Материал опубликован в издании “Тезисы докладов Международной научной конференции, посвященной памяти профессора А.М. Богомолова. ”Компьютерные науки и технологии”. Саратов: СГУ. 2007, с. 66-69.

Верификация автоматных программ

Г.А. Корнеев, В. Г. Парфенов, А. А. Шалыто
Санкт-Петербургский государственный университет информационных технологий, механики и оптики, Санкт-Петербург
Тел.: 8-921-973-48-99, e-mail: kgeorgiy@rain.ifmo.ru

При создании программного обеспечения для ответственных систем одной из основных проблем является обеспечение его качества. Основным подходом к решению этой проблемы является *верификация программ* [1]. Выделяются два основных подхода к верификации: *динамический* и *статический* (формальная верификация) [2].

При динамическом подходе программа проверяется в процессе исполнения. Наиболее частым применением этого подхода является тестирование программ. Однако, как показал Э. Дейкстра [3], тестирование не может гарантировать корректность программы.

Формальная верификация позволяет доказать, что программа соответствует спецификации [4]. При этом под спецификацией программы будем понимать высокоуровневые требования к ее поведению. В рамках данного подхода выделяются два основных направления: *доказательная верификация* [5] и *верификация модели программы* (model checking) [6].

В общем случае, доказательная верификация является алгоритмически неразрешимой задачей, даже в таком частном случае, как доказательство останова программы [7]. Таким образом, этот подход связан с огромной ручной работой, что делает его малоприменимым на практике.

Для верификации модели программы требуется решить следующие задачи: построение по программе модели с конечным числом состояний [6] и формальное описание требований к программе терминах одного из видов темпоральной логики [8, 9]. В результате верификации модели либо подтверждается, что модель удовлетворяет формализованным требованиям, либо строится контрпример. В последнем случае требуется определить причину некорректности: ошибка в исходной программе или ошибка при построении модели. При этом часто требуется решить задачу переноса контрпримера на исходную программу.

Отметим, что решения перечисленных задач для программ общего вида плохо поддаются автоматизации.

В [10] было предложено применять конечные автоматы для описания поведения программ. Этот подход был назван «Автоматное программирование», а соответствующие ему программы — автоматными [11]. Программы этого класса обладают следующей спецификой:

- конечные автоматы являются хорошо определенной абстракцией;
- конечный автомат — одна из наиболее простых моделей дискретной математики;
- простота описания взаимодействия автоматов;
- централизация логики работы в автоматах и логическая простота входных и выходных воздействий.

Рассмотрим влияние этих факторов на верификацию автоматных программ.

При динамической верификации программ часто выдвигается требование полного покрытия тестами операторов или условий [2]. В терминах автоматной программы покрытие операторов означает, что каждое входное и выходное воздействие было проверено в процессе тестирования, а покрытие условий — что проверен каждый переход. Это позволяет автоматизировать создание тестов для автоматных программ. Например, набор регрессионных тестов [12], покрывающий все переходы, может быть сгенерирован достаточно просто (например, на основе методов, предложенных в [13]).

Доказательная верификация автоматных программ может быть разбита на две подзадачи, каждая из которых достаточно проста: верификация автоматов и верификация входных/выходных воздействий. Для верификации автоматов применяются методы, разработанные в теории компиляторов [14], а так как входные и выходные воздействия обычно не содержат сложных условий и циклов, то во многих случаях они могут быть верифицированы автоматически.

При верификации модели, обычно, по программе строится модель Крипке [6], которая, фактически, является специальным видом автомата, что упрощает ее построение по автоматной программе. Поэтому возможно создание формальных методов построения модели Крипке по автоматной программе и доказательство корректности этих методов. Таким образом, для программ этого класса можно исключить ошибки при построении модели по программе.

Построение формальных требований к модели по спецификации автоматных программ также обычно не представляет сложности. Например, такие требования, как «После состояния «открыто» всегда следует состояние «закрыто»» или «Выходное воздействие z1 не должно появляться после входного воздействия x1» могут быть записаны в терминах темпоральной логики для состояний и переходов автоматов, а не модели программы, как это делается, например, в [15, 16]. Спецификация, записанная в терминах автоматов, может быть автоматически преобразована в формальные требования к модели.

В виду того, что модель гарантированно соответствует верифицируемой программе, обработка контрпримеров также упрощается. При этом путь, приводящий к ошибке в модели Крипке, может быть легко

отображен в терминах автоматной программы [17]. При этом наглядно демонстрируется нарушение спецификации.

