Институт Системного Программирования
Российской Академии Наук

На правах рукописи

БУРДОНОВ Игорь Борисович

Теория конформности для функционального тестирования программных систем на основе формальных моделей

специальность 05.13.11 –
математическое и программное обеспечение вычислительных машин,
комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание учёной степени
доктора физико-математических наук

Москва - 2008

Работа выполнена в Институте Системного Программирования РАН.

Официальные оппоненты: доктор технических наук,
профессор
Евтушенко Нина Владимировна

доктор физико-математических наук,
профессор,
Крюков Виктор Алексеевич

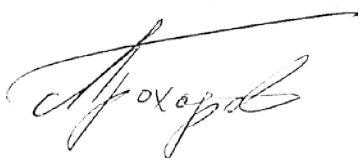
доктор физико-математических наук,
профессор
Подловченко Римма Ивановна

Ведущая организация: факультет Вычислительной Математики и Кибернетики МГУ им. М.В. Ломоносова

Защита диссертации состоится «21» мая 2008 г. в 15 часов на заседании диссертационного совета Д.002.087.01 при Институте Системного Программирования РАН по адресу:
109004, Москва, Б. Коммунистическая 25, Институт Системного Программирования РАН, конференц-зал.

С диссертацией можно ознакомиться в читальном зале библиотеки Института Системного Программирования РАН.

Автореферат диссертации разослан «__» _____ 2008 г.

Ученый секретарь
диссертационного совета
кандидат физико-математических
наук  /С.П.Прохоров/

Содержание:

Актуальность исследования

История информационных технологий – это, прежде всего, история их проникновения во все сферы деятельности человека: экономику, военную область, науку, образование, культуру и т.д. Они становятся всё более неотъемлемой частью инфраструктуры современной жизни. Этот процесс, естественно, вызывает интенсивное развитие самих этих технологий: создаются всё более мощные компьютеры и компьютерные системы, всё более сложные программы и программные комплексы.

Однако, чем шире распространяется использование компьютеров, чем более важные и ответственные задачи возлагаются на их «плечи», тем выше становится цена ошибок в программно-аппаратных средствах. Только в США потери от этих ошибок составляют от 20 до 60 млрд. долларов ежегодно, причём около 60% убытков терпят конечные пользователи. Складывается ситуация, когда производители выпускают, а потребители вынуждены оплачивать заведомо бракованный товар. Но дело не только в материальном ущербе: некорректное поведение компьютерных систем способно вызвать дезорганизацию управления государством, привести к техногенным и экологическим катастрофам, поставить под угрозу жизнь и здоровье людей.

Как показывает практика, чаще всего источником ошибок в компьютерных системах оказывается программное обеспечение (ПО), которое модифицируется гораздо чаще, чем аппаратура. Поэтому проблема повышения качества ПО становится всё более важной и актуальной. Программистские компании вынуждены тратить дорогостоящее время и не менее дорогие усилия разработчиков не только на то, чтобы создать нечто новое, но и гарантировать, что оно действительно решает нужные задачи, делает это правильно, надёжно и безопасно.

Ключевой частью решения этой проблемы является тестирование программ, часто требующее времени и ресурсов больше, чем создание этих программ. Из второстепенной и как бы «побочной» деятельности тестирование выходит на первый план, интегрируется с самим процессом разработки ПО и

существенно изменяет представления о том, как надо создавать сложные программные комплексы. Тестирование становится жёсткой «обратной связью», замыкает «жизненный цикл» разработки ПО и перестраивает пространственно-временную структуру процесса разработки и организацию коллектива разработчиков.

Противоречивая задача повышения качества программ и снижения затрат на тестирование диктует необходимость поиска путей автоматизации тестирования. Одним из важнейших методов решения этой задачи является тестирование на основе формальных (математических) моделей, целью которого является проверка того, что реализационная модель (модель тестируемой системы) соответствует (конформна) спецификации (модель требований). Конформность понимается как отношение «похожести» реализации на спецификацию. Если спецификационная модель и конформность определены строго формально, то становится возможным по спецификации автоматически генерировать тесты, проверяющие эту конформность.

В то же время внедрение в практику соответствующих методов тестирования сталкивается сегодня с двумя серьёзными препятствиями. Первое – это большое разнообразие типов моделей и, что ещё важнее, видов конформностей. Для многих конформностей в последние годы были созданы соответствующие математические теории, на основе которых разработаны методы генерации тестов и построены практически используемые тестовые системы. И этот процесс далёк от завершения: продолжает расти количество предлагаемых конформностей и основанных на них методов и инструментов тестирования. В результате тестировщик оказывается перед серьёзной проблемой выбора того, что ему подходит.

Причина такого экстенсивного роста не только, и не столько, во внутренней логике развития теории тестирования, сколько в потребностях практики: «спрос превышает предложение». Поэтому каждый раз, когда практическое тестирование сталкивается с проблемой, которая «не вписывается» в уже существующие теоретические подходы, поиск путей её

решения приводит к новому пониманию «похожести» реализации на спецификацию, «изобретению» новой конформности и созданию основанных на ней теории, методов и инструментов тестирования. При этом каждый автор, предлагающий свою конформность, формально начинает теорию «с нуля», и больше внимания уделяет тому, чтобы показать адекватность своей теории той практической проблеме, которая послужила стимулом к её созданию, чем поиску идей и методов, общих с другими теориями конформности.

Вместе с тем сегодня уже становится ясно, что такие общие идеи и методы «витают в воздухе», поскольку в математических формализмах различных конформностей и методах тестирования на их основе много общего. Поэтому задача выявления этих общих идей и методов, задача создания обобщающей теории конформности, покрывающей широкий класс практически используемых или требуемых конформностей, стала чрезвычайно актуальной как для теории, так и для практики, и впервые наметились пути её решения.

Среди всех видов тестирования наиболее важным для практики является функциональное тестирование. Это тестирование методом «чёрного ящика», когда внутреннее устройство реализации, её внутренние работа и состояния не наблюдаемы, и о правильности или ошибочности реализации судят по её внешнему поведению в ответ на внешние тестовые воздействия. Именно набор этих тестовых воздействий и возможных наблюдений над ответным поведением реализации может быть различным в различных практических случаях. Этот набор определяет ту или иную семантику тестового взаимодействия, которая, в свою очередь, определяет ту или иную конформность. Для программных систем чаще всего используется отношение редукции – «сводимости» реализации к спецификации, когда спецификация предоставляет на выбор несколько вариантов правильного поведения, а правильная программа демонстрирует лишь некоторые из них.

