворяет ли система определенным свойствам, таким как безопасность, живучесть и доступность [4]. Методы проверки моделей включают в себя проверку достижимости состояний, проверку отсутствия тупиков и проверку инвариантов.

Алгоритм синтеза тестов. Предлагаемый алгоритм синтеза тестов предназначен для автоматической генерации тестовых сценариев на основе формальной модели программного обеспечения реального времени, представленной в виде конечного автомата. Целью данного алгоритма является обеспечение полного покрытия всех возможных состояний и переходов системы, что позволяет максимально точно оценить корректность её работы. Шаги алгоритма.

1. Преобразование UML- и SysML-диаграмм в конечный автомат. Процесс начинается с извлечения информации из UML- и SysML-диаграмм. Для этого используется специализированное программное обеспечение, такое как IBM Rhapsody, которое позволяет экспортировать модели в формат, пригодный для дальнейшей обработки. Затем производится построение графа состояний и переходов, который служит основой для последующих этапов.

2. Определение критериев покрытия теста. Критерии покрытия определяют, какие аспекты системы должны быть проверены в процессе тестирования. Обычно выбираются следующие критерии: а) покрытие состояний: каждый узел графа (состояние) должен быть посещен хотя бы один раз; б) покрытие переходов: каждая дуга графа (переход) должна быть пройдена хотя бы один раз; в) комбинаторное покрытие: проверка всех возможных комбинаций состояний и/или переходов.

3. Генерация тестовых сценариев. На основе графа состояний и определённых критериев покрытия генерируются тестовые сценарии. Для этого используются алгоритмы обхода графов, такие как поиск в глубину (DFS) или поиск в ширину (BFS). Важно обеспечить, чтобы каждый сценарий обеспечивал достижение необходимого покрытия.

4. Оптимизация тестового набора. Оптимизация включает удаление дублирующихся или избыточных тестовых случаев, а также комбинирование нескольких тестов в один, чтобы уменьшить общее количество запусков. Это

улучшает эффективность тестирования, сокращая временные и ресурсные затраты.

Пример применения. Для демонстрации практической ценности предложенного подхода рассмотрим пример применения нашего алгоритма синтеза тестов для формальной верификации программного обеспечения реального времени. Задача: Проверка корректности работы системы управления мобильным роботом, выполняющего задачи навигации и избегания препятствий.

Система управления мобильным роботом включает три основных компонента: навигационный модуль (отвечает за планирование маршрута и определение текущего местоположения робота); модуль обхода препятствий (обнаруживает препятствия и рассчитывает траекторию движения, избегающую столкновений); контроллер двигателя (управляет двигателями робота, выполняя команды движения).

Для тестирования данной системы мы создаем UML-модель, отражающую взаимодействие модулей. Далее она преобразуется в конечный автомат, который используется для генерации тестов.

Алгоритм синтеза тестов генерирует следующие тестовые сценарии:

1. searching → detect_obstacle → avoiding_obstacle → returning_to_route → searching;

2. searching → search → detect_obstacle → avoid_obstacle → return_to_route → return_to_search,

где *searching* – поисковая фаза (робот ищет возможные препятствия в окружающей среде); *detect_obstacle* – обнаружение препятствия (робот распознает появление препятствия и инициирует реакцию); *avoiding_obstacle* – избегание препятствия (после обнаружения препятствия робот переходит в режим избегания, стараясь обойти препятствия); *returning_to_route* – возврат к маршруту (если препятствие успешно преодолено, робот возвращается на заранее определенный маршрут); *return_mo_search* – продолжение поиска (возвращаясь в поиск, робот продолжает искать новые препятствия).

Эти сценарии гарантируют, что система правильно реагирует на внешние условия, предотвращая опасные ситуации, а тесты обеспечивают полное покрытие состояний и переходов модели, что позволяет убедиться в корректности функционирования системы.

В ходе эксперимента были получены положительные результаты, подтверждающие эффективность предложенного подхода к синтезу тестов для формальной верификации программного обеспечения реального времени. Ниже представлены основные результаты и обсуждение их значимости.

Результаты.

1. Покрытие состояний и переходов: в результате применения предложенного алгоритма было достигнуто полное покрытие всех состояний и переходов в финальном автомате, что свидетельствует о высокой степени достоверности тестов.

2. Минимальный набор тестов: был получен минимальный набор тестов, необходимый для проверки всех возможных сценариев работы системы. Это позволило сократить время и ресурсы, необходимые для тестирования.

3. Выявленные ошибки: в процессе тестирования были выявлены и исправлены несколько ошибок, которые могли привести к сбоям в работе системы. Благодаря раннему выявлению недостатков удалось повысить качество программного обеспечения.

Заключение. В заключение можно сделать вывод, что предложенный алгоритм синтеза тестов демонстрирует высокую эффективность в контексте формальной верификации программного обеспечения реального времени. Полученные экспериментальные результаты подтверждают, что использование этого подхода позволяет достичь полного покрытия состояний и переходов, минимизировать набор тестов и своевременно выявлять, и устранять ошибки на ранних этапах разработки.

Финансирование. Работа выполнена в рамках государственного задания Министерства науки и высшего образования Российской Федерации (тема №122030400209-9 Разработка интеллектуальных моделей и методов управления сложными человеко-машинными системами в условиях критических ситуаций).

Список литературы

1. Петров Д.Ю. Модельно-управляемое проектирование автоматизированной системы идентификации дефектов листового стекла // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2023. – № 4. – С. 41-48.
2. Мешалкин В.П., Чистякова Т.Б., Петров Д.Ю. Инжиниринг цифрового тренажера для обучения операторов формования листового стекла // Программные продукты и системы. – 2024. – Т. 37, №2. – С. 561-566.
3. Hopcroft J.E., Motwani R., Ullman J.D. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2006.
4. Clarke E.M., Wing J.M. Formal methods: state of the art and future directions // ACM Computing Surveys (CSUR). 1996, vol. 28, no. 4, pp. 626-643.

Сведения об авторе:
Гуляев Станислав Сергеевич – аспирант, ведущий математик.