# Содержание

# От издательства

## *Отзывы и пожелания*

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

## *Список опечаток*

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

## *Нарушение авторских прав*

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

# Предисловие ко второму изданию

Второе издание по большей части совпадает с первым, но есть некоторые отличия. Во-первых, были устранены обнаруженные опечатки и неточности, переработан текст, добавлены иллюстрации и примеры. Во-вторых, была написана новая лекция (глава 15), посвященная SMV-подобным инструментам символьной проверки моделей (теперь теоретический материал лекции 14 подкреплен практикой). Кроме того, в книгу был добавлен именной указатель, а также сноски с краткой информацией о математиках, логиках и программистах, внесших вклад в становление и развитие формальных методов.

## *Структура книги*

Книга состоит из 16 лекций (глав), снабженных вопросами для самопроверки и упражнениями: лекция 1 – вводная; лекции 2–8 посвящены дедуктивному анализу программ; лекции 9–15 – проверке моделей; лекция 16 – использованию формальных методов в тестировании. Более подробная информация о содержании лекций приведена ниже.

В лекции 1 дается представление о корректности программ, т. е. их соответствии требованиям, и верификации – процессе проверки корректности. Делается обзор базовых методов верификации, как формальных (проверка эквивалентности, проверка моделей, дедуктивный анализ), так и неформальных (экспертиза, тестирование). Рассматривается общая схема формальной верификации: программе ставится в соответствие формальная модель программы, требованиям – формальная модель требований; проверка корректности программы сводится к формальной процедуре, устанавливающей соответствие между первой и второй моделями. Приводятся некоторые типы моделей и отношений соответствия.

В лекции 2 на примере простого императивного языка while рассматриваются два метода формализации семантики языков программирования: операционная семантика (структурная операционная семантика Плоткина) и аксиоматическая семантика (дедуктивная система Хоара и метод Дейкстры, основанный на понятии слабейшего предусловия). Показывается, что программа – код на языке с формализованной семантикой – является математическим объектом, к которому применимы строгие логические рассуждения, в частности о его корректности или некорректности (конечно, при наличии формальных спецификаций требований).

В лекции 3 рассматриваются основы дедуктивной верификации последовательных программ: приводятся сведения о дедуктивных системах и теории

доказательств; дается представление о верификации программы как поиске доказательства условия корректности в дедуктивной системе, формализующей язык программирования (платформу). Показывается, что инварианты циклов не выводимы из структуры программы. Задание инвариантов дает набросок доказательства корректности, проверку которого можно автоматизировать.

В лекции 4 рассматриваются элементы языка ACSL (ANSI/ISO C Specification Language), предназначенного для формальной спецификации (аннотирования) программ на языке C. Делается обзор платформы статического анализа Frama-C (Framework for Modular Analysis of C Programs), а также модулей (plugins) Jessie и AstraVer, автоматизирующих дедуктивную верификацию C-программ на соответствие ACSL-спецификациям. Изложение базируется на примерах (подробное описание языка ACSL и основанных на нем инструментов можно найти в документации). Описанные средства могут использоваться в повседневной программистской практике.

В лекции 5 рассматривается один из первых подходов к формальной верификации программ – метод Флойда. Подход состоит из двух частей: первая (метод индуктивных утверждений) предназначена для доказательства частичной корректности, т. е. корректности без учета завершимости; вторая (метод фундированных множеств) используется для доказательства завершимости и полной корректности (этой части посвящена лекция 6). Метод Флойда имеет дело с блок-схемами и позволяет анализировать программы с общим видом потока управления, в том числе не отвечающие принципам структурного программирования. Подход состоит в рассечении циклов, сопоставлении точкам сечения утверждений, построении условий верификации, связывающих утверждения соседних точек (индуктивных переходов), и их доказательстве.

В лекции 6 рассматривается метод фундированных множеств, применяемый в паре с методом индуктивных утверждений для доказательства завершимости и полной корректности программ. Идея подхода заключается в следующем: каждой точке сечения сопоставляется оценочная функция; значение функции уменьшается при каждом прохождении через эту точку (при условии что истинно индуктивное утверждение), но в то же время не может уменьшаться бесконечно долго в силу характера изменений и ограничений на область значений. Если удается найти такие функции и доказать условия верификации, программа завершается и, более того, является полностью корректной.

