УДК 519.876.5

И. В. Рудаков, А. В. Ребриков

МАСШТАБИРОВАНИЕ АЛГОРИТМОВ ДЛЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ МОДУЛЬНЫХ ТЕСТОВ

Рассмотрены вопросы разработки метода масштабирования алгоритмов, необходимого для проведения верификации программных систем, обеспечивающей максимальный показатель структурного покрытия элементов системы. Приведен разработанный метод масштабирования, показано его место в процессе верификации программных систем, а также представлены основные результаты применения метода и их интерпретация.

E-mail: irudakov@yandex.ru; rebrikov_a@mail.ru

Ключевые слова: *тестирование, автоматическая генерация тестов, верификация, масштабирование алгоритмов.*

Постоянно возрастающая сложность разрабатываемых программных систем требует автоматизации процесса составления тестов, проверяющих корректность работы алгоритмов. В частности, особый интерес представляет построение регрессионных тестов и тестов, направленных на обнаружение статических ошибок, поскольку именно они позволяют выявить в среднем 90 % ошибок, вносимых в код при разработке нового или доработке старого функционала [1].

При анализе функционирования сложных алгоритмов полная, или формальная, верификация [2, 3] требует больших временны́х затрат [3], поэтому в целях их сокращения необходимо иметь возможность проведения неполной верификации [4]. Помимо сокращения временны́х затрат неполная верификация позволяет проверять системы с нецелочисленной логикой, что невозможно сделать при полной верификации.

Необходимость разработки метода для неполной верификации программных систем обусловлена отсутствием единого синтетического метода такой верификации [4]. Существующие аналоги являются узкоспециализированными, не обеспечивают требуемой степени покрытия кода или требуемого уровня автоматизации верификации [4].

При проверке выполнения функциональных требований оказывается полезным повысить уровень абстракции так, чтобы при составлении тестов вычислительные ресурсы не расходовались на проверку несущественных (с точки зрения данного функционального требования) свойств алгоритма. Эта задача решается с помощью масштабирования алгоритма, при котором задается набор событий, контролируемых в процессе составления тестов. Множество наблюдаемых событий описывается как совокупность операторов, информация о выполнении которых проверяется, и набора операндов, значения которых также должны включаться в событие.

Требуется, с одной стороны, убрать из алгоритма как можно больше операторов, не существенных для заданного уровня абстракции, а с другой

— сохранить поведение алгоритма неизменным с точки зрения его функционирования. Таким образом, необходим аппарат, ограничивающий число ненаблюдаемых операторов; в качестве такого аппарата предлагается использовать механизм зависимостей [5].

Для алгоритмов можно использовать известные зависимости по данным и управлению, определения которых даны в работе [5]. Отношение зависимости между операторами алгоритма может быть выявлено по описанию алгоритма и используется для предсказания возможного хода его выполнения. Существует два основных типа зависимостей в последовательной программе: зависимости по управлению, которые определяются управляющими конструкциями программы, и зависимости по данным, которые определяются переменными, используемыми в алгоритме.

Под алгоритмом понимается кортеж (G, Var), где $G = (N, E, n_0)$ — управляющий граф алгоритма; N — множество вершин, каждая из которых соответствует оператору алгоритма; E — множество дуг, соответствующих переходу управления в алгоритме; $n_0 \in N$ — начальная вершина алгоритма; Var — множество переменных алгоритма.

Оператор $s \in N$ зависит по управлению: от предиката c, который содержится в операторе условного ветвления; от выбора пути выполнения, на который потенциально влияет предикат c; от того, будет ли выполнен оператор s. Оператор $s \in N$ зависит по данным от оператора $s' \in N$, если данные, определяемые в s', используются в s, потенциально могут достичь оператора s через последовательность присваиваний переменных.

Если событие, отмечающее факт выполнения оператора s, входит в наблюдаемое поведение алгоритма, то множество операторов S_c, от которых зависит оператор s по управлению, также войдет в остаточный алгоритм. Также туда войдут все операторы, от которых операторы из множества S_c зависят синтаксически.

Функциональная модель процесса масштабирования алгоритма показана на рис. 1. На первом шаге строятся зависимости, ограничивающие редукцию операторов в алгоритме, после чего определяются наблюдаемые операторы. Построение остаточного алгоритма выполняется до тех пор, пока результат, полученный на очередной итерации, не совпадет с предыдущим результатом. Операторы, не вошедшие в остаточный алгоритм, не соответствуют выбранному уровню абстракции, поэтому они либо удаляются, либо заменяются на эквивалентные по потреблению вычислительных ресурсов операторы, если в цели тестирования входит оценка времени выполнения алгоритма.

