Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной математики и компьютерных наук

КУРСОВАЯ РАБОТА

Конечно-автоматное модульное тестирование программных реализаций на языке Java

Разенков Семен Игоревич

Направление подготовки 10.05.01 Компьютерная безопасность

Направленность (профиль) «Анализ безопасности компьютерных систем»

Руководитель работы

канд. физ.-мат. наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. С. Твардовский

*подпись*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Автор работы

студент группы № 931825

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.И. Разенков

*подпись*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Томск – 2022

**ОГЛАВЛЕНИЕ**

Введение 3

1 Основные понятия и определения 4

1.1 Конечные автоматы 4

2Методы построения проверяющих тестов 4

3 Платформа JUnit5 4

Выводы

Список использованных источников и литературы

**ВВЕДЕНИЕ**

Конечно-автоматные методы широко используются в тестировании программного обеспечения, поскольку имеют гарантированную полноту покрытия неисправностей. Однако, высокая сложность (длина) этих тестов подталкивает к автоматизации тестирования. Одним из объектов тестирования являются реализации на объектно-ориентированных языках программирования, для проверки которых может быть использовано модульное тестирование.

В данной работе мы решаем задачу автоматизации модульного тестирования автоматными методами для классов на языке Java.

**1 ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ**

**1.1 Конечные автоматы**

Конечным автоматом называется пятерка *A* = (*S*, *I*, *O*, *TS*, *s*0), где *S* - конечное множество состояний c выделенным начальным состоянием *s0*, *I* - входной алфавит,*O* - выходной алфавит и *TS* – отношение переходов[1]. Четверка (*s*, *i*, *o*, *s*`) *TS*, описывает переход в автомате *A* из состояния *s* в состояние *s*` под действием входного символа *i* с выходным символом *o*.

Автомат называется наблюдаемым, если для любой тройки (*s*, *i*, *o*) , существует не более одного состояния *s*` *S* такого, что (*s*, *i*, *o*, *s*`) *TS* .

Если для каждой пары (*s*, *i*) существует хотя бы один переход (*s*, *i*, *o*, *s*`) *TS*, то автомат называется полностью определенным. В противном случае он называется частично определенным.

Автомат называется детерминированным, если для любой пары (*s*, *i*) существует не более одной пары (*o*, *s*`) такой, что (*s*, *i*, *o*, *s*`) *TS*. Иначе автомат называется недетерминированным.

В данной работе будут рассматриваться детерминированные полностью определённые конечные автоматы.

Входной последовательностью автомата называется последовательность входных символов, в которой за символом следует [2]. Записывается следующим образом: .

Выходная последовательность автомата представляет собой последовательность выходных символов, в которой за символом следует . Записывается: .

Пусть . Параназывается входо-выходной последовательностью если и только если существует последовательность состояний таких, что , …, .

Состояния *s* и *p* автоматов A1 и A2 соответственно эквивалентны, если автомат A1 в состоянии *s* и автомат A2 в состоянии *p* под воздействием любой входной последовательности выдают одинаковые выходные последовательности. Если состояния не эквивалентны, то они различимы.

Автомат называется *приведенным*, если любые два состояния в нем различимы[3], т.е. для них существует такая последовательность входных символов, при подаче которой выходные последовательности автомата, находившегося изначально в каждом из этих двух состояний, будут разными.

Пример. Рассмотрим конечный автомат, описывающий поведение муравья. Входной алфавит автомата представляет собой множество всех 3-битных чисел. Каждый разряд числа, начиная со старшего, соответственно показывает истинность утверждений: «видит муравейник», «видит листик», «видит хищника». Выходной алфавит представляет собой множество 2-битных чисел, характеризующих скорость движения муравья. Множество состояний автомата следующее: «сидит в муравейнике», «ищет листик», «тащит листик», «убегает от хищника». Сопоставим каждому состоянию соответственно числа от 0 до 3. Тогда получим следующий конечный автомат (рис. 1):