Второе серьёзное препятствие, с которым сталкивается тестирование на основе формальных моделей, – это неразвитость теории тестирования сложных, иерархически построенных систем, состоящих из многих компонентов. Помимо

ошибок в реализациях компонентов, здесь появляется новый вид архитектурных ошибок, вызванных тем, что требования к системе некорректно декомпозированы в требования к её компонентам. Такие ошибки не только труднее обнаруживать, но они имеют более печальные последствия: изменение архитектуры системы и переделка всех или части её компонентов. В существующей теории тестирования проблема верификации декомпозиции системных требований не имеет общего решения. Предлагаются лишь частичные решения этой проблемы для некоторых конформностей.

Таким образом, актуальной задачей является создание теории конформности («сводимости» реализации к спецификации при функциональном тестировании), которая предлагала бы как методы автоматической генерации тестов по спецификации, так и методы автоматической верификации декомпозиции системных требований для сложных систем, состоящих из многих компонентов.

Решению этой задачи посвящена данная диссертация.

Общая характеристика работы

В работе рассматривается класс конформностей типа редукции, основанных на наблюдаемом поведении и параметризуемых семантикой тестового взаимодействия при функциональном тестировании. Семантика определяется набором допустимых тестовых воздействий и возможных ответных наблюдений над поведением системы. Наблюдения бывают двух типов: наблюдение внешнего действия, выполняемого реализацией, и, в некоторых случаях, наблюдение отказа – отсутствия действий. В начестве основной модели в диссертации выбрана система помеченных переходов (Labelled Transition System – LTS).

Для того, чтобы каждое тестовое воздействие завершалось наблюдением, не должно возникать ненаблюдаемых отказов и дивергенции («зацикливание») – бесконечной внутренней работы системы без взаимодействия с тестом. Кроме

этого, в диссертации впервые вводится понятие разрушения, моделирующего поведение, которого не должно быть при тестировании по тем или иным причинам. **Предлагается новая концепция безопасного тестирования**, которое избегает ненаблюдаемых отказов, дивергенции и разрушения. Вводится гипотеза о безопасности, определяющая класс безопасно-тестируемых реализаций, и основанная на ней безопасная конформность. Для этой конформности предлагается метод генерации безопасных тестов по спецификации, параметризуемый семантикой тестового взаимодействия. Этот метод является обобщением методов генерации тестов для частных случаев, известных в теории и применяемых на практике. Для этих частных случаев он даёт те же результаты, но область его применения гораздо шире.

Также **в диссертации рассматривается асинхронное тестирование** (тестирование в контексте), когда взаимодействие теста и реализации происходит не напрямую, а через ту или иную промежуточную среду взаимодействия. Проблема несохранения конформности заключается в том, что асинхронные тесты могут ловить ложные ошибки, то есть те, которые не обнаруживаются при синхронном тестировании, когда тест и реализация взаимодействуют непосредственно друг с другом без какой-либо буферизации в среде. Это является особым случаем общей проблемы композиции, когда составная система, собранная из реализаций компонентов, конформных своим спецификациям, оказывается неконформной композиции этих спецификаций. Тесты, сгенерированные по такой композиции спецификаций, также могут ловить ложные ошибки.

В диссертации предлагается общее решение проблемы несохранения конформности при композиции, в том числе, при асинхронном тестировании с различными средами взаимодействия. Решение основано на специальном алгоритмическом преобразовании спецификаций. Преобразованная спецификация эквивалентна исходной спецификации в том смысле, что определяет тот же класс конформных реализаций и не суженный класс безопасно-тестируемых реализаций. Для преобразованных спецификаций

конформность уже сохраняется при композиции и при асинхронном тестировании.

Решение проблемы композиции впервые даёт **полное решение проблемы верификации декомпозиции системных требований** в рассматриваемой области. Спецификация составной системы оказывается согласованной со спецификациями компонентов тогда и только тогда, когда ей конформна композиция преобразованных спецификаций. Это можно проверить после преобразования спецификаций компонентов и их алгоритмической композиции с помощью методов, основанных на генерации тестов по заданной спецификации системы.

В практическом плане результаты диссертационной работы сводятся к предлагаемым методам генерации тестов и преобразования спецификаций. Это даёт, во-первых, возможность создавать тестовые системы для самых разных семантик тестового взаимодействия на базе единого подхода с использованием общего метода генерации тестов. Во-вторых, в сочетании с преобразованием спецификации метод применим для асинхронного тестирования с самыми разными средами взаимодействия. В-третьих, заложена основа для построения инструментов автоматической верификации декомпозиции системных требований, причём спецификации компонентов и системы в целом могут пониматься в самых разных семантиках тестового взаимодействия.

Область исследования

Тестирование на основе формальных моделей

Тестирование на основе формальных моделей – это новая дисциплина, сложившаяся последние десятилетия на стыке теории и практики информационных технологий. Это актуальное направление во многом опирается на работы различных исследователей, выполненные в 80-х и в 90-х годах прошлого века и посвященные проблемам тестирования

телекоммуникационных протоколов. Самые первые же статьи, которые с полным правом можно отнести к этой области, появились еще в конце 60-х – начале 70-х годов. Тем не менее, серьезный практический и массовый интерес к результатам в этой области возник только на рубеже веков, в связи с востребованностью эффективных методов контроля и обеспечения качества сложных программных и программно-аппаратных систем в индустрии. Тестирование на основе моделей начинает применяться в промышленных компаниях. Первые же успехи (и, в не меньшей степени, первые неудачи) сделали разработку формальных методов тестирования чрезвычайно актуальными на сегодняшний день.

Основная идея тестирования на основе моделей вполне прозрачна: поскольку сложная система не поддается проверке «в лоб», создайте сначала более простую ее *модель*, описав в ней *только то, что для вас важно* на данном этапе, а затем стройте тесты *только* на основании полученной модели. Такой подход хорошо работает на практике, если саму модель и набор тестов удаётся строить по частям (инкрементально), постепенно отражая в модели набор свойств реальной системы и, соответственно, расширяя охват тестами различных аспектов работы системы. Как возможность такой инкрементальной разработки моделей и тестов, так и конкретные техники, используемые при этом, существенно зависят от самого *вида* применяемых моделей, и от того, какие имеются *возможности по управлению* тестируемой системой и *наблюдению* ее поведения.

Модели должны быть одновременно достаточно просты, чтобы поддаваться строгому анализу, и достаточно универсальны, чтобы описывать большой класс практически важных систем. Возможность строгого анализа свойств модели становится актуальной, если необходимы определенные гарантии того, что все важные аспекты поведения системы покрываются построенными тестами. Такие модели должны быть *формальны*, то есть описаны на языке математики. Само же тестирование строится на некоторой математической теории, предписывающей определенные способы построения

тестов, которые обеспечат им нужные свойства полноты, а тестированию придадут необходимую глубину и надежность.