В лекции 7 рассматриваются вопросы автоматизации дедуктивной верификации программ, в частности вопросы автоматического синтеза инвариантов циклов. Задача синтеза оценочных функций может быть сведена к построению инвариантов программ, расширенных счетчиками. Описываются два класса методов: эвристические методы и методы, основанные на абстрактной интерпретации, т. е. символьном исполнении в упрощенных предметных областях – абстрактных моделях. Определяются основные понятия абстрактной интерпретации; приводятся примеры для таких областей, как интервальная арифметика и теория выпуклых многогранников.

В лекции 8 рассматриваются основные подходы к автоматическому доказательству теорем, включая метод резолюций для логики высказываний и логики первого порядка. Вводятся понятия разрешающей процедуры и разрешимой теории, приводятся примеры разрешимых и неразрешимых теорий. Большое внимание уделяется базовым принципам работы SAT- и SMT-решателей – средств, составляющих ядро инструментов формальной верификации программ: описывается алгоритм DPLL проверки выполнимости КНФ; приводится разрешающая процедура для теории равенства неинтерпретируемых функций; дается представление о методе Нельсона–Оппена комбинирования разрешающих процедур для разных теорий.

В лекции 9 рассматриваются параллельные программы – программы, состоящие из нескольких процессов, исполняемых совместно и взаимодействующих друг с другом через общие переменные. Семантика таких программ определяется с помощью парадигмы чередования (на каждом шаге исполняется оператор одного из процессов, выбранного недетерминированным образом). Особое внимание уделяется так называемым реагирующим системам, которые работают в «бесконечном цикле» и отвечают на события окружения путем исполнения тех или иных действий. Описывается один из формализмов, применяемых для спецификации реагирующих систем, – темпоральная логика линейного времени (LTL, Linear-time Temporal Logic). Для иллюстрации методов спецификации параллельных программ используется алгоритм Петерсона взаимного исключения двух процессов.

В лекции 10 рассматриваются базовые возможности языка Promela и основанного на нем инструмента Spin. Язык Promela (Process Meta-Language) предназначен для моделирования асинхронных параллельных систем. Инструмент Spin (Simple Promela Interpreter) позволяет анализировать Promela-модели, в том числе устанавливать их соответствие LTL-спецификациям, т. е. проверять, является ли заданная модель (программа на языке Promela) «моделью» (в логическом понимании) заданной формулы LTL (истинна ли формула на всех вычислениях модели). Описанные средства могут быть использованы на практике при проектировании протоколов, параллельных программ и реагирующих систем.

В лекции 11 рассматриваются вопросы моделирования программ структурами Крипке, в том числе связанные с выбором уровня абстракции данных и гранулярности действий. Обсуждаются возможные аномалии: появление в модели поведения, не присущего исходной программе, и связанные с этим ложные сообщения об ошибках (false positives), а также исчезновение возможного поведения и связанные с этим пропуски ошибок (false negatives). Дается представление о предикатной абстракции программ и адаптивном уточнении абстракции на основе контрпримеров – методе CEGAR (Counter-Example Guided Abstraction Refinement). Рассматриваются базовые подходы к исследованию пространства состояний параллельной программы, позволяющие по моделям процессов строить модель программы.

В лекции 12 рассматриваются ω-языки, т. е. языки с бесконечными словами, и один из способов их описания – автоматы Бюхи. Структурно ав-

томаты Бюхи идентичны конечным автоматам, но отличаются от них условием распознавания слов. Показывается, как для языков, описываемых конечными автоматами и автоматами Бюхи, решаются задачи нахождения пересечения и проверки на пустоту. Если множество всех возможных траекторий системы и множество всех ошибочных траекторий системы могут быть представлены в форме автоматов Бюхи, то решение указанных задач дает ключ к проверке модели: если пересечение указанных множеств не пусто, модель ошибочна. Показывается, что множество траекторий структуры Крипке тривиальным образом описывается автоматом Бюхи. Вопрос о возможности представления автоматами Бюхи формул LTL рассматривается в следующей лекции.

В лекции 13 рассматривается ключевой этап теоретико-автоматного подхода к проверке моделей для логики LTL – синтез автомата Бюхи, распознающего все траектории, на которых истинна заданная формула LTL (и только их). Результатом синтеза является недетерминированный обобщенный автомат Бюхи, число состояний которого экспоненциально зависит от длины формулы. Описанная процедура может быть использована и для проверки выполнимости формулы: если язык, допускаемый построенным автоматом, не пуст, то формула выполнима (существует траектория, на которой она истинна).