Формально уровень абстракции [6] α определяется как отображение множества операторов алгоритма $G.N$ во множество всех подмножеств переменных алгоритма 2^{Var}, объединенных со специальным элементом φ:

$$\alpha : G.N \to 2^{Var} \cup \{\varphi\}.$$

Если $\alpha(x) = \varphi$, то на данном уровне абстракции для оператора x никакие переменные не контролируются. Уровень абстракции описывает, какую часть алгоритма требуется протестировать, однако в реальности должны быть протестированы операторы, составляющие заданный уровень абстракции, а также операторы, от выполнения которых зависят существенные операторы и значения существенных переменных в точках наблюдения.

Основной проблемой составления тестов является экспоненциальный взрыв, возникающий при ветвлениях алгоритма. Поэтому понятие уровня абстракции необходимо дополнить свойством глубины анализа (которое легко

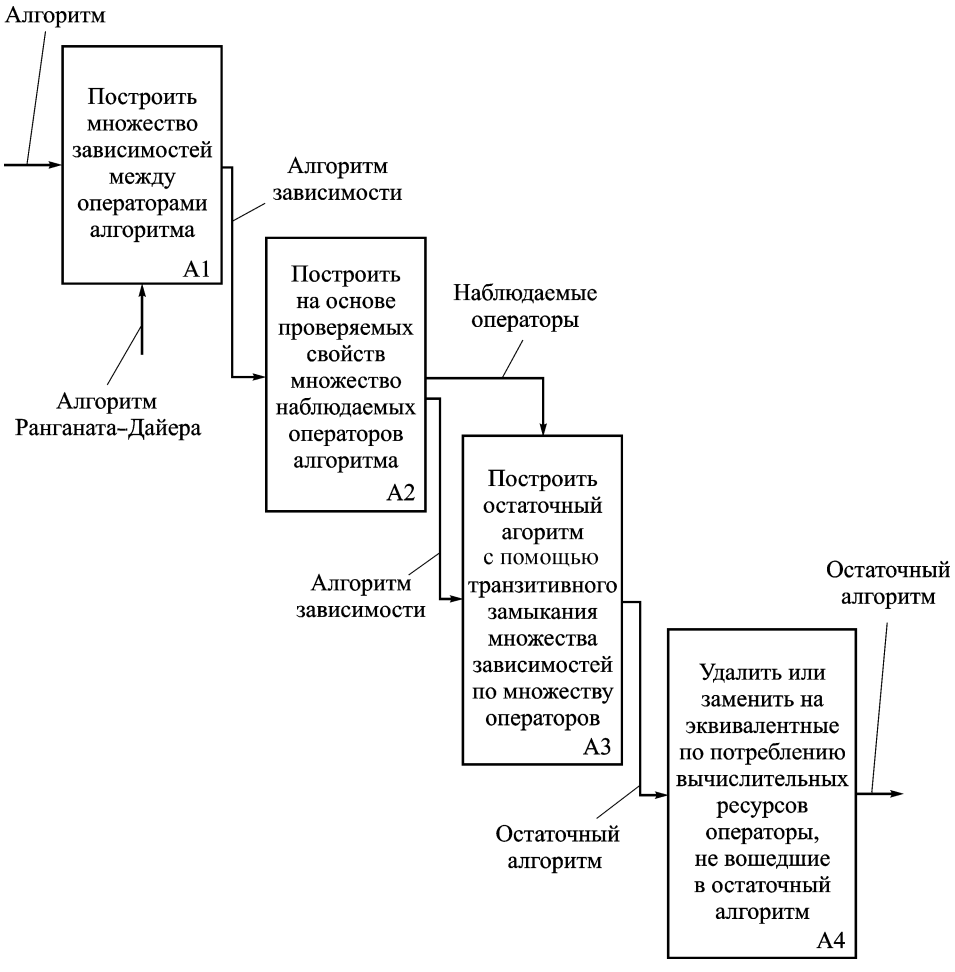

Рис. 1. Функциональная модель процесса масштабирования алгоритма

расширяется на циклы и операторы вызова). Данное свойство задает максимальную длину пути в поддереве ациклического графа выполнения алгоритма, корень которого лежит в начале ветвления. При превышении максимальной длины выполнение функциональных требований и отсутствие ошибок признаются истинными.

Предлагаемая схема масштабирования алгоритма обеспечивает 100 %-ное покрытие операторов, входящих в достаточный уровень абстракции, однако покрытие путей ограничено глубиной абстракции.

Структура программного комплекса, реализующего автоматическую генерацию тестов на основе масштабирования алгоритма, показана на рис. 2. На вход подается описание алгоритма в виде исходного текста, затем в модуле поиска достаточной абстракции определяется оптимальное с точки зрения соотношение временны́х затрат и числа обнаруживаемых ошибок уровня абстракции. На следующем этапе строится достаточный уровень абстракции с помощью определения синтаксической зависимости, зависимости по управлению и по данным. Для построенного остаточного алгоритма с помощью структурной генерации [7, 8] строятся наборы тестов, затем определяется уровень покрытия и другие статистические характеристики тестов.

