Ю. Г. Карпов

MODEL CHECKING

Верификация параллельных и распределенных программных систем

«Автор будет считать свою задачу выполненной, если его книга хотя бы в малой степени изменит существующее положение в отечественной науке и образовании в области верификации программ. Следует признаться, однако, что забота об отечественной науке составила лишь долю мотивации написания этой книги. Намного более сильным было желание автора поделиться с читателем тем восхищением, которое вызывают изящество формальных моделей и их удивительная мощь при решении важнейших насущных практических проблем информатики».

Юрий Карпов

+cd

Ю. Г. Карпов

MODEL CHECKING

Верификация параллельных и распределенных программных систем

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
К26

Карпов Ю. Г.

К26 MODEL CHECKING. Верификация параллельных и распределенных программных систем. — СПб.: БХВ-Петербург, 2010. — 560 с.: ил. + CD-ROM

ISBN 978-5-9775-0404-1

В книге рассказывается о новых результатах в области верификации с помощью метода *model checking* и приводятся примеры приложений этого метода в самых разных областях. Рассматриваются проблема верификации, темпоральные логики, алгоритмы *model checking* для CTL и LTL, структуры Крипке как модели реагирующих систем, спецификация свойств реагирующих систем формулами темпоральной логики, бинарные решающие диаграммы, символьная верификация, количественный анализ систем и системы реального времени, а также применение фундаментальных идей алгоритма *model checking* в различных приложениях.

На прилагаемом компакт-диске находятся: инструкция по установке системы верификации Spin, методическое пособие по этой системе и описание курсовой работы по верификации нетривиальной системы логического управления с несколькими вариантами заданий.

Для студентов вузов, программистов
и исследователей в области информатики

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Татьяна Лапина*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Сержантова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 01.10.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,15.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0404-1

Оглавление

Анализ корректности программных систем — тема, которая не теряет своей актуальности. Поэтому неудивительно, что тысячи исследователей и инженеров во всем мире пытаются изобрести новые методы и разработать новые инструменты проверки программ. Специалисты по теоретической информатике даже создали отдельную ветвь этой дисциплины, которая называется "формальные методы разработки и анализа программ". Однако, несмотря на массированные атаки на проблему, до последнего времени не удавалось добиться реальных результатов, которые можно было внедрять в широкую практику.

Model checking оказался одним из первых формальных методов, который вышел из лабораторий и университетов в мир промышленной разработки программных и аппаратных систем. Это одновременно радует и удивляет специалистов в области верификации программ — ведь для того, чтобы тонкий математический инструмент стал простым и понятным для обычных программистов, нужно было решить не только многие научные и технические проблемы, но и преодолеть привычный консерватизм, традиционную практику не рисковать и не полагаться в серьезных проектах на слишком новые и еще неопробованные технологии.

Такое "преодоление" невозможно без энтузиазма людей увлеченных и авторитетных. Лишь они могут сломать предубеждение по отношению к новым технологиям. Юрий Глебович Карпов — один из них. Он сочетает в себе высокую компетенцию и юношескую увлеченность изяществом теории, о которой он пишет. В результате читатель получил, пожалуй, первую книгу на русском языке, которая раскрывает потенциал нового поколения инструментов разработки программ, построенных на основе использования формальных методов.

В. П. Иванников

Академик РАН, директор Института системного программирования РАН, заведующий кафедрами системного программирования на факультете вычислительной математики и кибернетики МГУ и в МФТИ, главный редактор журнала "Программирование"

Это необычная книга. Особенность ей придают три обстоятельства:

- Актуальность и прикладная значимость ее тематики: сейчас уже смело можно сказать, что метод model checking проверки правильности функционирования компьютерных систем и устройств и их программного обеспечения вышел за пределы лабораторий и активно стал использоваться в промышленности.
- Личность автора: известного специалиста и одного из основных подвижников в исследованиях применения темпоральных логик для верификации программ и систем у нас в стране. Дар преподавателя помог ему найти и построить свою методику изложения, отличную от таких известных публикаций, например, как E. M. Clarke, O. Grumberg, D. Peled. Model checking // MIT Press, 1999. Русский перевод: Э. М. Кларк, О. Грамберг, Д. Пелед. Верификация моделей программ: Model Checking // М., 2002.
- На сегодня эта книга, пожалуй, наиболее полное изложение метода model checking и многочисленных его приложений.

Р. Л. Смелянский

Заведующий лабораторией вычислительных комплексов факультета ВМиК МГУ, доктор физико-математических наук, профессор, академик РАЕН

Предисловие

21 июня 2008 года на торжественном банкете Ассоциации по вычислительной технике (ACM) в Сан-Франциско, Калифорния, трем ученым была вручена высшая награда Ассоциации — премия Тьюринга 2007 г. — "*за их роль в превращении метода Model checking (по-русски — "проверка модели") в высокоэффективную технологию верификации, широко используемую в индустрии разработки программного обеспечения и аппаратных средств*".