Тестирование на основе моделей проверяет соответствие тестируемой системы требованиям. Предполагается, что требования к системе выражены в терминах её взаимодействия с внешним миром (окружением), что как раз и даёт возможность проверять их выполнение в тестовом эксперименте. На формальном уровне соответствие системы требованиям означает, что модель тестируемой системы и модель требований связаны заданным математическим соответствием (бинарным отношением) (рис.1).

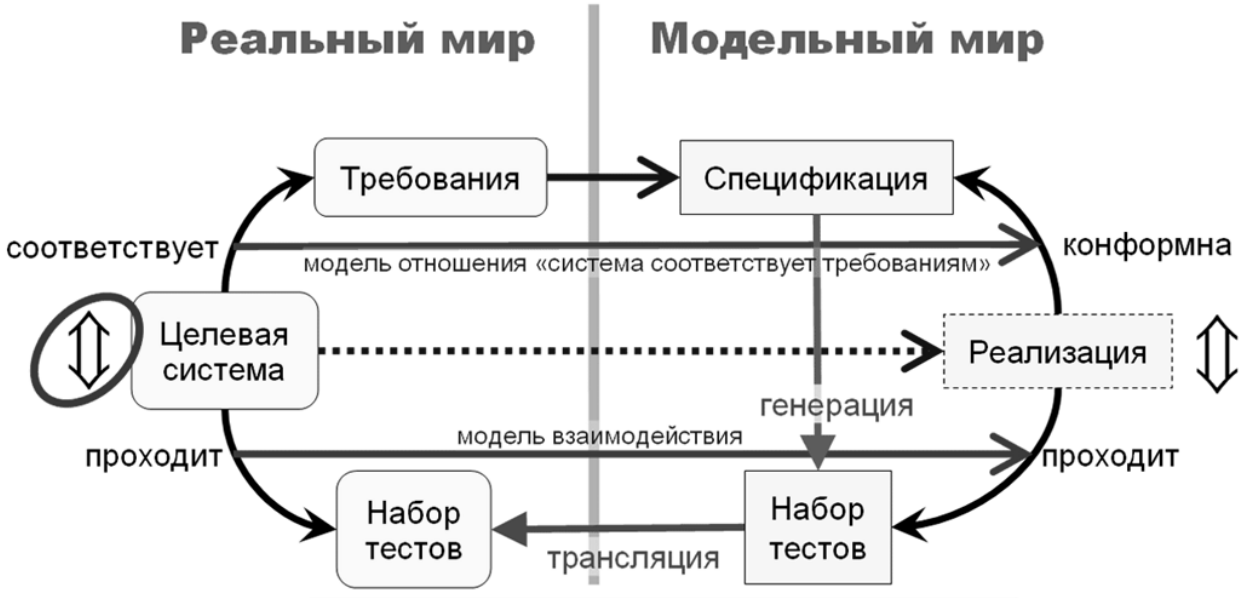


Рис.1

В диссертационной работе модель тестируемой системы называется реализационной моделью или (для краткости) *реализацией*, модель требований – *спецификацией*, а их соответствие – (модельной) *конформностью*.[1] При тестировании (в отличие от аналитической верификации) предполагается, что реализационная модель не задана или слишком сложна для анализа. Поэтому

[1] Это соответствует общепринятым терминам в англоязычной литературе: *implementation, specification* и *conformance*. Отметим, что в русскоязычной литературе до сих пор не сложилось единой терминологии. Слово «соответствие» применяется несколько чаще, чем слово «конформность», но в русском языке оно имеет более широкое смысловое поле, что иногда приводит к недоразумениям. Более того, английское словосочетание "*conformance testing*" в текстах на русском языке встречается чаще, чем все варианты русских словосочетаний (по данным информационно-поисковой системы *Google*).

основной задачей является генерация по заданной спецификации модельных тестов, назначение которых – проверить конформность любой реализационной модели из рассматриваемого класса. Эта проверка основана на модели взаимодействия реальной системы с её окружением. Такую модель, как сказано выше, называют *семантикой взаимодействия*. Вводится модельное отношение «реализация проходит тест». Набор тестов принято называть полным, если реализация проходит каждый тест из набора тогда и только тогда, когда она конформна спецификации. После того, как тесты созданы, они транслируются в реальные тестовые программы, которые, взаимодействуя с реальной системой, устанавливают её соответствие или несоответствие исходным требованиям. Эта схема работы основана на следующих предположениях: во-первых, адекватны модели объектов (реализация и спецификация) и отношений (конформность и семантика взаимодействия), во-вторых, правильно выполняются преобразования (генерация модельных тестов и их трансляция в реальные тестовые программы).

Спецификация неоднозначно определяет реализацию, оставляя разработчику значительную свободу в рамках специфицированных требований. Как именно нужно понимать эти требования, в чём разработчик свободен, а в чём «скован» требованиями, как раз и описывается конформностью. Оно определяется предполагаемыми *правилами взаимодействия* системы с окружением, которые должны быть адекватно отражены на уровне моделей. Спецификация не требует от реализации правильного поведения при взаимодействии с любым её окружением, а только с таким, которое само ведёт себя правильно. Например, в терминах пред- и постусловий тесты должны проверять выполнение постусловия (программа работает правильно) только при таком взаимодействии, которое не нарушает предусловий (к программе правильно обратились). При тестировании правила взаимодействия как раз и определяют те тестовые возможности по управлению и наблюдению за тестируемой системой, которые необходимы и достаточны для проверки конформности.

Выбор модели

По поводу вида моделей, которые нужно использовать для описания сложных систем, большинство исследователей сегодня еще не пришли к общему мнению. Можно выделить три вида моделей в порядке их усложнения.

Самая простая модель – математическая функция как абстракция реентерабельной процедуры. Основная проблема здесь – большое количество значений аргумента, которые нужно проверить для того, чтобы убедиться, что результат всегда правильный. Особенно, когда аргумент – это большие данные со сложной структурой как, например, для компиляторов.

Вторая модель – это автомат (*finite automata, finite state machines*), взаимодействие с которым сводится к той же простой схеме «стимул-реакция», что и для функции. В отличие от функции результат зависит не только от аргумента, но и от состояния автомата, которое является абстракцией таких объектов как глобальные переменные или поля данных объектов классов. Здесь добавляется проблема перебора состояний. Сложность в том, что состояние может быть не наблюдаемо при тестировании.

Третья модель – это система помеченных переходов – LTS (*Labelled Transition System*). Для таких систем характерно сложное взаимодействие. Например, в ответ на стимул может быть несколько реакций или ни одной. Можно подавать несколько стимулов подряд или получать реакции без стимулов. Кроме того, система может выполнять внутренние действия, не наблюдаемые извне. В этих случаях, когда взаимодействие с системой сводится к обмену сообщениями: стимулами и реакциями, такая система называется *реактивной*. В общем случае говорят просто о внешних действиях, наблюдаемых при взаимодействии.

