004.054

ТЕМПОРАЛЬНАЯ ЛОГИКА И MODEL CHEKING В ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Шаталова Ольга Викторовна
магистрант 1 курса факультета «Инфокоммуникационных технологий»
Санкт-Петербургский национальный исследовательский университет *информационных технологий, механики и оптики, Россия, г. Самара*
E-mail: baal.tinnit@gmail.com

Статья посвящена проблемам анализа качества аппаратного и программного обеспечения становится сегодня всё более острой, особенно по мере расширения использования нанотехнологий в приборостроении и информационных технологий при разработке программного обеспечения. Экспоненциальный рост сложности аппаратного и программного обеспечения вычислительных процессов порождает повышенные требования к бездефектному проектированию. Существует множество примеров того, как дорого обходятся ошибки, допущенные на различных этапах разработки программного обеспечения. Не менее актуальными являются проблемы, связанные с обеспечением проектирования надёжных программ.
Выбор машинно-ориентированных методов верификации аппаратно-программных компонентов вычислительных процессов является актуальной.
В соответствии с МЭК 60880 верификацией признается процесс, в котором проверяется правильность интегрирования прошедших верификацию модулей аппаратных средств и программного обеспечения (ПО) в систему, их совместимость и функционирование в виде системы согласно требованиям к интеграции.

Особую важность имеет обеспечение качества ПО, так как в современных системах с его помощью реализуется основная функциональность. Современная практика программирования показывает, что системы проверяются в значительной степени людьми (экспертный анализ) и динамическим тестированием. Поддержка этих процессов даже относительно простыми инструментами, не говоря уже о методах и инструментах с серьезной математической базой, в настоящее время недостаточна. Ввиду возрастания размеров и сложности систем, становится важным обеспечение процесса валидации систем с использованием методов и инструментов, которые облегчают автоматический анализ корректности, так как, например, ручная верификация может быть настолько же ошибочна, как и сама программа.

Важнейшими методами валидации являются экспертный анализ, тестирование, симуляция, формальная верификация и проверка моделей. Тестирование – эффективный путь проверки того, что заданная реализация системы согласуется с абстрактной спецификацией. По своей природе

тестирование может быть использовано только после реализации прототипа системы.
Гарантированное обоснование качества систем может быть получено только при помощи альтернативного подхода, принципиально отличного от тестирования. Данный подход называется верификацией.
Формальная верификация, как противоположность тестированию, основана на математическом доказательстве корректности программ. Оба этих метода могут поддерживаться и частично поддерживаются инструментами. Например, в области тестирования возрастает интерес к разработке алгоритмов и программ для автоматической генерации и выбора тестов по формальной спецификации системы.
Основой формальной верификации являются программы для автоматического доказательства теорем и проверки доказательств, однако даже их использование обычно требует высокой квалификации пользователей, что может сильно повышать стоимость разработки ПО.

Формальная верификации базируется на темпоральных логиках и позволяет уменьшить участие разработчика в верификационном процессе. Эта техника называется Model checking (проверка на моделях) [1].
Model checking – это автоматизированный подход, позволяющий для заданной модели поведения системы с конечным (возможно, очень большим) числом состояний и логического свойства (требования) проверить, выполняется ли это свойство в рассматриваемых состояниях данной модели.
Основная идея Model checking состоит в моделировании – описании разработчиком поведенческой модели системы, подлежащей верификации, и спецификации – формулировке требований (желаемого поведения системы. Модель программы не всегда полно отражает ее поведение. Разработчик при построении модели, как правило, абстрагируется от несущественных ее свойств. Такая концепция дает возможность уменьшить размер самой модели и ускорить процесс ее проверки. Если модель удовлетворяет указанным требованиям, то программаверификатор сообщает об этом. Если же обнаруживается ошибка, то она предоставляет контрпример, который показывает, при каких условиях могло возникнуть данное несоответствие. Контрпример представляет собой сценарий, в котором модель ведет себя нежелательным образом. Это означает, как правило, что модель ошибочна и подлежит пересмотру. [2]
Model checking позволяет разработчику обнаружить ошибку и исправить модель или требования. Если не найдено ни одной ошибки, разработчик может усовершенствовать описание модели.

Основная трудность моделирования – не потерять важные детали программы, а трудность задания требований – сформулировать их корректно и исчерпывающе. Алгоритмы для Model checking обычно базируются на полном просмотре пространства состояний модели: для каждого состояния проверяется, удовлетворяет ли оно сформулированным требованиям.

Алгоритмы гарантированно завершаются, так как модель конечна. Принципиальная схема Model checking приведена на рис. 1.