Model checking — это формальная проверка того, выполняется ли заданная логическая формула на данной структуре. Структура представляет собой модель разрабатываемой системы, логическая формула описывает требования к поведению системы. Этот метод уже сегодня широко используется во всем мире для проверки сложных объектов — как программного обеспечения, так и аппаратуры. Он позволяет существенно повысить степень уверенности разработчиков в правильности функционирования интегральных схем, протоколов коммуникации, драйверов устройств, бортовых систем управления для автомобилей, самолетов, космических аппаратов.

Премия Тьюринга рассматривается мировым сообществом как Нобелевская премия в области информатики. Она вручается ежегодно с 1966 г. за выдающийся вклад в усовершенствование компьютерных технологий. Премия Тьюринга за 2007 г., пожалуй, впервые отметила ученых, вклад которых состоял не столько в разработке удивительно красивой теории, имеющей важное практическое применение, сколько в доведении этой теории до уровня "высокоэффективной промышленной технологии". Президент ACM Стюарт Фельдман сказал: "*Это великий пример того, как технология, изменившая промышленность, родилась из чисто теоретических исследований*".

Новые научные результаты были сразу востребованы практикой. Без концептуального прорыва в теории верификации, без использования этих новых результатов разрабатываемые сегодня программные и аппаратные продукты

беспрецедентной сложности не могли бы быть созданы с достаточным уровнем надежности, а компании-разработчики в перспективе разорились бы, выплачивая неустойки за ущерб, нанесенный ошибочным функционированием их разработок. Однако едва ли не большее значение метод model checking приобрел в последние годы в связи с возможностью применения его результатов и идей во многих областях, весьма далеких от области верификации технических систем, для которой этот метод был первоначально разработан.

Данная книга посвящена изложению этих новых идей и описанию их применений. В книге рассказывается о новых результатах в области верификации, полученных с помощью метода model checking, и приводятся примеры использования этого метода в самых разных областях. Эта область теоретической информатики была разработана сравнительно недавно, но она уже привела к "прорывным" практическим результатам и вызвала значительный интерес из-за своей изящности и эффективности.

Верификация программных систем — фундаментальный раздел информатики, основанный на формальных методах. Трудность изучения и восприятия формальных методов часто является следствием оторванности таких формализмов от практики. Среди программистов широко распространено мнение, что формальные методы не имеют никакого реального применения к решению практических проблем, что это просто необязательное приложение к рутинным приемам техники программирования, которые одни только и нужны программисту. Подход *model checking* опровергает это мнение. Несмотря на то, что полученные здесь результаты базируются на таких абстракциях, как темпоральные логики и структуры Крипке, бесконечные языки и автоматы Бюхи, неподвижные точки операторов, а также на других "страшных" формализмах, материализация этих абстрактных понятий дала удивительные практические результаты. На их основе созданы и работают автоматические системы верификации, позволяющие разработчикам систем методом "*нажатия кнопки*" (push button) проверять бортовые системы управления космических аппаратов, встроенные программы военной техники, мобильных телефонов, медицинской аппаратуры. Путь от этих абстракций к практическим задачам информатики удивительно прям и очевиден. Некоторые ученые называют это направление "*реабилитацией формальных методов*": здесь формальные модели позволили получить результаты, без применения которых невозможно выполнить любую практическую разработку программных и аппаратных систем для серьезных, критических применений.

Материал книги будет полезен трем категориям читателей.

Во-первых, это студенты, обучающиеся по направлению "Информатика". Изучение теории и методов, объединенных под общим термином model checking, включено в учебные планы многих иностранных университетов.

Такой интерес вызван тем, что этот раздел представляет наиболее значительные результаты в области верификации с тех пор, как эта проблема впервые была поставлена 40 лет назад. Интерес вызван также простотой и логической ясностью этих результатов, а также возможностью их практического применения. Этот новый раздел теоретической информатики станет, несомненно, обязательной частью базового фундаментального образования в области информатики. "Формальные методы должны стать частью образования каждого исследователя в области информатики и каждого разработчика ПО так же, как другие ветви прикладной математики являются необходимой частью образования в других инженерных областях", — говорится в отчете NASA. Хотя техника применения этого подхода проще, чем теория, верификация систем требует достаточно высокой квалификации исполнителей. Специалисты в этой области весьма востребованы.

Во-вторых, это научные работники. В отличие от многих теоретических разделов вычислительной науки, которые стали уже классикой, данное направление является "горячей" областью исследований, здесь продолжается активная исследовательская работа, которая направлена на расширение возможностей верификации технических систем и на применение этого подхода к широкому кругу других задач. Богатство применений этой новой техники в совершенно разных областях, огромное число открытых проблем не могут оставить равнодушным ни одного исследователя. В книге приводятся примеры того, как эти новые идеи используются для решения многих проблем в различных областях.