Для LTS характерны такие явления как недетерминизм, отказы, когда нет никаких действий, а также дивергенция – зацикливание, когда система выполняет только внутреннюю работу и не взаимодействует с тестом. В данной работе внимание концентрируется, прежде всего, на этом.

Для систем последнего типа применяются и другие модели: сети Петри, алгебраические спецификации, в частности ASM (*Abstract State Machine*), а также контрактные спецификации, построенные на пред- и постусловиях.

Можно назвать три причины, по которым в данной работе выбрана модель LTS:

- Распространённость этой модели.
- Удобство генерации тестов по LTS.
- На LTS определена композиция, моделирующая взаимодействие компонентов составной системы.

Взаимодействие между LTS-моделями понимается как синхронное: передатчик передаёт сообщение, а приёмник принимает его без какой-либо буферизации. Это моделируется оператором параллельной композиции той или иной алгебры процессов. В диссертации выбрана композиция $\upharpoonleft\downharpoonright$ в духе алгебры процессов CCS (Calculus of Communicating Systems), предложенной Милнером. Выдача сообщения x обозначается как $!x$, а приём – как $?x$; внутренние действия обозначаются символом τ.

Функциональное тестирование и конформность типа редукции

Что касается конформности, которую следует выбирать для построения тестов, то на этот счёт имеются десятки различных точек зрения. В 1990-93 г.г. Ван Глаббек сумел классифицировать для LTS многие из конформностей, описав 155 семантик тестового взаимодействия, на которых они основаны. При этом он не учитывал семантики, специфические для реактивных систем, когда принципиально различаются передача стимулов и передача реакций.

В диссертационной работе исследуется тестирование, которое основано только на наблюдаемом поведении системы. Такое тестирование называют тестированием «чёрного ящика»: тест может только извне взаимодействовать с системой, осуществляя тестовые воздействия и наблюдая внешнее поведение системы, но «не знает и не видит» её внутреннего устройства, отражаемого в понятии состояния, и её внутреннего поведения, не наблюдаемого извне. Такое тестирование называют также *функциональным*, акцентируя внимание на том,

что оно проверяет только те свойства системы, которые отражаются в её внешней функциональности и наблюдаемы при её внешнем функционировании. Понятно, что такое тестирование имеет важное практическое значение, поскольку во многих реальных случаях внутренние действия и состояния реализации не наблюдаемы из теста (например, при удалённом взаимодействии). Но более важно то, что учёт состояний и внутреннего поведения может оказаться «переспецификацией», требуя от реализации того, что никак не проявляется при взаимодействии с ней.

При тестировании методом «чёрного ящика» мы можем наблюдать только такое поведение реализации, которое, во-первых, «спровоцировано» тестом (управление) и, во-вторых, наблюдаемо во внешнем взаимодействии. Такое взаимодействие моделируется с помощью, так называемой, машины тестирования (Милнер и Ван Глаббек). Она представляет собой «чёрный ящик», внутри которого находится реализация. Управление сводится к тому, что оператор машины, выполняя тест, нажимает кнопки, «приказывая» или «разрешая» реализации выполнять те или иные действия, которые могут им наблюдаться. Наблюдения (на «дисплее» машины) бывают двух типов: наблюдение некоторого *внешнего действия*, разрешённого оператором и выполняемого реализацией, и наблюдение *отказа* как отсутствия каких бы то ни было действий из числа тех, что разрешены нажатыми кнопками. Важно отметить, что отказ принципиально не сводится к какому-либо внешнему действию, вроде реакции «нет действий», поскольку, в отличие от действия, гарантированно не меняет состояния реализации.

Результатом тестового эксперимента является трасса (последовательность) наблюдений: действий и отказов. В соответствующих конформностях, тем самым, не учитываются состояния. Поэтому в диссертации не рассматриваются конформности типа *симуляций*, которые основаны на соответствии состояний реализации и спецификации.

Также не рассматриваются конформности типа *эквивалентностей*, когда внешнее поведение реализации и спецификации совпадают. В диссертационной

работе внимание сосредоточено на конформностях типа *редукции* (сводимости) реализации к спецификации. Суть такой конформности в том, что конформная реализация не должна демонстрировать внешнее поведение, которое так или иначе запрещено спецификаций, но спецификация может предоставлять на выбор несколько вариантов правильного поведения, и конформная реализация может демонстрировать лишь некоторые из них. Разумеется, речь идёт о поведении в реализации и спецификации в одной и той же ситуации, то есть после одной и той же трассы наблюдений.

К рассматриваемому классу семантик и конформностей относятся: трассовая семантика (*trace semantics*) и трассовый предпорядок (***tr*** – *trace preorder*); семантика завершённых трасс (*completed trace semantics*) и предпорядок завершённых трасс (***ct*** – *completed trace preorder*); семантика трасс с отказами (*failure trace semantics* или *refusal semantics*) и предпорядок трасс с отказами (*failure trace preorder* или ***rf*** – *refusal preorder*); семантика для реактивных систем без блокировки стимулов, но с возможностью наблюдения отсутствия реакций, и конформности ***ior*** (*input-output refusal relation* или *repetitive quiescence relation*) и ***ioco*** (*input-output conformance*); семантика для реактивных систем с блокировками стимулов и конформность $\boldsymbol{ioco}_{\beta\delta}$ (модификация отношения ***ioco***, учитывающая блокировку стимулов); семантика для систем с мультиканалами стимулов и реакций (*MIOTS – Multi Input-Output Transition Systems*) и конформность ***mioco*** (модификация конформности ***ioco*** для таких систем) и др.

Как дальнейшее развитие теории тестирования, так и её практическое применение сталкиваются с рядом проблем, решению которых посвящена диссертационная работа.

Проблематика

Выбор семантики взаимодействия и генерация тестов

Прежде всего, на практике приходится сталкиваться с разнообразием семантик тестового взаимодействия, которое строится на пересечении того, что мы можем, и того, что нам нужно. Речь идёт о тестовых возможностях: какие есть тестовые воздействия и какие есть наблюдения. То, что мы можем или не можем, – это ограничение тестовых возможностей «сверху». То, что нам требуется для проверки конформности, – это ограничение «снизу».