В лекции 14 рассматривается символьная проверка моделей для логики LTL. Отличие от классического подхода, рассмотренного ранее, состоит в том, что модель (множество состояний и отношение переходов), а также автомат Бюхи, построенный по формуле LTL, представляются неявно, в символьной форме. Общая идея метода остается прежней, но алгоритмы поиска в графах состояний заменяются на манипуляции с их символьными представлениями, что частично решает проблему комбинаторного взрыва. Ключевую роль здесь играют способы символьного представления множеств и отношений. Рассматривается один из таких способов, позволяющий создавать эффективные символьные алгоритмы, – двоичные решающие диаграммы (BDD, Binary Decision Diagrams).

В лекции 15 (добавленной в этом издании) рассматриваются основные возможности инструментов семейства SMV (Symbolic Model Verifier) и лежащего в их основе языка моделирования: представление параллельных систем в виде иерархически организованных модулей, спецификация требований на языке темпоральной логики и собственно символьная проверка моделей, базирующаяся на двоичных решающих диаграммах (BDD) и техниках проверки выполнимости (SAT). Описанные средства могут быть использованы на практике для разработки и анализа компьютерных протоколов, моделей программ и цифровой аппаратуры.

В лекции 16 рассматриваются вопросы применения формальных методов к тестированию программ. Подходы, в которых формальные модели и техники используются для решения задач тестирования, называются методами тестирования на основе моделей (model-based testing). Для них характерны автоматическая проверка правильности поведения программ в ответ

на тестовые воздействия, наличие формализованных критериев полноты тестирования, нацеленность генерации тестов на тестовое покрытие. Описываются некоторые распространенные подходы: тестирование на основе программных контрактов, символьное исполнение и конколическое (конкретно-символьное) тестирование, генерация тестов с помощью обхода графа состояний.

## *Благодарности*

Автор хотел бы поблагодарить **А. М. Коцыняка** – коллегу по ИСП РАН, на протяжении многих лет проводящего практические занятия по курсу (без которых невозможно прочувствовать трудность верификации и красоту формальных методов). Все прежние благодарности, конечно, остаются в силе.

## *Рекомендуемая литература*

Для более глубокого погружения в тему формальной верификации программ рекомендуем книги [Apt09], [Bai08], [Lei23], [Вел11], [Кар10] и [Мир23].

## *Замечания*

Автор будет рад любой информации об ошибках и неточностях, встретившихся на страницах книги; приветствуются любые предложения по ее улучшению. Контактная информация не изменилась (см. предисловие к первому изданию).

# Предисловие к первому изданию

*Корректность* программы – соответствие ее *функциональности* (того, *что* программа делает) *требованиям* (тому, что она *должна* делать) – может достигаться либо *безошибочным проектированием* (*correctness by design*), либо путем постепенного выявления и устранения несоответствий, ошибок. Первый подход кажется предпочтительным. В самом деле, зачем тратить усилия на поиск и исправление ошибок, если можно сразу все сделать правильно? Однако не все так просто. Во-первых, не сразу понятно, что значит «правильно»: требования к программе уточняются по мере ее разработки: то, что кажется правильным сегодня, оказывается неправильным завтра. Во-вторых, программа эволюционирует: требования к каждой конкретной версии могут быть заданными, но меняются от версии к версии; «синтез» корректной программы с нуля по обновленным спецификациям требует огромных затрат; если же идти путем доработки имеющейся версии (что обычно и делают), может нарушиться корректность.

*Верификацией* программ, или, шире, компьютерных систем, называется деятельность, направленная на выяснение их правильности или, наоборот, ошибочности. Деятельность эта может принимать разные формы: *инспекция кода* (просмотр и рецензирование «исходников»); *статический анализ* (поиск типовых ошибок: чтений неинициализированных переменных, разыменований нулевых указателей и т. п.); *тестирование* (запуск программы на примерах и проверка корректности результатов); *имитационное моделирование* (создание исполнимой модели системы и ее исследование в специальном окружении); *формальная верификация* (построение логической модели системы и ее анализ средствами математической логики). Подходов много, однако задача верификации настолько трудна, что ни один из них не годится на роль «серебряной пули», способной гарантировать корректность действительно сложных проектов; лучшие результаты, как показывает практика, достигаются при совместном использовании разных методов.

Предлагаемое вашему вниманию пособие посвящено одному классу методов – *формальным методам*. Их суть состоит в создании математических моделей программ и требований и в логическом анализе соответствия между построенными моделями. В некотором смысле формальные методы – это фундамент, на котором стоит здание программной инженерии. (Аналогичным фундаментом в свое время выступил математический анализ по отношению к физике.) Следует отметить, что это не «голая теория», как часто думают: формальные методы давно вышли за пределы академического сообщества и стали частью индустрии; сегодня это неотъемлемая часть процесса

разработки систем ответственного назначения (микропроцессоров, операционных систем, бортовых комплексов). Пособие преследует две основные цели: способствовать более глубокому пониманию природы программирования и сформировать практические навыки верификации компьютерных систем.