Третьей категорией читателей, которые могут извлечь пользу из данной книги, являются инженеры-разработчики новых систем. "Формальные методы обеспечивают интеллектуальную основу нашей области, они помогают оформить наши мысли и направить их на плодотворный путь при решении проблем; они обеспечивают нотацию при документировании требований и разработки, позволяют выразить мысли при общении разработчиков в точной и ясной манере, они обеспечивают нас инструментальными средствами для анализа свойств систем", — говорится в упомянутом выше отчете NASA. Раздел формальной верификации, основанный на методе *model checking*, достиг уровня зрелости. Он предоставляет разработчику программных и аппаратных систем теоретические основы, ясную методику, простую нотацию и эффективный инструментарий для верификации систем. Материал данной книги достаточен для того, чтобы любой программист, заботящийся о качестве результатов своего труда, смог немедленно применить изложенные здесь методы для проверки своей разработки и повышения степени доверия к ней, используя доступный инструментарий и технику "push button".

Для понимания книги необходимы только начальные знания в области дискретной математики: логики высказываний и теории автоматов в объеме

вводных университетских курсов, а также первоначальный опыт программирования.

В одной вводной книге невозможно изложить все, или даже большую часть вопросов, относящихся к данной области. Автор постарался по возможности просто и компактно раскрыть основные идеи и принципы этого направления. Интересующиеся читатели без труда смогут найти литературу, относящуюся к другим разделам этой области, и после прочтения данной книги разобраться в ней.

К настоящему времени формальная верификация уже стала этапом технологического цикла при создании сложных встроенных систем во многих фирмах-разработчиках программно-аппаратных систем для критических применений. Но среди названий этих фирм трудно найти российские. Курсы по формальным моделям, темпоральным логикам и верификации стали традиционными во всех ведущих университетах мира — но их читают только в ничтожно малом числе вузов России. Существует обширная литература, посвященная исследованиям в этой области, но на русском языке такой литературы почти нет. Исследования проводятся во всех развитых странах — кроме России, где этими проблемами занимаются только в двух-трех научных группах. Результаты этого очевидны. Например, компания "Сухой" с гордостью рапортует о заключенных контрактах на поставку ее истребителей за рубеж, но на экспортируемые самолеты ставится электроника, сделанная не в России, а во Франции. Именно во Франции работает один из трех лауреатов премии Тьюринга 2007 г., в нескольких лабораториях Франции ведутся интенсивные исследования в области верификации и разработаны технологии создания надежных бортовых систем. Очевидно, что зарубежная авионика может иметь лишь очень ограниченное применение в отечественной боевой авиации и в других критических областях.

Автор будет считать свою задачу выполненной, если его книга хотя бы в малой степени изменит существующее положение в отечественной науке и образовании в этой области.

Следует признаться, однако, что забота об отечественной науке составила лишь долю мотивации для написания этой книги. Намного более сильным было желание автора поделиться с читателем тем восхищением, которое вызывают изящество формальных моделей и их удивительная мощь при решении важнейших насущных практических проблем информатики.

Структура книги

Представленный в книге материал может составить основу одно- или двухсеместрового университетского курса по верификации параллельных и распределенных программ.

Структура представленного материала следующая.

Глава 1 является вводной. В ней описываются некоторые примеры ошибок в программных системах и их последствия. Показывается, что именно параллельные и распределенные программы являются сложными для их понимания и анализа, что формальная верификация является важнейшим средством выявления тонких ошибок в параллельных системах. Приводится общая схема верификации и рассматривается метод model checking как один из подходов к формальной верификации. Здесь же анализируются сильные и слабые стороны верификации и тестирования, этих двух классических методов повышения уровня доверия к разработкам. Приводятся примеры успешного применения верификации к практическим разработкам программного обеспечения и аппаратуры: коммуникационным протоколам, бортовым системам управления космическими аппаратами и т. п.

В *главе 2* вводятся темпоральные логики LTL, CTL и CTL* и рассматривается их использование для спецификации свойств поведения реагирующих систем.

Главы 3 и *4* посвящены изложению алгоритмов проверки модели для случаев, когда проверяемые свойства системы выражены формулами логик CTL и LTL.

В *главе 5* изучаются системы переходов как модели реагирующих систем.

Глава 6 посвящена рассмотрению свойств реагирующих систем и их классификации: безопасности, живости и справедливости.

В *главе 7* описываются некоторые примеры верификации с помощью представленной ранее методики проверки модели.

Глава 8 приводит многочисленные примеры применения алгоритма проверки моделей в самых разных областях — решение логических задач, задач планирования, поиск атак в криптографическом протоколе.

