сов необходимо специальным образом составить файл-шаблон в формате XML. После этого подать его на вход программе-генератору инструментируемых библиотек. Эти библиотеки создаются в результате обхода дерева описания ресурсов с созданием кода, который будет встроен в анализируемую программу. Далее с помощью штатного компилятора происходит «вживление» инструментируемого кода. Затем следует запуск программы, в результате которой все функции, которые работают с описанными ресурсами, будут перехвачены, все входные и выходные параметры проанализированы. Результат работы протоколируется в файл или отсылается программе-анализатору. Данная программа на основании полученных протокольных сообщений генерирует отчёт для разработчика об обнаруженных дефектах в программе, что позволяет ему их устранить.

Метод анализа реализован для прикладных программ на языке Си под системы POSIX и Windows, однако при необходимости может быть расширен на Си++, а также адаптирован для ядер операционных систем.

УДК 004.94

О ПОДХОДЕ К ГЕНЕРАЦИИ ДАННЫХ ДЛЯ ТЕСТИРОВАНИЯ ПРИКЛАДНОГО И СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МЕТОДОМ ФАЗИНГА[1]

А. Н. Макаров

Для тестирования надежности системного и прикладного программного обеспечения (ПО) часто применяют метод, получивший название фазинг (fuzzing). Фазинг является одним из видов стрессового тестирования, потенциально позволяющий выявлять ошибки в процессе функционирования ПО [1]. Существует много различных инструментальных средств тестирования методом фазинга, являющихся как коммерческими продуктами, так и свободно распространяемыми. Интерес к тестированию методом фазинга обусловлен все возрастающими требованиями к надежности ПО.

Привлекательность метода заключается в простоте реализации, возможности проведения тестирования в автоматическом режиме, проведении тестирования без исходных кодов, а также в широкой применимости метода. Тестирование методом фазинга можно адаптировать ко многим видам ПО. Тестируемое ПО будем называть *целевым*, а входные данные, сгенерированные для целевого ПО, — *тестами*.

Тестирование посредством фазинга заключается в передаче некорректных данных целевому ПО. Основная задача в том, как описать модель данных и на ее основе создать некорректные данные, т. е. тесты. Различают два типовых подхода к генерации тестов: мутационный и генерационный фазинг. Первый подход предполагает внесение искажений в уже готовые корректные данные. Второй подход требует использования генератора данных заданного формата. Для мутационного фазинга требуется определения «места» внесения искажений, поскольку беспорядочное внесение искажений приводит к тому, что целевое ПО игнорирует тесты. Для применения генерационного фазинга требуется формальное описание структур данных, которое не всегда возможно выполнить. Ряд исследователей отдают предпочтение генерационному фазингу. В работе [2] авторы приводят результаты исследований, которые показывают преимущество генерационного фазинга перед мутационным. На примере формата файлов *png* показывается, что область покрытия тестируемого кода существенно больше при

[1]Работа выполнена при поддержке гранта МД № 2.2010.10.

использовании генерационного фазинга, чем при мутационном. На практике применение генерационного фазинга к сложноструктурированным данным является сложной задачей, поскольку требуется генерировать не только и даже не столько корректные данные, а данные, которые «незначительно» отличаются от спецификации.

Опыт применения тестирования посредством фазинга указывает на целесообразность комбинирования генерационного и мутационного фазинга для целевого ПО, обрабатывающего сложноструктурированные данные. Для целевого ПО, обрабатывающего относительно простые данные (например, файлы медиа форматов), предпочтение должно отдаваться генерационному фазингу.

Автором проводятся исследования, целью которых является создание метода формирования тестов, основанного на описании структур данных и процедур формирования входных данных (ПФВД) с использованием скриптового языка. ПФВД обеспечивают проверку корректности структур данных на основе заданных ограничений. Для описания структуры данных и ПФВД за основу был взят язык C. Подход, основанный на применении скриптового языка, оказался удобным. Достигнута хорошая масштабируемость среды тестирования, независимость среды тестирования от методов генерации тестов. В листинге 1 приведен пример каркаса скрипта, обеспечивающий поиск специфической структуры во входных данных.

```
1  // заголовки среды тестирования
2  #include "main_environment.h"
3  // генерируемая или искомая структура данных
4  struct struct_a {
5      struct software_dependence a;
6      int verify( RETVAL *retval, int cnt );
7      int modify( int cnt );
8  };
9  // алгоритмы проверки корректности и искажения структуры
10 int struct_a::verify( RETVAL *retval, int cnt ) {...}
11 int struct_a::modify( int cnt ) {...}
```