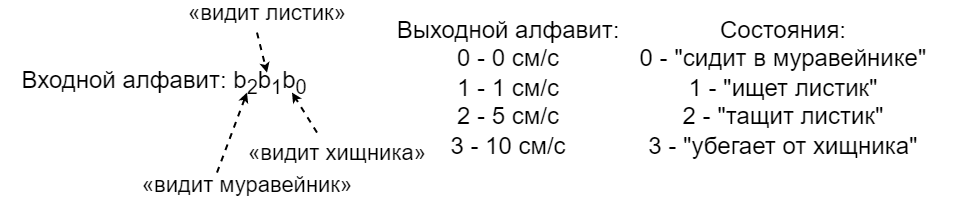
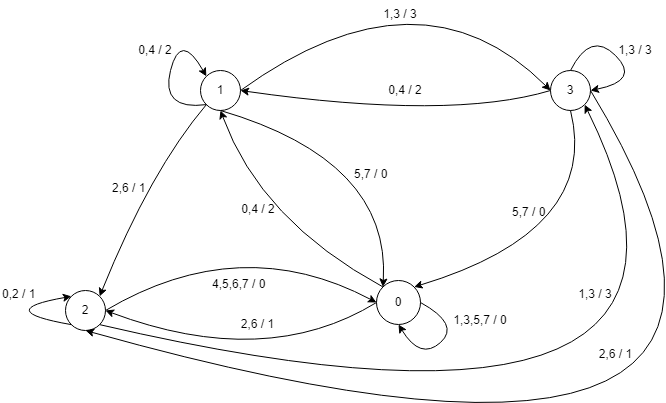


Рисунок 1. Диаграмма переходов конечного автомата.

Состояния на рисунке отмечены окружностями, представляющими собой вершины ориентированного графа, а вместе с дугами они показывают отношения переходов. Каждая дуга подписана входными значениями, вызывающими данный переход из одного состояния в другое, а также соответствующими выходными значениями.

**1.2 Проверяющие тесты**

Модель неисправности конечного автомата представляет собой тройку: эталон – конечный полностью определенный детерминированный приведённый инициальный автомат с конечным числом состояний; область неисправности – множество конечных полностью определенных детерминированных инициальных автоматов с тем же выходным алфавитом; отношение конформности – отношение эквивалентности. Отметим, что определение области неисправности может различаться в зависимости от используемого метода синтеза теста и будет более детально рассмотрено в последующих разделах.

Задание модели неисправности для того или иного метода синтеза теста позволяет формально определить и доказать гарантированную полноту покрытия неисправностей. Область неисправности задаёт тот класс ошибок, который гарантированно обнаруживается простроенным тестом.

Полным проверяющим тестом называется конечное множество входных последовательностей конечной длины, по реакциям на которые можно отличить всякую неконформную реализацию из области неисправности. Если проверяемый автомат на последовательности теста реагирует так же, как эталонный автомат, то гарантируется, что на все остальные последовательности он будет реагировать, как эталон. Длиной теста называется сумма длин всех его последовательностей.

**2 МЕТОДЫ ПОСТРОЕНИЯ ПРОВЕРЯЮЩИХ ТЕСТОВ**

Примерами методов построения проверяющих тестов являются transition tour и метод Василевского (имеет много модификаций)[4].

Передаточной последовательностью для некоторого состояния эталонного автомата *s* является последовательностью входных символов (возможно пустая), по которой можно достигнуть состояния s из начального состояния.

Множество достижимости – это множество передаточных последовательностей для всех состояний эталонного автомата.

Множество различимости представляет собой множество, состоящее их входных последовательностей, по реакции на которые можно различить любую пару состояний.

Метод transition tour представляет собой метод построения теста обходом графа переходов для автомата. Порядок его построения следующий:

1. Формируется множество достижимости, т.е. для каждого состояния *s* автомата *A* cтроится передаточная последовательность α*s*
2. Для каждого состояния *s*, добавляем в тест все последовательности α*si*, где *i*∈*I.*

Однако, у данного алгоритма есть недостаток, заключающийся в отсутствии гарантии покрытия ошибок переходов.

Построение теста методом Василевского осуществляется в следующем порядке:

1. cтроится передаточная последовательность α*s* для каждого состояния *s* автомата *A* (т.е. множество достижимости)
2. Для каждого состояния *s*, добавляются в тест все последовательности α*si*, где *i*∈*I*∪ε, ε - пустой символ (покрытие переходов).
3. К каждой последовательности теста добавляется диагностическая (различающая) последовательность автомата *A* из множества различимости.

Этот метод является полным проверяющим тестом и позволяет обнаружить все ошибки из области неисправности.

**3 JUNIT-ТЕСТЫ**

**3.1 Платформа JUnit 5**

JUnit 5 представляет собой программный комплекс для тестирования программного обеспечения на языке Java [5]. Он состоит из следующих модулей: JUnit Platform, JUnit Jupiter и JUnit Vintage.