Отношение ***ioco*** (*Input-Output Conformance*), предложенное Яном Тритмансом, является одной из наиболее распространённых, хорошо проработанных в теории и зарекомендовавших себя на практике конформностей типа редукции для реактивных систем. Тестовые воздействия двух типов: можно либо послать в реализацию один выбранный стимул, либо принять из реализации одну, но любую из выдаваемых ею реакций. Если тест принимает реакции, он всегда что-нибудь наблюдает: либо реакцию, либо отсутствие реакций (например, по тайм-ауту). Отсутствие реакций называется *стационарностью* (*quiescence*). Если тест посылает стимул, он может наблюдать только приём стимула реализацией. Соответствующий отказ, когда реализация отвергает стимул, называется *блокировкой стимула* (*input refusal*) и считается ненаблюдаемым. Это ограничение «сверху» на наши тестовые возможности.

Для ***ioco*** создано несколько инструментов для генерации тестов, в том числе: TGV (Test Generation with Verification technology), интегрированный в CADP (Construction and Analysis of Distributed Processes Software Tools for Designing Reliable Protocols and Systems), TestGen (входной язык LOTOS, генерирует тест для верификации hardware design и компонентов), TorX (генерация тестов, выполнение их и анализ результатов тестирования on-fly).

В то же время существует целый ряд практических систем, для тестирования которых ***ioco***-семантики недостаточно. Приведём ряд примеров.

Блокировка стимулов обычно (в частности, для ***ioco***) считается ненаблюдаемой (ограничение «сверху»), но такое наблюдение требуется (ограничение «снизу») для тестирования сред взаимодействия ограниченной ёмкости, в частности, очередей ограниченной длины. Здесь стимул – это помещение в очередь, а реакция – выборка из очереди. Когда очередь полностью заполнена, стимул должен блокироваться и это нужно уметь проверять.

Другим примером наблюдения блокировок могут служить системы с графическим интерфейсом. Широко распространена ситуация, когда в окне высвечивается меню, в котором некоторые кнопки «бледные», то есть соответствующие стимулы блокируются. Это существенно отличается от приёма стимула с последующей выдачей соответствующей реакции, когда кнопка не «бледная», а в ответ на её нажатие всплывает окошко с надписью «операция не может быть выполнена». Ещё один пример – автомат по продаже «чего-нибудь», который блокирует приём монет, когда это «чего-нибудь» кончается. Это также отличается от автомата, в котором монета в любом случае принимается, но потом возвращается в окошке для сдачи.

Обычно (в частности, для ***ioco***) тест посылает в реализацию стимулы по одному. Необходимость (ограничение «снизу») в передаче сразу нескольких стимулов, из которых реализация сама выбирает один стимул для приёма, требуется для тестирования систем с несколькими входными портами, когда спецификация разрешает любой порядок приёма из портов (система **F** на рис.2).

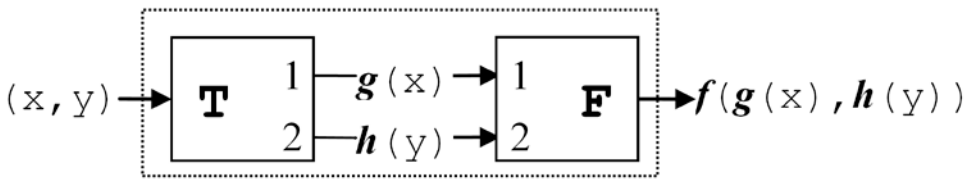


Рис.2

После передачи одного стимула, тест предлагает реализации оставшиеся стимулы и так далее до тех пор, пока не будут переданы стимулы на все входные порты реализации. Этот пример представляет класс пороговых систем,

которые сначала принимают в любом порядке несколько стимулов на разные входные порты, а потом выдают реакцию.

Такая система-приёмник может получать стимулы от другой системы-передатчика, в которой, наоборот, имеется несколько выходных портов и для которой передаваемые стимулы являются реакциями (система **T** на рис.2). Чтобы обеспечить передачу всех сообщений при любом порядке их приёма, мы вынуждены (ограничение «снизу») потребовать от передатчика того же, что требуем от теста приёмника: начинать с выдачи сразу всех сообщений. Выдача двух реакций последовательно: сначала по первому порту, а потом по второму, или наоборот, является ошибкой. Если тест, как это чаще всего бывает (в частности, для ***ioco***), принимает любую реакцию, то по какому бы порту ни пришла реакция, это приходится считать правильным, и ошибка не обнаруживается. Чтобы обнаружить ошибку, нам нужно уметь в тесте принимать не все реакции, а только из одного порта: один тест обнаружит ошибку как отсутствие реакций по первому порту, а другой – по второму. Та же самая тестовая возможность требуется для тестирования широковещания: для каждого выходного порта нужно проверить, что тест, принимающий реакции из этого порта, не обнаружит их отсутствие.

Совмещение передачи стимулов с приёмом реакций, обычно являющееся излишним (не увеличивающим мощность тестирования), необходимо (ограничение «снизу») для тестирования систем с откатами (рис.3).

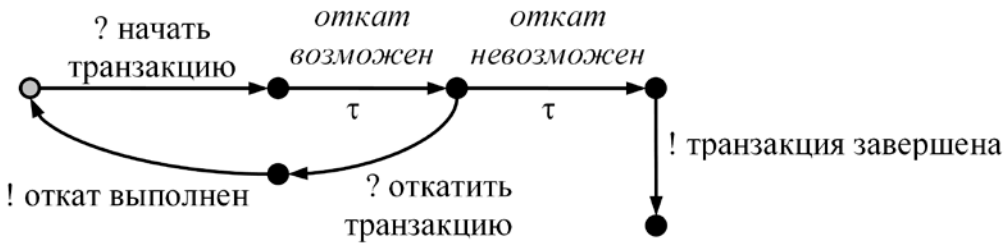


Рис.3

В такой системе два типа стимулов: «начать транзакцию» и «откатить транзакцию», и, соответственно, два типа реакций: «транзакция завершена» и «откат выполнен». После начала транзакции реализация часто выполняет её в два этапа так, что откат возможен только на первом этапе, а на втором этапе

стимул «откатить» отвергается, то есть блокируется. Если блокировки ненаблюдаемы (ограничение «сверху»), мы не можем давать команду «откатить» и, тем самым, не можем тестировать интересующие нас реализации. Однако мы можем обойти это ограничение «сверху», если одновременно с посылкой стимула «откатить» будем принимать реакции. Отказа не будет, так как всегда либо будет принят откат, либо выдана реакция «транзакция завершена».

В целом актуальна задача создания теории конформности, применимой для различных семантик тестового взаимодействия, значимых для практики.

Поведение, не наблюдаемое при тестировании

Кроме наблюдаемого поведения (действия и отказы) реализация может иметь поведение, которого следует избегать при тестировании по тем или иным причинам. Можно выделить три типа такого поведения.