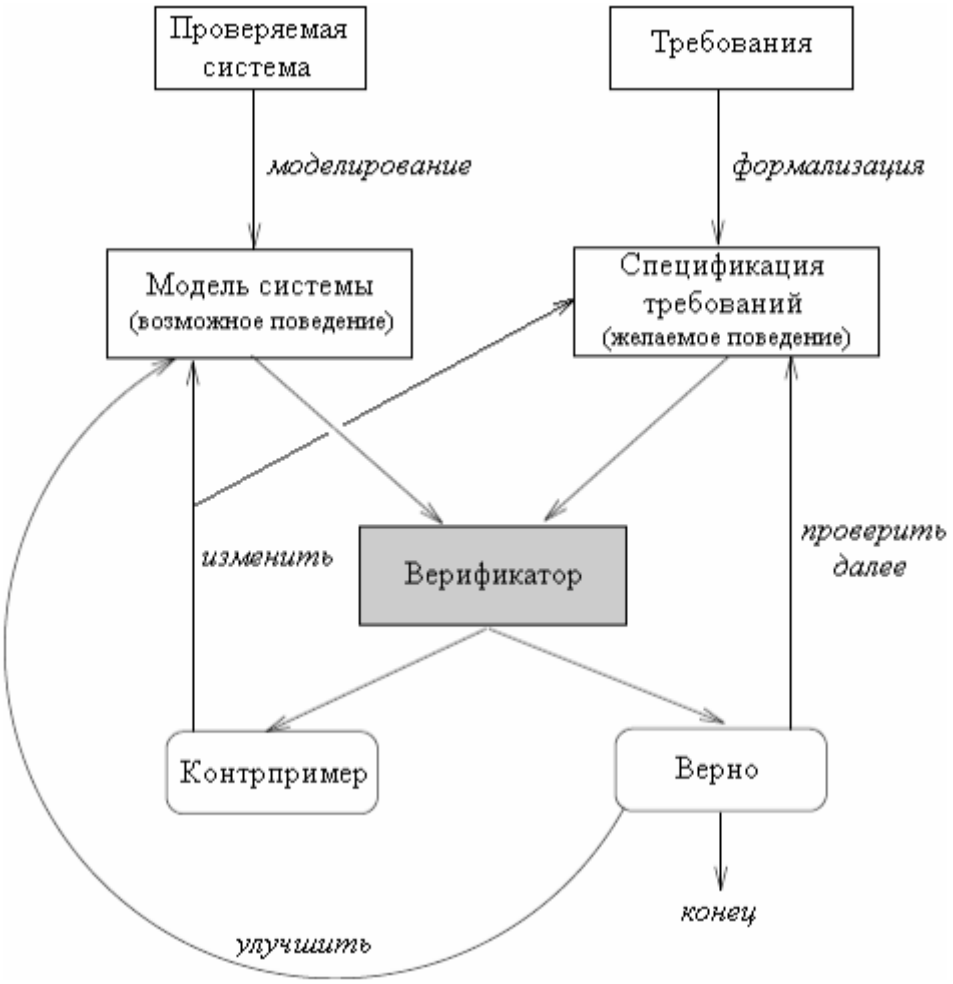

Рис. 1. Model checking

Достоинства метода Model checking:
1. Эффективность. Программы для верификации моделей способны работать с достаточно большими пространствами состояний благодаря концепции упорядоченных двоичных разрешающих деревьев. [3]
2. Контрпримеры. Ограничения Model checking:

- Поддержка только конечных моделей. Для большинства классов систем с бесконечным числом состояний необходимо выполнять формальную верификацию системы – математическое доказательство свойств самой программы, а не ее модели.
- Ограниченность верификации. С использованием Model checking проверяется модель системы вместо реальной системы. Таким образом, любое применение метода Model checking настолько же качественно, как и сама модель системы.
- Для многопроцессорных систем размер пространства состояний в худшем случае пропорционален произведению размеров пространств

состояний их индивидуальных компонент. Так как Model checking выполняется на моделях, близких по структуре к конечному автомату, то для сложных многопроцессорных систем эта концепция перестает быть эффективной.

Составление сценариев (в том числе, контрпримеров) верифицирующими инструментами позволяет проводить исследования в области автоматической или интерактивной коррекции модели или автомата с целью удовлетворить предъявляемым условиям. Например, если программа-верификатор предъявила путь, опровергающий некоторое желательное свойство для системы, она может предложить разработчику исказить/ликвидировать этот путь, скажем, за счет удаления какого-либо перехода. При этом, разумеется, не гарантируется, что в модели при этом не возникнет других противоречий со спецификацией, хотя, не исключается возможность и более интеллектуальной коррекции.[6]

Исходя из изложенного выше, можно кратко сформулировать основные достоинства автоматных программ в части их верификации:
• класс автоматных программ является наиболее удобным для верификации методом Model checking, так как в этом случае модель программы может быть автоматически построена по спецификации ее поведения, задаваемой в общем случае системой взаимодействующих конечных автоматов, в то время как для программ других классов модель приходится строить вручную;
• структура автоматных программ, в которых функции входных и выходных воздействий почти полностью отделены от логики программ, делает практичным верификацию этих функций на основе формальных доказательств с использованием пред- и постусловий [7].

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Katoen J.–P. Concepts, Algorithms, and Tools for Model Checking. Lehrstuhl für Informatik VII, Friedrich-Alexander Universität Erlangen-Nürnberg. Lecture Notes of the Course (Mechanised Validation of Parallel Systems) (course number 10359). Semester 1998/1999.
2. Clarke E. M., Grumberg O., Peled D. A. Model Checking. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=3730
3. Clarke E. M, Jr. Lectures on Model Checking. Computer Science Department, Carnegie Mellon University, 2005.
4. Соколов В. А., Чалый Д. Ю. Методы исследования поведения транспортных протоколов в условиях интенсивного сетевого трафика. Ярославль: Ярославский государственный университет, 2004. http://www.teletraffic.ru/public/pdf/Sokolov_Chalyi_2004.pdf
5. Покозий Е. А. Методы спецификации и верификации параллельных моделей с непрерывным временем. Автореферат диссертации на

соискание ученой степени кандидата физико-математических наук. Институт систем информатики Сибирского отделения РАН. Новосибирск, 1999.
6. Мейер Б. Объектно-ориентированное конструирование программных систем. М.: Русская редакция. 2005.
7. Миронов А. М. Математическая теория программных систем. http://intsys.msu.ru/study/mironov/mthprogsys.pdf