Книга основана на курсах лекций, читаемых автором на факультете ВМК МГУ им. М. В. Ломоносова, ФУПМ МФТИ и ФКН ВШЭ. В ней изложены основы широко известных подходов к верификации: *дедуктивного анализа* и *проверки моделей* (*model checking*). Рассматриваются такие темы, как методы формализации семантики языков программирования (операционная и аксиоматическая семантика), методы формальной спецификации требований (программные контракты и темпоральная логика линейного времени), методы доказательства корректности программ (метод индуктивных утверждений и метод фундированных множеств) и методы проверки моделей (теоретико-автоматный подход в явной и символьной формах). Кроме того, в пособии затрагиваются вопросы абстрактной интерпретации, автоматизированного доказательства теорем и разрешения ограничений, применения формальных методов в тестировании; даются сведения об инструментах Frama-C и Spin.

Пособие написано для студентов и аспирантов программистских специальностей, а также преподавателей и исследователей в области информатики и вычислительной техники. От читателя требуется знание основ дискретной математики и математической логики.

## *Благодарности*

Автор благодарен академику **В. П. Иванникову** (1940–2016) – основателю, первому директору и научному руководителю ИСП РАН, заведующему кафедрами системного программирования МГУ, МФТИ и ВШЭ, – инициировавшему создание учебного курса по верификации и предложившему автору его читать, и профессору **А. К. Петренко** – заведующему отделом Технологий программирования ИСП РАН, – всячески поддерживающему автора в написании этого пособия.

Отдельных слов благодарности заслуживают студенты – слушатели курса.

## *Замечания*

Как и программы, книги содержат ошибки, и эта – не исключение. Автор будет признателен за любую информацию об ошибках и неточностях, встретившихся на страницах пособия, а также за замечания и предложения по его улучшению.

**Контактная информация**
Камкин Александр Сергеевич, к. ф.-м. н., в. н. с. ИСП РАН
109004, г. Москва, ул. Александра Солженицына, д. 25 (ИСП РАН)
E-mail: kamkin@ispras.ru

# Лекция 1

# Введение в формальные методы верификации программ

*Большая формальность подходит для проектов с большой критичностью.*

А. Коберн.
Быстрая разработка
программного обеспечения

*Переход от неформального к формальному существенно неформален.*

М. Р. Шура-Бура

Дается представление о корректности программ, т. е. их соответствии требованиям, и верификации – процессе проверки корректности. Делается обзор базовых методов верификации, как формальных (проверка эквивалентности, проверка моделей, дедуктивный анализ), так и неформальных (экспертиза, тестирование). Рассматривается общая схема формальной верификации: программе ставится в соответствие формальная модель программы, требованиям – формальная модель требований; проверка корректности программы сводится к формальной процедуре, устанавливающей соответствие между первой и второй моделями. Приводятся некоторые типы моделей и отношений соответствия.

# 1.1. Ошибки в программах

Программистский фольклор гласит, что в каждой (даже тщательно отлаженной) программе есть хотя бы одна ошибка[1]. Эта шутка появилась не на пустом месте: при высокой сложности компьютерных систем (а это миллиарды транзисторов и миллионы строк кода) избежать ошибок – несоответствий системы требованиям – практически невозможно. Речь идет не только о прикладных решениях, создаваемых в спешке не вполне компетентными специалистами, но и об ответственных (safety-critical) компонентах, включая аппаратуру (микропроцессоры, устройства ввода-вывода, хранения и передачи данных) и системное ПО (операционные системы, компиляторы, драйверы устройств). Системы, при проектировании или реализации которых допущены ошибки, могут в той или иной ситуации повести себя непредсказуемо, приводя к фатальным последствиям. Примеров этому много (см., например, [Адж98] и [Кар10]). Вот некоторые из них, на наш взгляд, достаточно показательные.