Первый тип – это **ненаблюдаемый отказ**. Его возникновение плохо тем, что мы не получим никакого наблюдения в ответ на тестовое воздействие: действий нет, а отказ не наблюдаем. Обычно предполагается, что в тестируемой реализации вообще нет ненаблюдаемых отказов. Для отношения ***ioco*** реализация должна принимать стимулы в каждом своём состоянии (*input-enabled*). В то же время такое требование слишком сильное.

Например, на рис.4 приведена система A, взаимодействие с которой происходит по «автоматному» правилу «стимул-реакция». Спецификация – LTS A_0. Формально, она не может служить своей собственной тестируемой реализацией, так как в ней есть блокировка стимула. В то же время тесты, построенные для ***ioco***, не только не вызовут блокировку стимула (после наблюдения трассы стимул не посылается, если его нет в спецификации после этой трассы), но и не обнаружат никаких ошибок. Эти тесты вообще не отличат LTS A_0 от тестируемой и конформной LTS A_1.

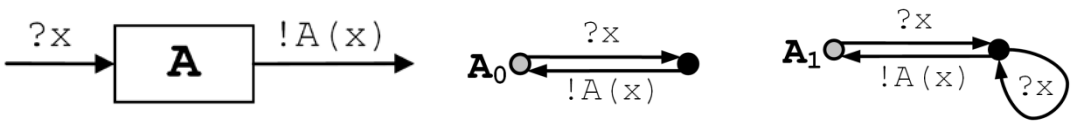


Рис.4

Второй тип – это **дивергенция** – бесконечная последовательность внутренних действий системы, «зацикливание». При дивергенции мы также можем не получить в ответ на тестовое воздействие никакого наблюдения просто потому, что всё время будут выполняться внутренние действия. Проблеме дивергенции посвящено довольно много теоретических работ. Однако в теориях, имеющих выход в практику, то есть развитых до алгоритмов генерации тестов, обычно просто предполагается, что в реализации не должно быть дивергенции.

В то же время проблема дивергенции не в том, что она может возникнуть в реализации, а в том, что она не должна возникать при том или ином взаимодействии. К такому взаимодействию, в частности, относится взаимодействие реализации как компонента сложной, составной системы с другими заданными компонентами. Эти другие компоненты взаимодействуют с реализацией не произвольным, а вполне определённым образом, из-за чего имеющаяся в реализации дивергенция может стать недостижимой. Есть и обратная проблема: даже при отсутствии дивергенции в компонентах она может возникнуть в их композиции как бесконечное взаимодействие этих компонентов между собой. При иерархическом построении системы такая дивергенция может, в свою очередь, оказаться промежуточной, то есть исчезающей при дальнейших композициях. Поэтому, запрещая дивергенцию на всех уровнях, мы выбрасываем из рассмотрения целый класс работоспособных систем.

Таким образом, представляется важным и полезным допустить наличие дивергенции в реализации, но при тестировании избегать тестовых воздействий в ситуации, когда возможна дивергенция.

Третий тип – это **поведение, которое запрещено при тестировании** по тем или иным причинам. Этой теме посвящено очень мало работ. Лишь в

редких случаях вводятся запрещённые состояния, в которые реализация не должна попадать. Между тем, это довольно частое явление. Например, запрещённым можно считать поведение программы при нарушении предусловия обращения к ней. Тестирование такого поведения не только бессмысленно, но может быть и опасным, если тестируемая система имеет высокий уровень доступа к привилегированным программам и данным. В частности, эта тема весьма актуальна в теории и практике информационной безопасности, где система считается безопасной, если в ней нет недекларированного поведения, которое можно понимать как запрещённое. В то же время, такая система может быть построена из компонентов, которые сами по себе не являются безопасными, но запрещённое поведение недостижимо при работе системы в целом за счёт того, что каждый компонент взаимодействует не с произвольным окружением, а с заданными компонентами системы.

В целом, является актуальной задача расмотрения в теории конформности всех трёх указанных типов ненаблюдаемого поведения. Спецификация должна позволять описывать случаи, когда в реализации допустимы (не влияют на конформность) **ненаблюдаемые отказы, дивергенция и запрещённое поведение.** В то же время **при тестировании их следует избегать**, что требует определённой модификации самой конформности**.** Особое внимание должно быть уделено дивергенции и запрещённому поведению в случае сложной системы, собранной из отдельных компонентов.

Асинхронное тестирование

Ещё одной проблемой является степень доступа теста к реализации. На практике чаще приходится иметь дело не с синхронным, а с асинхронным взаимодействием теста и реализации. В стандарте ISO асинхронное тестирование определяется как тестирование реализации, помещённой в тестовый контекст. Чаще всего такой контекст понимают как пару неограниченных очередей, буферизующих стимулы и реакции. Чтобы

правильно оценивать результаты тестирования, мы должны понимать, что в реализации проходится не обязательно та трасса, которую мы наблюдаем, как это имело место при синхронном тестировании. Наблюдаемой трассе соответствует, вообще говоря, множество синхронных трасс, которые могла бы пройти реализация при таком наблюдении. Ошибка фиксируется, если ни одной из этих трасс нет в спецификации. Этот процесс называется сериализацией. Кроме того, ослабляются требования к допустимым тестовым воздействиям, поскольку теперь никаких блокировок нет: входная очередь всегда принимает посылаемый стимул. Дивергенцию и разрушение в реализации мы, по-прежнему, должны учитывать.

В общем случае тестовый контекст можно понимать как среду взаимодействия, которая моделируется подходящей LTS и компонуется с LTS-реализацией. Это может быть несколько входных и выходных очередей, очереди с приоритетами, стеки, порядок передачи может нарушаться, стимулы и реакции могут пропадать или генерироваться лишние и так далее. Для каждой такой среды определение допустимых тестовых воздействий и сериализацию нужно делать по своим правилам. Составление правил такого тестирования – это интеллектуальный труд, далеко не всегда простой.

Решением является общий метод, когда тесты генерируются не по исходной спецификации с учётом этих правил, а по композиции спецификации со средой, и по ней же проверяются наблюдения. Однако здесь возникает, так называемая, проблема несохранения конформности. Дело в том, что асинхронные тесты могут ловить ложные ошибки, то есть ошибки, не обнаруживаемые при более полном синхронном тестировании. Продемонстрируем эту проблему на двух примерах.

Первый пример – это система на рис.4, рассматриваемая в ***ioco***-семантике (семантике с ненаблюдаемыми блокировками). При асинхронном тестировании через неограниченную входную очередь всегда можно послать два стимула x подряд: очередь их примет, а реализация потом будет выбирать стимулы из очереди. Если после этого принимать реакции, то, согласно спецификации A_0,

должны быть последовательно приняты две реакции A(x). В то же время, хотя реализация A_1 конформна, при асинхронном тестировании в ней будет обнаружена ошибка, поскольку второй реакции может не быть (стационарность в начальном состоянии).