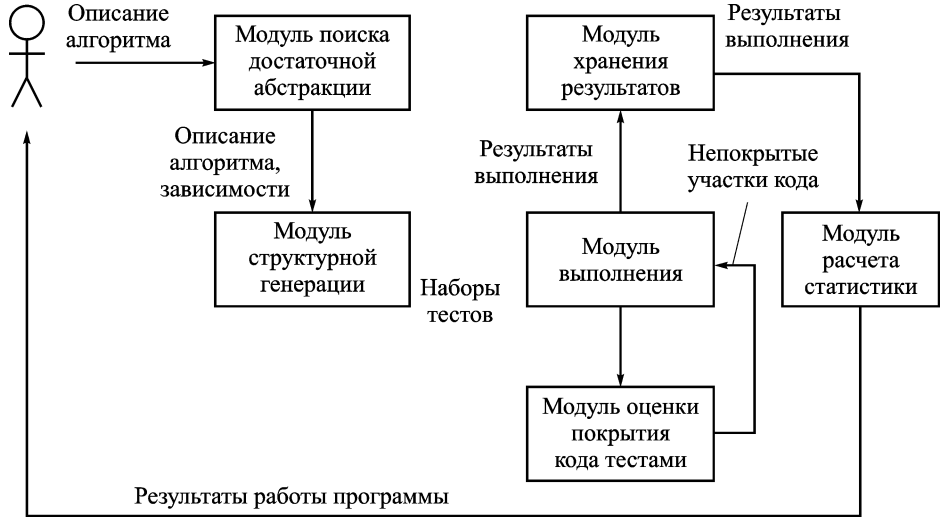

Рис. 2. Структура программного комплекса

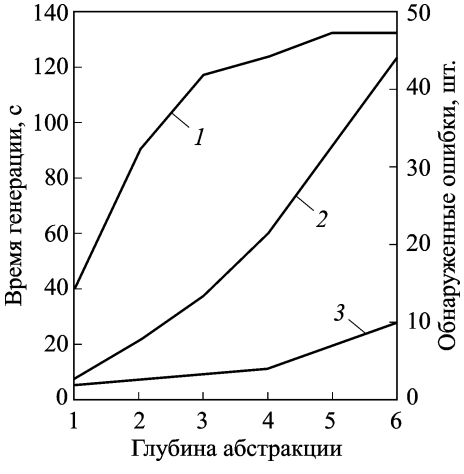

Рис. 3. Время генерации символьных решателей

На рис. 3 показано время генерации для различных символьных решателей, используемых в разработанном программном комплексе, в зависимости от глубины абстракции, а также число обнаруженных ошибок. График числа обнаруженных ошибок позволяет определить примерный уровень насыщения при глубине абстракции 3, после которого не происходит существенного повышения качества тестирования.

Структурная генерация с масштабированием показала существенное улучшение покрытия кода при одновременном уменьшении общего числа тестов (рис. 4). При этом она показывает лучшие временны́е характеристики, чем полная верификация.

Проведенные исследования позволили установить глубину абстракции, после которой не происходит существенного повышения числа найденных ошибок. Полученный результат используется в методе как первая оценка при поиске достаточного уровня абстракции, что позволяет: существенно сократить время структурной генерации; установить число ложных срабатываний (из 47 найденных ошибок 44 были подтверждены составленными вручную тестами); показать, что разработанный метод позволяет генерировать тесты, выявляющие больше ошибок и покрывающие больше кода, чем традиционные методы ручного и автоматизированного составления тестов.

Кроме того, структурная генерация тестов применима к нецелочисленным алгоритмам, которые не могут быть проверены с помощью формальной верификации.

Эксперименты проводились на модулях с открытым исходным кодом репозитория CPAN [9]. Программный комплекс, реализующий данную схему