Одним из препятствий при использовании метода проверки модели является экспоненциальный рост числа состояний параллельных систем при увеличении числа компонентов и параметров модели. Эта проблема, называемая "*взрывом числа состояний*", рассматривается в *главах 9* и *10*. В *главе 9* рассказывается о новом методе представления булевых функций — бинарных решающих диаграммах (Binary Decision Diagrams, BDD), который позволяет эффективно представлять любые конечные дискретные структуры данных и эффективно оперировать ими, а также приводятся примеры применений этого метода. Данную главу можно изучать независимо от других глав, но этот материал является основным для рассмотрения в следующей, *10 главе*, символьных алгоритмов верификации, позволяющих анализировать системы с астрономическим числом состояний — 10^{100} и более.

В *главе 11* изучаются подходы к анализу количественных свойств дискретных систем — вероятностных систем и систем дискретного времени. Здесь рассматривается, как проверить свойства, выражаемые утверждениями типа: "с вероятностью, не меньшей 0.99, на переданный пакет будет получено подтверждение не более чем через 6 единиц времени".

Наконец, *глава 12* посвящена проблемам верификации систем реального времени.

Практическая часть курса состоит из двух компонентов. Во-первых, в каждой главе книги существует раздел с задачами и упражнениями. Во-вторых, студенты могут получить опыт формального моделирования и анализа параллельных и распределенных систем на входном языке Promela системы верификации Spin в рамках курсовой работы. Материалы этой курсовой работы содержатся на диске.

Сопровождающий компакт-диск

На диске, приложенном к книге, находятся:

- инструкция по установке системы верификации Spin на компьютере студента;
- методическое пособие по этой системе верификации, содержащее описание языка Promela, примеры представления на нем параллельных и распределенных программ и примеры использования системы Spin для верификации программ и протоколов;
- описание курсовой работы по верификации нетривиальной системы логического управления с несколькими вариантами заданий для группы студентов.

По запросу преподавателей им могут быть высланы ответы и решения к задачам и упражнениям, приведенным в книге, а также слайды курса лекций, которые читает автор по этому предмету.

Благодарности

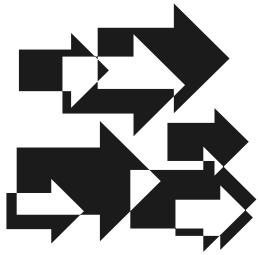
Материал книги основан, прежде всего, на фундаментальной монографии Э. Кларка, О. Грамберга и Д. Пеледа [34], которая во многих университетах мира является базовым учебником для курсов по верификации и model checking. В книге использованы также материалы из руководств, оригинальных статей и многочисленных учебных материалов, доступных в Интернете. Недавняя монография Кристела Байера и Питера Катоена по принципам model checking [19] также была использована при подготовке книги.

Автор выражает огромную благодарность всем людям, которые предоставили в Интернете в открытом доступе свои статьи, курсы лекций, слайды презентаций на эту тему. Всех их трудно перечислить, но особую благодарность хотелось бы выразить Марте Квятковской (Marta Kwiatkowska) из Бирмингемского университета, Борису Коневу из Ливерпульского университета, Юрию Лифшицу из Калифорнийского технологического университета, Питеру Катоену (Joost-Pieter Katoen) из университета Аахена, Киму Ларсену (Kim Guldstrand Larsen), Полю Петерсону (Paul Pettersson), Пьеру Валперу (Pierre Wolper) и многим другим.

Автор читает курс с этим названием в Санкт-Петербургском политехническом университете уже 10 лет. Изложение множества вопросов, казавшихся ясными, многократно перерабатывалось после того, как на экзаменах выяснялось, что эти вопросы вызывали затруднения у студентов. За эту обратную связь автор приносит благодарность своим студентам.

Издание книги поддержано Я. Ю. Карповым, за что автор приносит ему особую благодарность.

ГЛАВА 1



Проблема верификации

В этой главе обосновывается необходимость верификации программ. Приводятся примеры тяжелых последствий программных ошибок в системах для критических применений, от которых все больше зависят жизнь и здоровье людей, использующих технику. Дается сравнительный анализ верификации и тестирования, представлена общая схема верификации и кратко охарактеризован тот конкретный подход к верификации, которому посвящена книга, — метод *проверки модели* (model checking).

1.1. Ошибки в программах и их последствия

Компьютерные программы после их разработки очень часто содержат ошибки. С этим знаком каждый, разработавший программу длиной хотя бы в несколько десятков строк. В современных программных системах избежать ошибок разработки невозможно. Сложность промышленных программных систем все время возрастает. Например, ОС Microsoft Windows 3.1, разработанная в 1992 г., содержит около 3 млн строк кода, Microsoft Windows 98 — 18 млн строк, RedHat Linux 6.2 (2000 г.) — около 20 млн строк, RedHat Linux 7.1 (2001) — уже 30 млн строк, Microsoft Windows XP (2002) — 40 млн строк. Мысленно охватить функционирование таких систем не в состоянии ни один человек даже при использовании современных технологий и методов абстрагирования и управления сложностью.

