МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ОПТИКИ АТМОСФЕРЫ СО РАН им. В.Е. ЗУЕВА

НОВЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ИССЛЕДОВАНИИ СЛОЖНЫХ СТРУКТУР

МАТЕРИАЛЫ
ДВЕНАДЦАТОЙ КОНФЕРЕНЦИИ С МЕЖДУНАРОДНЫМ УЧАСТИЕМ
4–8 июня 2018 г.

Мероприятие проведено при финансовой поддержке
Российского фонда фундаментальных исследований (проект № 18-07-20033)

Томск
Издательский Дом Томского государственного университета
2018

Секция 6. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ИССЛЕДОВАНИИ ДИСКРЕТНЫХ СТРУКТУР

ТЕСТИРОВАНИЕ JAVA ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТА FSMTEST2JUNIT*

С.В. Батрацкий, С.А. Прокопенко, М.Л. Громов, Н.В. Шабалдина

Национальный исследовательский Томский государственный университет, Томск, Россия
pride080993@gmail.com, s.prokopenko@sibmail.com, maxim.leo.gromov@gmail.com, nataliamailbos@mail.ru

Тестирование программных реализаций является актуальной задачей, поскольку данные реализации могут быть не лишены различного рода ошибок, возникающих в процессе программирования. Поэтому в данной работе мы обращаемся к задаче обнаружения ошибок в программных реализациях, написанных на языке Java. Разработчики программ при тестировании своих продуктов, как правило, ограничиваются проверкой некоторых условий, например, что значения некоторой переменной не выходят за границы области допустимых значений типа данной переменной. Такая проверка позволяет обнаружить некоторые ошибки в программах, но полноту теста оценить трудно.

В работе мы предлагаем другой подход к тестированию программных реализаций. Его идея заключается в следующем. Сначала по программе строится математическая модель, так называемый расширенный автомат [1]. Поскольку методы синтеза тестов для расширенных автоматов недостаточно развиты, то поведение расширенного автомата моделируется на последовательностях определенной длины и строится эквивалентный конечный автомат [2]. Затем для конечного автомата строится проверяющий тест одним из методов, которые содержатся в пакете прикладных программ FSMTest-2.0 [3, 4], разработанном для анализа и синтеза конечных автоматов. Далее построенный тест с помощью плагина FSMTest2JUnit для среды разработки eclipse подается на программную реализацию. Встроенный в eclipse инструмент JUnit сравнивает реакции реализации с эталонными реакциями на последовательности теста и выносит вердикт, обнаруживает ли построенный тест ошибки в программной реализации.

Для тестирования мы выбрали программу ndfsm_test на языке Java пакета FSMTest-2.0. Состояниям расширенного автомата, описывающего поведение класса, поставлены в соответствие наборы значений полей экземпляра класса, а входо-выходным парам, помечающим переходы, – методы класса и значения, возвращаемые вызванным методом. Поскольку программа ndfsm_test имеет один класс, и мы рассматривали лишь методы getInputs, getOutputs, addState, addInput, то расширенный автомат, построенный для ndfsm_test, имеет одно состояние и четыре перехода. По расширенному автомату путем моделирования на входных последовательностях длины 3 строится конечный автомат. Полученный конечный автомат имеет 3 состояния, 9 переходов и является неполностью определенным. Проверяющий тест для конечного автомата построен HSI методом, программная реализация которого включена в пакет FSMTest-2.0, и тест состоит из 10 входо-выходных последовательностей. Затем при помощи плагина FSMTest2JUnit построенный тест конвертируется в соответствующий JUnit тест. Встроенный в eclipse инструмент JUnit проверяет программу ndfsm_test.

В результате проведенной проверки установлено, что в методах getInputs и getOutputs содержатся ошибки. Оказалось, что, вопреки ожиданиям, те методы, которые должны менять поля экземпляра класса (менять состояние соответствующего автомата) этого не делают, поскольку данные поля фиксируют свои значения после вызова конструктора класса и более в течение жизни объекта не меняются.

Данная ошибка не обнаруживается при стандартном сценарии использования JUnit, поскольку этот сценарий предполагает вызов одного и того же метода класса на различных наборах входных параметров (граничное тестирование). Использование формальных моделей для тестирования позволило сформировать последовательность вызовов *различных* методов с различными параметрами, приводящую к обнаружению ошибки.

Литература

1. *Petrenko A.*, *Boroday S.*, *Groz R.* Confirming configurations in EFSM testing // IEEE Trans. Software Eng. 2004. № 30(1). P. 29–42.
2. *Гилл А.* Введение в теорию конечных автоматов. М. : Наука, 1966. 272 с.
3. *Shabaldina N.*, *Gromov M.* FSMTest-1.0: a manual for researches // Proceeding IEEE East-West Design & Test Symposium. 2015. P. 216–219.
4. *Батрацкий С.В.*, *Белых В.С.* К программной реализации пакета прикладных программ «FSMTest-2.0» // Труды Четырнадцатой Всероссийской конференции студенческих научно-исследовательских инкубаторов. Томск, 17–18 мая 2017 г. / под ред. В.В. Дёмина. Томск : Изд-во НТЛ, 2017. 132 с.

* Работа выполнена при поддержке Российского научного фонда, проект № 16-49-03012.