- В 1982 году канадской фирмой Atomic Energy of Canada Limited был запущен в серию медицинский аппарат «Therac-25», предназначенный для лучевой терапии – облучения раковой опухоли. В «редких» ситуациях из-за ряда ошибок, допущенных при проектировании и реализации встроенной системы управления, интенсивность облучения возрастала на 2 порядка и более. За время эксплуатации прибора (период с июня 1985 года по январь 1987 года) как минимум два пациента умерли, а несколько человек остались инвалидами[2].
- 2 сентября 1988 года была потеряна связь с советской межпланетной станцией «Фобос-1», запущенной 7 июля того же года для исследования Марса и его спутника Фобоса. За несколько дней до этого, 29 августа, с Земли была передана некорректная команда, ошибочно воспринятая как отключение системы ориентации и стабилизации. Станция перестала направлять солнечные батареи на Солнце, из-за чего ее аккумуляторы разрядились. Через полгода была потеряна связь с «Фобос-2», дублером «Фобос-1». Основная задача обеих миссий – доставка на по-

---

[1] Это высказывание известно как первая аксиома М. Р. Шуры-Буры. Полностью эта шутливая аксиоматика выглядит так:

**Аксиома 1**. В каждой программе есть хотя бы одна ошибка.
**Аксиома 2**. Если ошибки нет в программе, она есть в реализуемом алгоритме.
**Аксиома 3**. Если ошибки нет и в алгоритме, такая программа никому не нужна.

Михаил Романович Шура-Бура (1918–2008) – советский и российский ученый, внесший существенный вклад в становление и развитие программирования в СССР; лауреат Сталинской премии (1955) и Государственной премии СССР (1978).

[2] Случай с медицинским аппаратом «Therac-25» не является единственным в своем роде: в 1991 году в Испании при использовании аппарата «Sagitar-35» подверглись передозировке не менее 25 пациентов, из которых как минимум трое умерли.

верхность Фобоса долгоживущей автономной станции – осталась невыполненной[1].

- 13 июня 1994 года Томас Найсли (Thomas Nicely), профессор математики из Линчбургского колледжа, обнаружил ошибку в реализации инструкции деления чисел с плавающей точкой в микропроцессоре Pentium фирмы Intel (как выяснилось, в таблице начальных приближений присутствовало несколько неправильных значений). И хотя рядовых пользователей ПК эта ошибка, казалось бы, не должна была беспокоить, компании для сохранения имиджа пришлось организовать бесплатную замену выпущенных микросхем. Затраты на это составили 475 миллионов долларов – более половины прибыли Intel за последний квартал 1994 года.
- 4 июня 1996 года на 39-й секунде после старта взорвалась ракета-носитель «Ариан 5», разработанная Европейским космическим агентством (ЕКА). Бортовая система частично использовала ПО предыдущей версии ракеты, «Ариан 4», в том числе модуль управления инерциальной навигационной системой[2]. При преобразовании 64-битного числа с плавающей точкой (угла наклона ракеты) в 16-битное целое возникло переполнение, не обработанное модулем. При разработке предполагалось, что переполнение невозможно из-за физических ограничений ракеты, но прогресс не стоит на месте – в «Ариан 5» использовались новые двигатели.
- 23 марта 2003 года во время войны в Ираке американский зенитно-ракетный комплекс «Patriot» ошибочно идентифицировал британский истребитель-бомбардировщик «Tornado», летевший близ кувейтской границы, как приближающуюся вражескую ракету и автоматически произвел залп. Погибли два пилота. Спустя полторы недели, 2 апреля, «дружественным огнем» (friendly fire) был уничтожен американский палубный самолет F/A-18 «Hornet». Еще один пилот погиб.

Список можно продолжить... Следует понимать, что ошибки в системах не являются чем-то исключительным: как говорили римляне, errare humanum est – человеку свойственно ошибаться. Согласно статистике, среднее число ошибок на тысячу строк неотлаженного кода колеблется в пределах 10–50[3] [Mcc04]. К сожалению, в некоторых областях наблюдается тенденция к деградации качества проектирования: «С точки зрения способности все за-

---

[1] Неудачная участь постигла и межпланетную станцию «Фобос-Грунт», запущенную 9 ноября 2011 года: из-за нештатной ситуации станция не смогла покинуть окрестности Земли и 15 января 2012 года сгорела в плотных слоях атмосферы (некоторые фрагменты станции, вероятно, затонули в Тихом океане).

[2] Кстати говоря, ПО в ЕКА разрабатывается на языке Ada, созданном по заданию Министерства обороны США как язык программирования встроенных систем с повышенными требованиями к надежности и безопасности.

[3] Известно, что ошибки распределены в программах неравномерно и имеют тенденцию образовывать скопления [Gla02]. Согласно [Boe01], примерно 80 % ошибок находятся в 20 % модулей (это одно из проявлений принципа Парето).