Сложность разрабатываемого программного обеспечения подошла к границе его понимания и, следовательно, его управляемости. Число потенциальных ошибок в программных системах постоянно растет. Например, в ОС Windows 95, по оценкам Microsoft, до сих пор остается несколько тысяч оши-

бок. Но эта ОС используется в бытовых персональных компьютерах, от нее не зависят жизни людей. Только сумасшедший поставит эту ОС в бортовую систему управления самолетом или даже автомашиной.

Особенно подвержены ошибкам параллельные, распределенные и многопоточные программы, которые характерны для бортовых систем управления. Хорошо известно, что даже в тех случаях, когда функционирование каждой из параллельных взаимодействующих компонентов системы абсолютно ясно, человеку трудно понять работу всей параллельной системы, процессы в которой взаимозависимы. Как пишет Лесли Лэмпорт, "очень легко ошибиться, когда делаешь утверждения о последовательности событий, происходящих в параллельных программах" [94]. Причина этого, по-видимому, в том, что человек в каждый момент может думать только об одной вещи. Параллельные системы, которые работают правильно "почти всегда", годами могут сохранять тонкие ошибки, проявляющиеся в редких, исключительных ситуациях. Такие ошибки обычно невозможно обнаружить тестированием. "Существует обширный печальный опыт того, что параллельные программы весьма упорно сопротивляются серьезным усилиям по выявлению в них ошибок", — писали Сьюзен Овицки и Лесли Лэмпорт еще 25 лет тому назад [121]. В результате все чаще компании подвергаются штрафным санкциям и вынуждены выплачивать неустойки за последствия ошибок, проявившихся после передачи готовой системы заказчику.

Человек не любит рассказывать о своих ошибках. Разработчики программ обычно не спешат уведомлять всех о своих неудачах. Ни одна компания, производящая программы и компьютерную аппаратуру, не склонна без особой надобности объявлять об ошибках в своих продуктах и их последствиях. Это одна из причин того, что ошибки в программных и аппаратных системах происходят значительно чаще, чем мы об этом узнаем. Только в случаях, имеющих серьезные последствия ошибочного поведения программ, проводится анализ причин ошибок и возникших из-за них потерь. Те результаты такого анализа, которые становятся широко известными, показывают, что материальные последствия ошибок в программных и аппаратных системах управления могут быть весьма значительными, порой приводящими к полному провалу дорогостоящих проектов и даже к разорению компаний-разработчиков ПО и аппаратуры. Вот некоторые наиболее впечатляющие примеры последствий ошибок в программах.

- 4 июня 1996 г. при первом запуске на 39-й секунде после старта взорвалась ракета "Ариан 5". "Ариан 5" — это аналог российской ракеты-носителя "Протон", которая разработана и производится Европейским космическим агентством. Бортовая система ракеты "Ариан 5" использовала ПО предыдущей версии, "Ариан 4", но архитектура аппаратной части

была модифицирована. После проведенного тщательного расследования было установлено, что причина аварии — ошибка в навигационной программе бортового компьютера. Потери от аварии были огромными. Кроме прямых потерь — утраты ракеты (затраты на ее разработку превышали 7 млрд долларов) и установленного на борту научного оборудования стоимостью более полумиллиарда долларов, ущерб составили и упущенная выгода от несостоявшихся запланированных запусков, и потеря на долгие годы репутации ракеты как надежного средства выведения полезной нагрузки на околоземную орбиту. Непрямые потери составили около 2 млрд долларов.

- 25 декабря 2004 г. после семилетнего полета космический зонд Гюйгенс отделился от автоматической международной межпланетной станции "Кассини" и начал спуск в атмосферу Титана — спутника Сатурна. 15 января 2005 г. зонд впервые в истории начал передавать информацию с поверхности Титана. Ошибка в бортовой программе зонда привела к тому, что половина переданной информации была потеряна. Затраты на разработку зонда составили около 2,5 млрд долларов.
- 28 мая 2008 г. Сообщение Ассошиэйтед Пресс: "NASA потеряло связь с марсоходом "Феникс". Выполнение задач, запланированных на второй день нахождения на Марсе американского космического аппарата "Феникс", отложено из-за возникших неполадок. Связь с "Фениксом" утеряна". Стоимость проекта — 420 млн долларов.
- В 1994 г. при разработке процессора Intel Pentium была допущена ошибка во встроенной программе деления: несколько констант вспомогательного массива не были инициализированы. Ошибка была обнаружена уже после того, как дефектные процессоры поступили в продажу, и компанией "Интел" все они были заменены. Потери компании составили сотни миллионов долларов.
- В декабре 2007 г. в новых четырехъядерных процессорах AMD Phenom (Barcelona) и Opteron, произведенных по новой, 65-нанометровой технологии, выявлена ошибка реализации буфера быстрого преобразования адреса кэш-памяти третьего уровня. Ошибка может приводить к зависанию системы.
- 26 февраля 2009 г. Агентство Bloomberg сообщает: "Японское подразделение крупнейшего швейцарского банка UBS из-за компьютерной ошибки чуть было не потратило 3 триллиона иен (31 млрд долларов) на выкуп облигаций компании Capcom. Финансовая организация объяснила, что собиралась разместить заказ на покупку облигаций на сумму в 30 млн иен (310 000 долларов), но из-за системной ошибки к заказу добавились пять нулей".