Таким образом, верификация автоматных программ является существенно более простой задачей, чем верификация программ общего вида. При этом многие этапы верификации могут быть автоматизированы, что позволит более широко применять ее в реальных проектах.

Наибольшие надежды связаны с верификацией моделей. Первые работы в этом направлении были проведены на кафедре технологий программирования СПбГУ ИТМО и кафедре теоретической информатики в ЯрГУ им. П. Г. Демидова. При этом для верификации ряда программ, приведенных на сайте is.ifmo.ru, применялся верификатор SPIN [18]. Полученные результаты позволяют говорить о перспективности этого подхода.

Для повышения уровня автоматизации в СПбГУ ИТМО планируется создание верификатора автоматных программ, который будет получать на вход набор автоматов, по которому строится модель, а также спецификацию программы. В качестве ядра будет использован один из существующих верификаторов, такой как, например, SPIN или Bogor [19]. При получении контрпримера будет обеспечен его перевод в термины исходной программы.

Отметим, что применение средства автоматической генерации кода позволит не проводить его верификацию, так как он будет полностью соответствовать уже верифицированному набору автоматов. Перспективность такого подхода подтверждается опытом NASA в области построения надежного программного обеспечения [20]. При этом автоматическая генерация кода по автоматам выполнялась с помощью инструментального средства Stateflow [21]. Применение разработанного верификатора совместно с программным комплексом автоматного программирования UniMod [22] позволяет надеяться на получение программ аналогичного качества.

Литература

1. Кузьмин Е. В., Соколов В. А. О дисциплине специализации «Верификация программ» // Доклады II научно-методической конференции «Преподавание математики в компьютерных науках» Ярославль: ЯрГУ. 2007. http://is.ifmo.ru/verification/_ver_prog.pdf
2. Abran A., Moore J. Swebok: Guide to the Software Engineering Body of Knowledge. http://www.swebok.org/
3. Дейкстра Э. Дисциплина программирования / Дал У., Дейкстра Э., Хоор К. Структурное программирование. М.: Мир. 1975.
4. Formal Verification http://www.nist.gov/dads/HTML/formalverf.html
5. Грис Д. Наука программирования. М.: Мир. 1984.
6. Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ. Model Checking. М.: МЦНМО. 2002.
7. Turing A. On computable numbers, with an application to the Entscheidungsproblem // Proceedings of the London Mathematical Society, V. 2, 42.
8. Pnueli A. The Temporal Logic of Programs // Proceedings of the 18th IEEE Symposium on Foundation of Computer Science. 1977.
9. Emerson E. A. Temporal and modal logic // Handbook of Theoretical Computer Science. Chapter 16. 1990.
10. Шалыто А. А. Программная реализация управляющих автоматов // Судостроительная промышленность. Серия «Автоматика и телемеханика». 1991. Вып. 13. http://is.ifmo.ru/works/switch_prr/
11. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998. http://is.ifmo.ru/books/switch/1
12. Kaner C., Falk J., Nguyen Q. Testing Computer Software. NY: Wiley. 1999.
13. Бурдонов И. Б., Косачев А. С., Кулямин В. В. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программирование. 2003, № 5. http://www.ispras.ru/~igor/doctor/papers_2003/3/1.html
14. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений. М.: Вильямс. 2002.
15. Васильева К. А., Кузьмин Е. В. Верификация автоматных программ с использованием LTL // Моделирование и анализ информационных систем. Ярославль: ЯрГУ. Т. 14. № 1. 2007. http://is.ifmo.ru/verification/_LTL_for_Spin.pdf
16. Alur R., Yannakakis M. Model Checking of Hierarchical State Machines // Proceedings of the 6th ACM Symposium on Foundations of Software Engineering. 1998
17. Вельдер С.Э., Шалыто А. А. Введение в верификацию автоматных программ на основе метода Model checking. http://is.ifmo.ru/verification/modelchecking/
18. Holzmann G. Spin Model Checker: Primer and Reference Manual. NJ: Addison-Wesley Professional. 2003.
19. Bogor. Software Model Checking Framework. http://bogor.projects.cis.ksu.edu/
20. Риган П., Хэмилтон С. NASA: Миссия надежна // Открытые системы. 2004, № 3. http://www.osp.ru/os/2004/03/184060/
21. Stateflow — Design and simulate state machines and control logic http://www.mathworks.com/products/stateflow/
22. Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А. UML. SWITCH-Технология. Eclipse //Информационно-управляющие системы. 2005. №6(13). http://is.ifmo.ru/works/uml-switch-eclipse/