СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

– КА - конечный автомат

– ООП - объектно-ориентированное программирование

– JVM - Java Virtual Machine

– MBT - Model Based Testing

– PBT - Property Based Testing

– ПО - программное обеспечение

– СА - статический анализ

– ДА - динамический анализ

ТЕРМИНОЛОГИЯ

– SMT-решатель

– условия корректности

ВВЕДЕНИЕ

С развитием области анализа программ при решении большого количества задач в этом направлении разработчики все чаще стали сталкиваться с необходимостью использования формальных спецификаций. Формальная спецификация - описание поведения программы на специальном или ее исходном языке, включающее в себя такие детали как пред и пост условия вызовов, состояния и переходы между ними, что и делает спецификации идеальным кандидатом для применения в области анализа ПО. Например, спецификации не заменимы в символьном исполнении. Там они используются для аппроксимации поведения внешних библиотек или сложных частей программы, тем самым ускоряя анализ или вовсе делая его возможным в определенных местах. Реализацию данного подхода можно увидеть в USVM[ССЫЛКА] и UtBot[ССЫЛКА]. Еще один пример использования формальных спецификаций - Taint анализ, когда в них отмечаются потенциальные места ввода и утечки информации. Наличие подобных данных позволяет анализаторам сфокусироваться на проверке размеченных мест, представляющих возможные ошибки, тем самым сильно повышая эффективность[ЕСТЬ ЛИ ССЫЛКА НА ПРОЕКТ?]. Кроме этого, спецификации могут использоваться в MBT и PBT в качестве тестового оракула, отвечая на вопросы о корректности состояния ПО и переходов между ними, а также о том, удовлетворяют ли входные и выходные данные для различных методов соответствующим предикатам.

При всем этом формальные спецификации в общем случае не поставляются с программными библиотеками или другим ПО, что заставляет задуматься о способах их создания. Полностью ручное составление спецификаций

это довольно монотонная работа, при этом требующая от человека высокой квалификации в области разработки и анализа программ. Однако составление спецификаций можно частично автоматизировать.

Часть формальной спецификации можно найти в исходном коде библиотек или ПО. Если приводить в пример языки в парадигме ООП, то это классы и интерфейсы программы, их поля и сигнатуры методов. Другую же часть спецификации, описывающую поведение программы, а именно состояния и переходы между ними, можно попытаться извлечь из реальных примеров использования.

Именно автоматизации извлечения поведенческих моделей библиотек посвящена данная работа. В рамках ее выполнения будет реализована утилита, позволяющая автоматически получать общедоступные Java проекты и с помощью методов статического и динамического анализа извлекать из них сценарии работы определенной библиотеки с целью последующего восстановления поведенческой модели в виде КА. На данный момент комплексных решений для подобных задач нет, однако имеются работы в области восстановления КА из трасс и описаны способы получения трасс. Основные задачи, решаемые в этой работе, это автоматизация, интеграция и применение на практике существующих наработок в области восстановления поведенческой модели библиотек.

Важно сказать, что предполагается ручная обработка полученных с помощью реализованной утилиты автоматов. Необходимость этого является следствием использования пользовательских репозиториев, которые не гарантируют корректного применения библиотек. Также свои ограничения накладывает статический и динамический анализ. Использование points-to анализа в статическом подходе не позволяет точно различать объекты, методы которых

вызываются, что влияет на корректность получаемых трасс. Также при использовании СА, по крайней мере в рамках данной работы, не будут предприниматься попытки обработать многопоточную работу программы. Что касается ДА, то здесь на результат могут влиять точки входа в программу. Так как трассы будут собираться из запусков имеющихся или сгенерированных тестов, невозможно гарантировать корректность начального состояния библиотеки.

В первом разделе представлен обзор существующих решений, связанных с задачами поиска проектов, извлечением из них трасс вызовов и восстановлением модели. Также в данном разделе будет уделено внимание предшествующей работе по данной теме. На основе первого раздела сделан выбор в пользу определенных подходов и решений, используемых в реализации инструмента. Во втором разделе формулируются требования к создаваемому инструменту и описываются пути решения каждой из задач, стоящих на пути реализации. Третий раздел посвящен разработке подхода. В нем будет подробно описана задуманная схема работы, способы извлечения трасс и их восстановления, детали работы с репозиториями. Четвертый раздел содержит описание реализации инструмента. Пятый раздел посвящен тестированию полученной утилиты. Тестирование заключается в сравнении получаемых автоматов с несколькими заготовленными эталонами, а также демонстрацией получаемых КА для некоторого набора библиотек. Помимо этого, будет проанализировано количество успешно автоматически собранных проектов и полученных различными методами трасс. В заключении проведен анализ полученных результатов, отмечены преимущества и недостатки предложенного подхода, а также рассмотрены пути развития.

1 АНАЛИЗ РЕШЕНИЙ В ОБЛАСТИ ВОССТАНОВЛЕНИЯ ПОВЕДЕНЧЕСКИХ МОДЕЛЕЙ

При изучении предметной области было выявлено, что на текущий момент нет исследований и утилит, рассматривающих проблему извлечения поведенческих моделей библиотек включая все этапы задачи: получение самих проектов, извлечение из них трасс и восстановление модели. При этом непосредственно задаче восстановления КА посвящено не мало работ и для ее решения предложены определенные алгоритмы. При этом в подобных работах проекты для извлечения трасс представлены как исходные данные, а методу получения трасс уделено сравнительно небольшое внимание. В связи с этим, в данном разделе вместо сравнения работ схожих с реализуемой будет приведен анализ решений для подзадач, из которых состоит данная работа. Подробно будут разобраны методы извлечения трасс и методы восстановления поведенческих моделей в виде КА. Проблема получения и обработки подходящих для анализа проекта инженерная и требует частной реализации, поэтому она будет подробно разобрана в разделе 2.

1.1 Критерии сравнения

1.2 Методы извлечения трасс

Глобально методы можно поделить на статические и динамические. Первые подразумевают анализ исходного кода или байт-кода программы без его запуска. Динамические методы наоборот, предполагают запуск анализируемой программы. Статические методы уступают в точности, однако позволяют покрыть все возможные пути исполнения программы. Это позволяет находить

ошибки в тех участках кода, которые не покрытых тестами и до которых исполнение не доходит при штатной работе программы. Динамические методы, в свою очередь, за счет реального исполнения обеспечивают точность получаемых результатов, но с их помощью сложно получить все возможные состояния программы. Для восстановления поведенческой модели мы заинтересованы в получении как можно большего количества трасс, чему сопутствует использование статических методов, но в тоже время ошибки в трассах неизбежно приведут к ошибкам в модели. Рассмотрим, как решают проблему извлечения трасс в исследованиях, связанных с восстановлением спецификаций библиотек.

В работе[1]

Тест[2]

ТЕст2[1]

1.3 Сравнение методов восстановления КА

1.4 Результаты анализа

2 ПУТИ РЕШЕНИЯ

3 ПРОЕКТИРОВАНИЕ

4 РЕАЛИЗАЦИЯ

5 ТЕСТИРОВАНИЕ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Shoham S., Yahav E., J. S.F. Static Specification Mining Using Automata-Based Abstractions // IEEE Transactions on Software Engineering. 2008. т. 34, № 5. cc. 651–666.
2. Walkinshaw N., Taylor R., Derrick J. Inferring Extended Finite State Machine models from software executions // 20th Working Conference on Reverse Engineering (WCRE). 2013. cc. 301–310.