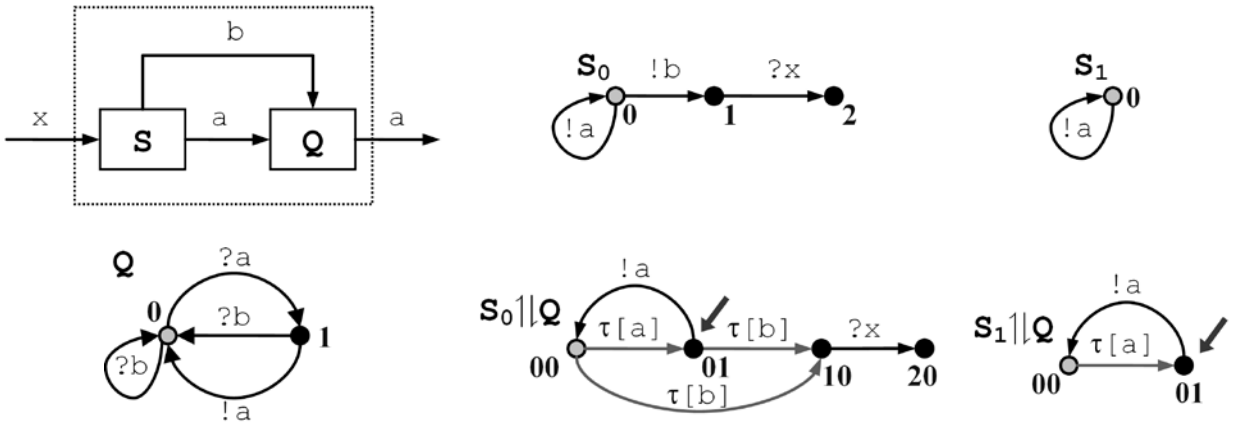


Рис.5

Второй пример – асинхронное тестирование системы S с помощью среды взаимодействия Q (рис.5). Среда – это одна ограниченная выходная очередь (на рисунке длины 1) с дополнительной командой «обнуления» (b). Спецификация S_0 рассматривается в семантике с наблюдаемыми блокировками (ненаблюдаемых отказов нет). Она описывает следующие требования к системе: сначала стимул блокируется, но можно выдавать цепочку реакций a, потом можно обнулить очередь (b), принять стимул x и закончить в терминальном состоянии. Реализация S_1 конформна: она не обнуляет очередь и, соответственно, не принимает стимул. Когда с очередью Q компонуется спецификация, первый посылаемый стимул не блокируется. Однако при композиции реализации S_1 такая блокировка появляется, что при асинхронном тестировании будет квалифицировано как ошибка (показано стрелками).

Таким образом, важным является решение проблемы несохранения конформности при асинхронном тестировании с различными средами взаимодействия.

Верификация декомпозиции системных требований

Последние годы большое внимание уделяется тестированию составных систем, которые компонуются инкрементально и иерархически из компонентов разных уровней. Основная проблема композиции – это проблема соотношения спецификации системы и спецификаций её компонентов. Это соотношение корректно, если композиция компонентов, конформных своим спецификациям, всегда конформна спецификации системы. К сожалению, это не всегда так. Проверку правильности такого соотношения называют верификацией декомпозиции системных требований.

Конечно, если спецификация системы никак не связана со спецификациями компонентов, то трудно ожидать их правильного соотношения. Казалось бы, спецификацию системы можно получить как композицию спецификаций компонентов, выполняя её по тем же правилам, по которым сама система компонуется из реализаций компонентов. Но оказывается, что полученная таким способом спецификация системы может быть некорректной. Эта проблема называется проблемой сохранения конформности при композиции или *проблемой монотонности*. В диссертации проблема несохранения конформности при асинхронном тестировании рассматривается как особый случай общей проблемы монотонности.

Примером может служить система, составленная из двух компонентов: передатчика и приёмника на рис.2. При несогласованности спецификаций компонентов может возникнуть тупик во взаимодействии: передатчик начинает с передачи сообщения через один порт, а приёмник начинает приём через другой порт (рис.6). Если спецификация системы в целом не предусматривает такого тупика, то это означает, что спецификации компонентов с ней не согласованы.

Для согласования требуется либо выдача передатчиком сообщений сразу по двум портам (после передачи сообщения через один порт оставшееся сообщение передаётся через другой порт) – строка 1, либо приём приёмником сообщений сразу по двум портам (после приёма сообщения через один порт

принимается оставшееся сообщение через другой порт) – столбец 1, либо и то и другое.

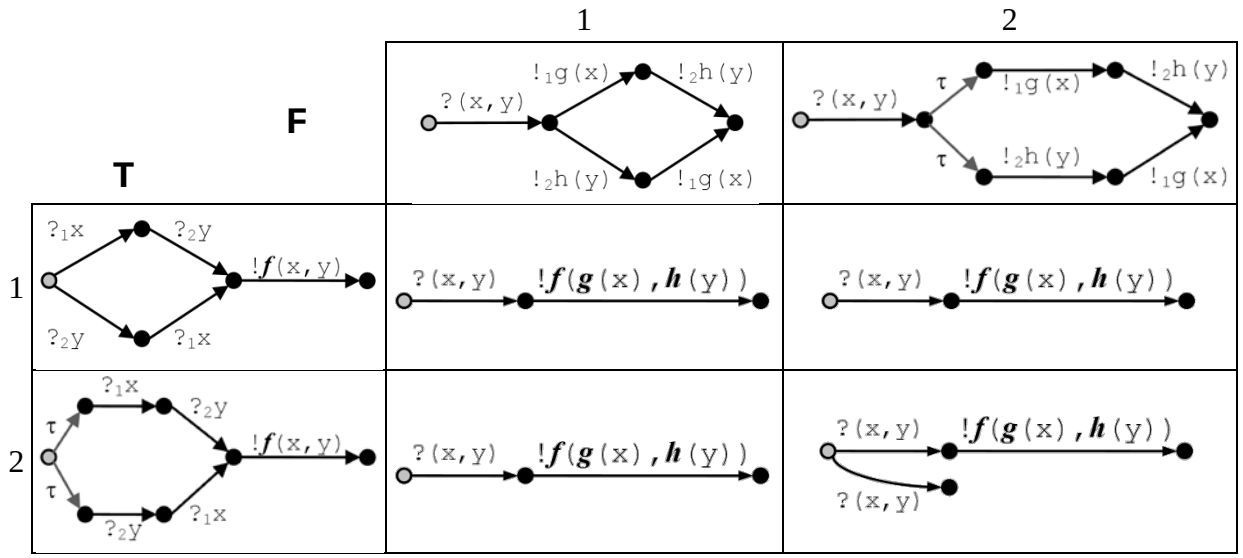


Рис.6

Отсутствие решения проблемы монотонности препятствует верификации декомпозиции системных требований. Наоборот, если найти такое решение, то можно **верифицировать согласованность имеющейся спецификации системы со спецификациями компонентов и автоматически получить корректную спецификацию системы по заданным спецификациям компонентов**. Сгенерированную спецификацию системы можно использовать как описание системы для её пользователей, как «техническое задание» разработчику системы при дальнейших модификациях системы и её компонентов, когда предполагается сохранение внешней функциональности системы, и, наконец, для генерации системных тестов.