Финансовые потери от ошибок в программах только в США оцениваются в десятки миллиардов долларов в год. Например, в 2002 г. эти потери оценены в 59,6 млрд долларов [52]. Необходимость более тщательной проверки программ и аппаратных систем приводит к задержкам в выполнении проектов, увеличению времени вывода продукта на рынок (time-to-market), а это грозит потерями из-за конкуренции и в связи со срывами сроков поставки.

Иногда ошибки в автоматических системах приводят не только к материальным потерям, но и к потерям человеческих жизней.

- В 1982 г. канадской компанией Atomic Energy of Canada Ltd. был запущен в серию медицинский аппарат Therac-25, предназначенный для облучения раковой опухоли потоком электронов. В приборе использовалось новое программное обеспечение встроенной системы управления процессом облучения. Несмотря на "тщательное тестирование", которому подверглось ПО, в нем остались ошибки: в редко встречающихся режимах, в которые прибор мог войти при некоторых входных управляющих последовательностях, интенсивность облучения возрастала на 2 порядка. Прибор эксплуатировался в разных клиниках несколько лет. За время эксплуатации прибора некоторые больные получили передозировки облучения, двое пациентов умерли, несколько человек остались инвалидами.
- 20 декабря 1995 г. в катастрофе самолета "Боинг-757" (Колумбия, рейс из Майами в Кали) погибли 159 человек. Расследование показало, что причиной катастрофы была ошибка в одном символе программной системы управления полетом. Изготовитель бортового компьютера, компания Honeywell Air Transport Systems и разработчик ПО компания Jeppesen Sanderson of Englewood были признаны виновными в гибели людей. Родственникам жертв аварии было выплачено 300 млн долларов.
- Февраль 2008 г. Сообщение СМИ: "Американский спутник-шпион вышел из-под контроля и может в конце февраля упасть на Землю. Обломки спутника, по информации источника в правительстве США, могут содержать опасные материалы, а точно установить, где именно упадет аппарат, пока не представляется возможным".
- 23 марта 2003 г. система "Patriot" ошибочно идентифицировала британский бомбардировщик "Tornado" как приближающуюся вражескую ракету и автоматически произвела залп. Погибли два пилота. 2 апреля 2003 г. системой "Patriot" уничтожен американский истребитель F-16, пилот погиб. В обоих случаях причиной явились ошибки в бортовой системе автоматического обнаружения цели и наведения.
- 11 апреля 2008 г., сообщение СМИ: "В результате операции во время патрульного рейда вертолет США, "спутав цель", нанес удар по бронемаши-

нам, принадлежащим ВВС США. Ранения получили пять человек — двое американских военнослужащих и трое мирных иракцев".

Во время войны в Ираке в США появился термин "дружественный огонь" (friendly fire). До 24 % всех потерь живой силы в 1-й Иракской войне произошло из-за этой причины. Выводы многочисленных комиссий по расследованию подобных инцидентов однозначны: главная причина "дружественного огня" — все возрастающая сложность технологий XXI века, используемых в военных системах, и ошибки в программном и аппаратном обеспечении автоматического управления в военных системах.

К этой же теме относится заметка в PC Week от 28.04.2008 г., озаглавленная так: "*Боевой робот взбунтовался против хозяев*": "В Ираке проводился эксперимент по эксплуатации усовершенствованных моделей боевых роботов "Talon", оснащённых пулемётом M249 и управляемых дистанционно. Один из ожидавших приказа аппаратов по непонятной причине "ожил" и стал самостоятельно шевелить манипулятором, в котором было установлено заряженное оружие".

- 2 сентября 1988 г. на борт советской межпланетной станции "Фобос-1", отправленной на Марс, была передана с Земли неверная команда. Бортовым компьютером эта команда была воспринята как команда на отключение системы ориентации и стабилизации. Эта ошибка бортовой системы управления, которая обязана выявлять некорректные внешние команды и должным образом реагировать на них, привела к тому, что станция "Фобос-1" потерялась в космосе. Через полгода была потеряна связь и с ее дублером — станцией "Фобос-2". Наиболее вероятная причина этой второй неудачи — также программная ошибка. В результате просторы Вселенной сейчас бороздят безмолвные, бесполезные для человечества советские станции "Фобос-1" и "Фобос-2".