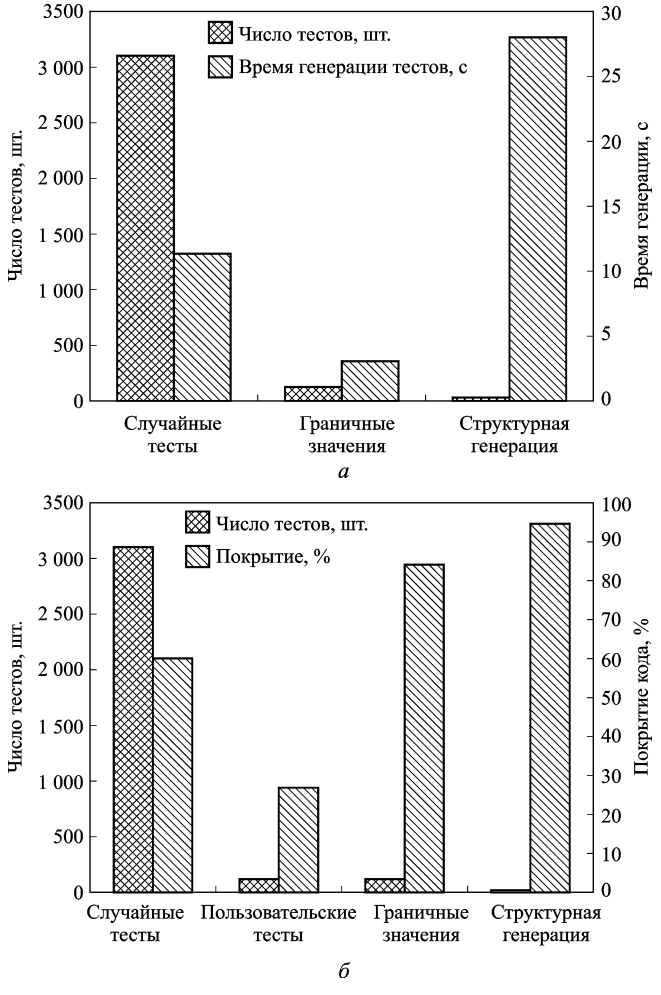

Рис. 4. Сранение способов генерации тестов по времени генерации (*а*) и степени покрытия кода (*б*)

для алгоритмов, записанных на языке Perl, зарегистрирован в Объединенном фонде алгоритмов и программ под номером ЕСПД.02076881.00425-01.

СПИСОК ЛИТЕРАТУРЫ

1. М а й е р с Г. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – 174 с.
2. B e n - A r i M. Principles of Spin // Springer Verlag. – 2008. – P. 216. of Software), volume 3639 of Lecture Notes in Computer Science, San Francisco, August 2005. Springer-Verlag.
3. B u r c h J., C l a r k e E., M c M i l l a n K., D i l l D., and H w a n g L. Symbolic model checking: 10∧20 states and beyond // Information and Computation. – 1992. – Vol. 98, No. 2. – P. 142–170.
4. К у л я м и н В. В. Методы верификации программного обеспечения // Всерос. конкурсный отбор обзорно-аналит. статей по приоритетному направлению “Информационно-телекоммуникационные системы”, 2008. – 117 с.

5. P o d g u r s k i A. and C l a r k e L. A. A formal model of program dependences and its implications for software testing, debugging, and maintenance // IEEE Trans. Softw. – Eng. 16. 9 (Sep. 1990). – P. 965–979.
6. С а в е н к о в К. О. Масштабирование дискретно-событийных имитационных моделей // Дис. . . . канд. физ.-мат. наук. – М.: МГУ им. М.В. Ломоносова, 2007.
7. Р у д а к о в И. В., Р е б р и к о в А. В. Неполная верификация систем, представленных в виде вероятностных автоматов с нечеткой функцией переходов // Информатика и системы управления в XXI веке: Сб. трудов молодых ученых, аспирантов и студентов МГТУ им. Н.Э. Баумана. – 2010.
8. Р е б р и к о в А. В., Р у д а к о в И. В. Неполная верификация систем, представленных в виде вероятностных автоматов с нечеткой функцией переходов // Материалы тринадцатого науч.-практич. сем. “Новые информационные технологии в автоматизированных системах”. – М.: Московский государственный институт электроники и математики, 2010. – С. 291–293.
9. C o m p r e h e n s i v e Perl Archive Network [Электрон. ресурс]. Режим доступа: http://cpan.org/http://cpan.org/, свободный.

Статья поступила в редакцию 23.03.2011

Игорь Владимирович Рудаков — канд. техн. наук, доцент кафедры “Программное обеспечение ЭВМ и информационные технологии” МГТУ им. Н.Э. Баумана. Автор ряда научных работ в области имитационного моделирования.

I.V. Rudakov — Ph. D. (Eng.), assoc. professor of “Computer Software and Information Technologies” department of the Bauman Moscow State Technical University. Author of a number of publications in the field of imitating simulation.

Александр Викторович Ребриков окончил МГТУ им. Н.Э. Баумана в 2010 г. Магистр техники и технологии МГТУ им. Н.Э. Баумана. Автор пяти научных работ в области верификации программного обеспечения.

A.V. Rebrikov graduated from the Bauman Moscow State Technical University in 2010. Holder of Master’s degree in engineering and technology of the Bauman Moscow State Technical University. Author of five publications in the field of software verification.