JUnit Platform предоставляет программный интерфейс для разработки тестов и предназначен для их запуска на виртуальной машине Java (JVM). В данном модуле есть дополнительное приложение для запуска тестов из командной строки, которое было использовано в данной работе. JUnit Platform поддерживается многими средами разработки ПО.

JUnit Jupiter предоставляет некоторые обновленные и дополнительные возможности для написания тестов, например, дополнить тесты доступом к какому-либо классу. JUnit Vintage позволяет запускать тесты, разработанные для JUnit 3 и JUnit 4.

* 1. **Реализация JUnit-теста**

Предполагается, что тестируется реализация некоторой системы, работа которой представима в виде инициального конечного детерминированного конечного автомата, на языке программирования Java. JUnit-тесту в виде текстовых файлов подаются на вход: название класса, реализующего систему (т.е. автомат), таблица соответствия входных символов автомата методам класса с соответствующими наборами аргументов, таблица соответствия выходных символов автомата значениям возвращаемых методами класса объектов, проверяющие тесты для автомата, построенные произвольным методом. Кроме того, необходимо наличие скомпилированного Java-класса (файл .class), реализующего необходимый конечный автомат.

Комплекс для тестирования состоит из Console Launcher для JUnit Platform, скомпилированного Java-класса, осуществляющего тестирование, а также нескольких вспомогательных Java-классов, необходимых для хранения информации об автоматном тесте.

Программа тестирования запускается из командной строки с помощью shell-скрипта, что делает её кроссплатформенной. Сначала она с помощью системного загрузчика классов ClassLoader [6] путем вызова метода loadClass динамически подгружает скомпилированный заранее Java-класс, реализующий автомат. Затем из упомянутых выше текстовых файлов считывает информацию, необходимую для тестирования реализации.

JUnit позволяет создавать генераторы тестов, которые создают unit-тесты (или контейнеры с множеством unit-тестов) динамически, во время выполнения JUnit теста. В данной работе этот функционал использован для генерации контейнеров unit-тестов, каждый из которых соответствует тестовой последовательности для проверки конформности реализации автомата. Каждый тест внутри контейнера соответствует одному переходу в автомате (входной/выходной символ, следующее состояние). Для каждого контейнера создается новый объект – экземпляр тестируемого автомата. В случае несоответствия значения возвращаемого объекта нужному выходному символу автомата программа выдаст ошибку и информацию о том, в какой тестовой последовательности и на каком входном символе было обнаружено несовпадение поведения реализации автомата и эталона.

Использована экспериментальная на момент выполнения данной работы возможность распараллеливать выполнение контейнеров unit-тестов (или просто unit-тестов), реализуемая путем настраивания специальных параметров JUnit. Во время выполнения тестирования параллельное тестирование последовательностей позволило сократить время выполнения тестов на 10%, что позволяет сделать вывод о пользе данной возможности для проведения тестирования программных реализаций конечных автоматов.

**ВЫВОДЫ**

Разработан и протестирован способ автоматизации конечно-автоматного тестирования программных реализаций на языке Java с помощью комплекса JUnit 5. Созданная программа позволяет обнаруживать все неисправности, доступные для используемого при этом метода построения проверяющих тестов. Выполнение тестовых последовательностей в параллельном режиме позволяет ускорить тестирование.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ**

1 Евтушенко Н.В. Недетерминированные автоматы: анализ и синтез. Ч. 2: Решение автоматных уравнений: учеб. пособие / Н. В. Евтушенко, М. В. Рекун, С. В. Тихомирова. – Томск: Томский государственный университет, 2009. – 111 с.

2 Гилл А. Введение в теорию конечных автоматов / А. Гилл. – М. : Издательство Наука, 1966. – 272с.

3 Агибалов Г. П. Лекции по теории конечных автоматов / Г. П. Агибалов, А. М. Оранов. – Томск : Издательство ТГУ, 1984. – 185 с.

4 M. P. Vasilevskii, "Failure diagnosis of automata," Kibernetika (Transl.), no.4, pp. 98-108, July-Aug. 1973.

5 JUnit 5 User Guide – URL: <https://junit.org/junit5/docs/current/user-guide/#writing-tests-dynamic-tests>

6 Java™ Platform Standard Edition 8 API Specification – URL: https://docs.oracle.com/javase/8/docs/api/java/lang/ClassLoader.html