Приведенные примеры показывают, что ошибки в сложных программных и аппаратных системах не являются чем-то исключительным, они повторяются регулярно в разработках сложных систем. Их непосредственными причинами являются и некорректные спецификации, и неправильное понимание спецификаций разработчиками, взаимное непонимание в коллективе разработчиков, непредвиденные события, режимы и условия работы, несогласованность параллельных ветвей программ и многое другое.

В наше время, когда все чаще человеческая жизнь зависит от функционирования автоматических систем, проблема гарантии правильности программных и аппаратных компонентов этих систем приобретает первостепенное значение. Надежность и предсказуемость поведения таких систем являются более важными свойствами, чем производительность, модифицируемость и т. п. Верификация, как один из основных методов повышения качества систем, становится важнейшей областью информатики.

1.2. Примеры ошибочных программ

Параллельные и распределенные программы становятся все более распространенными. В современном автомобиле существует целая сеть связанных микропроцессоров, в новых "мерседесах" их около 50. Производители соревнуются в выпуске многоядерных процессоров для персональных компьютеров, но реально их возросшая вычислительная мощность может быть использована только при программировании параллельно выполняемых процессов. В [136] происходящий сейчас поворот программирования в сторону параллелизма назван "революцией" с лозунгом: "Добро пожаловать в мир параллельной обработки". Но именно этот мир чреват проблемами, возникающими вследствие необходимости синхронизации параллельных процессов.

В мире параллельной обработки разработчик должен контролировать не последовательности возможных событий, как в последовательном программировании, а возможные комбинации частично упорядоченных событий, что значительно сложнее. Многочисленные примеры показывают, что нетривиальные многопоточные программы непостижимы для человеческого мозга. С ошибками, возможными в параллельных программах, может сегодня столкнуться каждый. В качестве примера трудновыявляемых ошибок в параллельных программах приведем несколько простых программ.

Пример 1.1

В период обучения в вузе студенты часто работают. Студент A поступил на работу в компанию SoftTech и разработал бухгалтерскую программу БухСофт, которую его компания использует сама и продает. Студент открыл депозитный банковский счет X_A, с начальным капиталом 0, на который бухгалтерская программа $БухСофт1$ компании SoftTech должна перевести ему 3000 долларов за эту работу.

Поскольку студент еще и учится, ему выплачивается стипендия. Как троечнику, ему полагается стипендия всего в 15 долларов. Бухгалтерская программа БухСофт2 университета в конце месяца должна увеличить счет этого студента на 15 долларов.

Таким образом, в двух независимых процессорах два параллельных процесса независимо выполняют операции над одним элементом общей памяти:

$$БухСофт1 :: X_A := X_A + 3000 \qquad БухСофт2 :: X_A := X_A + 15$$

К концу месяца студент рассчитывает, что после двух выплат у него на счете будет 3015 долларов, достаточные для покупки подержанной иномарки. К его

разочарованию, на счете оказалось только 15 долларов, хотя обе программы отработали и, по распечаткам, все деньги ему переведены.

Причиной некорректности является ошибка в разработанной студентом программе: отсутствие синхронизации двух параллельных процессов, выполняющих одновременный доступ к общим данным. Подобные ошибки типичны в программах, разработанных студентами-недоучками, пропускавшими лекции по теории параллельных процессов. Хотя операция начисления зарплаты записывается как $X_A := X_A + C$, реально в каждом процессе она выполняется как три неделимых операции:

1. Чтение в регистр значения из памяти.
2. Увеличение значения регистра.
3. Запись значения из регистра в память.

Если процесс *БухСофт*1 выполняет все три свои операции либо ДО, либо ПОСЛЕ того, как процесс *БухСофт*2 выполнит все свои операции, то значение X_A в результате увеличится на $C_1 + C_2$, т. е. программы будут работать правильно. Существует, однако, очень малая вероятность того, что эти две группы операций будут выполняться независимо работающими процессами приблизительно в одно время. В этом случае асинхронное параллельное выполнение независимых процессов может привести к тому, что неделимые микрокоманды разных процессов будут выполняться по очереди в некотором порядке. В итоге мы можем получить неожиданный результат: одно из изменений значения общей переменной, которое выполняет либо один, либо другой процесс, будет потеряно.

Таким образом, бухгалтерская программа, разработанная студентом, работает корректно "почти" все время, но иногда, очень редко, программа будет работать неправильно. Выявить подобные тонкие ошибки синхронизации параллельных процессов никаким тестированием невозможно, они зависят от относительных скоростей процессов, времени начала выполнения операций и т. п.

Пример 1.2

Программа разделения множеств [165]. Эта простая программа состоит из двух параллельно функционирующих взаимодействующих процессов, `Small` и `Large`. Процесс `Small` оперирует на конечном множестве чисел `S`, процесс `Large` оперирует на конечном множестве `L`. Процессы `Small` и `Large` взаимодействуют посредством посылки сообщений, обмениваясь числами из этих множеств по каналу `c1` от `Small` к `Large` и по каналу `c2` от `Large` к `Small` до тех пор, пока в `S` не соберутся все минимальные значения обоих множеств, а в `L` не соберутся максимальные значения множеств.