Листинг 1. Пример каркаса скрипта для среды тестирования

При генерации тестов, имеющих сложную внутреннюю организацию, возникают следующие трудности:

— написание на скриптовом языке генератора тестов становится сложной задачей, сопоставимой с созданием парсера требуемого формата данных;
— время проверки корректности самого генератора тестов резко возрастает;
— нетривиальной задачей становится создание тестов, обеспечивающих максимальное покрытие кода целевого ПО.

Опыт применения генератора тестов на основе скриптового языка выявил недостаточную выразительность средств описания внутренней организации структур данных. В связи с этим предлагается для генерации тестов расширить возможности скриптового языка, добавив дополнительную декларативную информацию. Например, при описании структур данных ввести атрибуты, в которых указывалась бы информация о внутренних связях, ограничениях и зависимостях. Целью введения данного расширения является возможность автоматической генерации входных данных, у которых были бы частично нарушены ограничения, внутренние связи и т. д.

В ряде языков существуют механизмы задания дополнительной декларативной информации (например, атрибуты в C#), но в контексте рассматриваемой задачи их воз-

можности ограничены. В связи с этим автором ведутся исследования по адаптации и применению специализированных языков (например, языка TreeDl [3]) для построения генераторов тестов. Ожидаемый результат от внедрения этого подхода заключается в появлении средства создания тестов, обладающего следующими свойствами:

— максимально возможная независимость генератора тестов от целевого ПО;
— минимальное время разработки тестов для нового целевого ПО;
— обеспечение повышения объема покрытия кода целевого ПО.

ЛИТЕРАТУРА

1. *Макаров А. Н.* Метод автоматизированного поиска программных ошибок в алгоритмах обработки сложноструктурированных данных // Прикладная дискретная математика. 2009. № 3(5). С. 117–127.
2. *Miller C., Peterson Z. N. J.* Analysis of Mutation and Generation-Based Fuzzing. `www.securityevaluators.com.` 2007.
3. `treedl.org` — сайт проекта TreeDl.

УДК 519.7

ПРОГРАММНАЯ ТРАНСЛЯЦИЯ АЛГОРИТМОВ В ПРОПОЗИЦИОНАЛЬНУЮ ЛОГИКУ ПРИМЕНИТЕЛЬНО К КОМБИНАТОРНЫМ ЗАДАЧАМ

И. В. Отпущенников, А. А. Семёнов

Интенсивный рост производительности современных вычислительных архитектур сделал возможным решение комбинаторных задач таких размерностей, которые казались непреодолимыми еще 15–20 лет назад. В связи с этим возникли новые направления в компьютерной алгебре и вычислительных отраслях дискретной математики и математической логики. Одной из наиболее актуальных в этом смысле областей является решение логических (булевых) уравнений. В последние 10 лет наблюдается существенный прогресс в разработке алгоритмов, эффективных на практически важных классах логических уравнений.

К логическим уравнениям эффективно сводятся многочисленные комбинаторные задачи. Однако при практическом осуществления соответствующих сведений приходится сталкиваться с различными препятствиями (разнородность структур данных; отсутствие идеологии, применимой к широкому классу проблем, и др.). Сказанное означает актуальность проблемы разработки многофункционального транслятора, осуществляющего сведение различных комбинаторных задач к задачам поиска решений логических уравнений. Следует отметить, что во многих имеющихся разработках данная проблема так или иначе решалась (см. [1–4]). Однако во всех этих случаях осуществлялась трансляция некоторого «узкого» класса проблем, ограниченного конкретной предметной областью. Комплекс Verilog [3] используется при описании логических схем, и все его базовые конструкции ориентированы на специалистов в схемотехнике. Комплекс ABC [4] предназначен для решения задач синтеза и верификации логических схем; схемы могут быть описаны в различных форматах, однако предполагается, что пользователь эти описания каким-либо образом получил (например, используя Verilog).

В работе излагаются общие принципы построения нового многофункционального транслятора Transalg. Данный программный комплекс позволяет осуществлять пропозициональное кодирование алгоритмов, не предполагая узкоспециальных знаний у