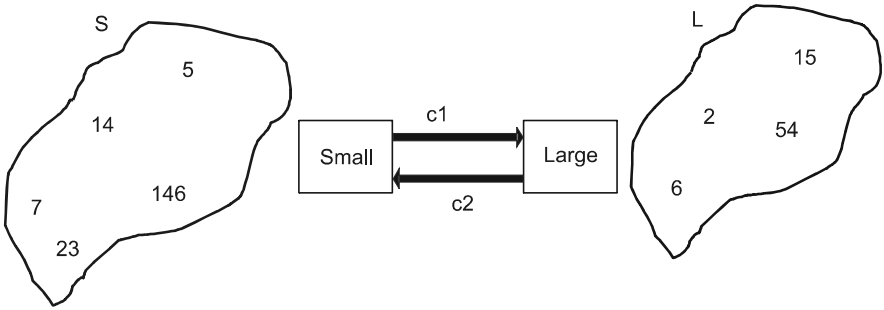


Рис. 1.1. Параллельная программа разделения множеств

Процесс `Small` на каждом шаге ищет максимальное значение в множестве `S` и посылает его процессу `Large` по каналу `c1`, затем принимает от `Large` найденное этим процессом минимальное значение `L`, посланное по каналу `c2`:

```
Small::
begin
  mx:=max(S); c1!mx; S:=S-{max(S)};
  c2?x; S:=S∪{x}; mx:=max(S);
  *[mx>x →    // цикл выполняется, пока mx>x
          c1!mx; S:=S-{max(S)};
          c2?x; S:=S∪{x}; mx:=max(S);
   ]
end
```

Процесс `Large` на каждом шаге принимает от `Small` найденное им максимальное значение `S`, посланное по каналу `c2`, сам ищет минимальное значение во множестве `L` и посылает его процессу `Small` по каналу `c1`:

```
Large::
begin
  c1?y; L:=L∪{y}; mn:=min(L);
  c2!mn; L:=L-{mn}; mn:=min(L);
  *[mn<y →    // цикл выполняется, пока mn<y
          c1?y; L:=L∪{y}; mn:=min(L);
          c2!mn; L:=L-{mn}; mn:=min(L)
   ]
end
```

Взаимодействие по каналам `c1` и `c2` очень простое — это синхронная коммуникация (хэндшейк, рандеву), при которой посылка одним процессом данных в канал может быть выполнена в том и только в том случае, если другой процесс готов прочесть эти данные.

Почти на всех вариантах исходных данных эта параллельная программа работает верно: процессы завершаются, по их завершении все данные сохраняются, число значений в каждом множестве не изменяется, каждый элемент `S` не больше каждого элемента `L`. Очень редко при некотором вполне определенном соотношении между значениями элементов множеств `S` и `L` эта параллельная программа работает некорректно: один из процессов будет бесконечно ожидать коммуникации с другим, который до этого благополучно завершился. Правильность этой программы нельзя доказать тестированием: почти на всех наборах исходных данных эта программа работает правильно. Только строгий формальный анализ, верификация программы может обнаружить ошибку.

Пример 1.3

Протокол W.C.Lynch [105]. Этот протокол был приведен Уильямом Линчем как пример убедительно выглядящей, но некорректной распределенной программы, опубликованной "одним из основных производителей вычислительной техники". Цель протокола — полнодуплексная передача (т. е. одновременная передача в обоих направлениях) последовательностей символов через ненадежную среду между пользователями `A` и `B`.

Каждое сообщение имеет две части: информационный символ и управляющую часть. Управляющая часть принятого сообщения всегда содержит один из трех управляющих символов: `ack`, `nack` или `err`. Если сообщение принято без ошибок, оно содержит `ack` или `nack` в управляющей части. Канал может искажать сообщения, но не может терять, дублировать или вставлять новые. Предполагается, что все ошибки передачи обнаруживаются нижним уровнем протокола, верхний уровень при этом получает символ `err` в управляющей части сообщения.

Правила функционирования протокола описываются его разработчиком так:

"В ответном сообщении в управляющей его части посылается положительное подтверждение `ack`, если полученное перед этим сообщение не содержало ошибок, в ответ посылается отрицательное подтверждение `nack`, если было принято ошибочное сообщение.

Если полученное сообщение несло отрицательное подтверждение или само оказалось ошибочным, то в ответ посылаются старые данные, в противном случае посылаются следующие данные".

Формализованное представление этих правил в стиле SDL дано на рис. 1.2, *а*. Схема полностью соответствует правилам: полученное новое сообщение `msg` может иметь три возможных типа (три разных значения в управляющей части). Если пришло неискаженное сообщение, которое содержит "`nack`" в управляющей части, то в ответном сообщении посылается положительное