Цели и задачи исследования

Целью диссертационной работы является создание теории конформности, которая обладала бы следующими свойствами:

1. могла применяться к широкому классу семантик взаимодействия, основанных на наблюдаемом поведении, с параметризацией конформности типа редукции той или иной семантикой из этого класса;

2. учитывала ненаблюдаемые отказы, дивергенцию и запрещённое поведение, допустимые в реализации, а не просто запрещала их всегда;
3. была развита до уровня алгоритмов генерации тестов;
4. решала проблемы сохранения конформности при асинхронном тестировании и верификации декомпозиции системных требований.

Достижение этой цели требует решения следующих задач.

1. **Формализация тестового эксперимента** с целью определения семантики взаимодействия, основанной на наблюдаемом поведении и параметризуемой тестовыми возможностями по управлению и наблюдению. Элементы такой семантики описывают управление и наблюдение и состоят из: 1) *действий*, которые может выполнять реализация и которые наблюдаемы тестом, 2) *кнопок*, на которых «написаны» множества разрешаемых каждой кнопкой действий (элемент управления), 3) *наблюдаемых отказов* – для тех «кнопочных» множеств, для которых возможно наблюдение отсутствия действий. Параметризация должна обеспечиваться возможностью задания любых (но согласованных между собой) наборов таких элементов.

2. **Построение трассовой теории**, в которой модель – это множество трасс наблюдений (действий и отказов) с ограничениями, вытекающими из семантики взаимодействия. С целью внесения в теорию дивергенции и запрещённого поведения следует предусмотреть в спецификации возможность завершать такие трассы специальными действиями, моделирующими дивергенцию и запрещённое поведение (действие, названное *разрушением*).

Определение в рамках трассовой теории для заданной семантики взаимодействия следующих понятий: 1) безопасное тестирование как тестирование, избегающее ненаблюдаемых отказов, дивергенции и разрушения в реализации, 2) гипотеза о безопасности реализации, определяющая класс безопасных реализаций, то есть реализаций, которые могут безопасно тестироваться для данной спецификации, 3) безопасная конформность: конформность типа редукции, определяющая класс конформных реализаций для заданной спецификации и проверяемая при безопасном тестировании.

Разработка методов генерации наборов тестов по заданной трассовой спецификации в заданной семантике взаимодействия, обеспечивающих полноту тестирования. Предлагаемые методы должны поддаваться алгоритмизации при достаточно слабых ограничениях на спецификацию (то есть ограничениях, обычно выполняемых в практических системах).

3. **Построение LTS-теории**, включающей определение LTS-модели с разрушением, трасс наблюдений LTS-модели, гипотезы о безопасности, безопасной конформности, а также включающей (алгоритмизуемые) методы генерации тестов. Кроме того, определяется композиция LTS-моделей, которую невозможно определить в теории трасс наблюдений.

4. **Разработка (алгоритмизуемых) методов пополнения спецификаций**, целью которого является переход к такой семантике взаимодействия, в которой наблюдаемы все отказы (для всех кнопок). При таком переходе должен сохраняться класс конформных реализаций и не сужаться класс безопасных реализаций. Этим, во-первых, достигается самоприменимость спецификации (она конформна сама себе), то есть рефлексивность безопасной конформности. При наличии ненаблюдаемых отказов конформность, вообще говоря, нерефлексивна; из-за этого спецификация не может рассматриваться как одна из конформных реализаций, что прямо противоречит интуиции разработчика. Во-вторых, пополнение спецификаций является первым шагом к решению проблемы монотонности.

5. **Построение теории верификации композиции** с целью решения проблемы монотонности (сохранения конформности при композиции и при асинхронном тестировании). С учётом решения проблемы пополнения спецификаций эту теорию достаточно построить только для семантик, в которых все отказы наблюдаемы. Проблема монотонности решается с помощью монотонного преобразования спецификации, то есть такого, что композиция преобразованных спецификаций компонентов оказывается корректной спецификацией системы (ей конформны композиции любых конформных реализаций компонентов). Такие преобразованные спецификации

названы монотонными. Требуется, чтобы монотонное преобразование сохраняло классы конформных и безопасных реализаций.

Кроме того, композиция монотонных спецификаций компонентов системы, рассматриваемая как реализация, должна быть конформна заданной спецификации системы. Тогда, по транзитивности конформности, любая корректная спецификация системы предъявляет к ней не больше требований, чем композиция монотонных спецификаций компонентов. Тем самым, задача верификации декомпозиции системных требований сводится к проверке того, что композиция монотонных спецификаций компонентов конформна заданной спецификации системы. В теории верификации должно быть не только доказано существование монотонных преобразований, но и построено такое преобразование, которое, во-первых, по возможности не слишком увеличивает размер спецификации и, во-вторых, поддаётся алгоритмизации. Кроме этого, монотонное преобразование должно быть применимо в асинхронном тестировании, когда преобразованию подлежит только реализация, но не фиксированная и известная среда взаимодействия.

Методологическая и теоретическая основа исследования

Для определения семантики взаимодействия и конформности в диссертации использовалось математическое моделирование тестового эксперимента, что позволило осуществить дедуктивный анализ математических моделей реализации, спецификации и теста. При построении и анализе этих математических моделей, при разработке методов генерации тестов и различных преобразованиях моделей, использовались элементы теории автоматов и формальных языков, теории графов и теории алгоритмов, а также формализм систем помеченных переходов (LTS). **Для построения теории верификации композиции применён специально разработанный автором метод моделирования с помощью вводимых в диссертации ϕ-трасс**, позволивший объединить в рамках одной теории трассовый подход,

достаточный для определения безопасности и конформности, и композицию моделей, которая обычно определяется только для LTS.

Научная новизна исследования

Новым в диссертационной работе являются:

1. **Метод формализации тестового эксперимента** с помощью параметризуемой машины тестирования, целью которого является определение семантики взаимодействия.
   a. Отличие от машин Милнера и Ван Глаббека заключается в параметризации машины набором «множественных» кнопок. Это является способом задания семантики взаимодействия, основанной только на наблюдениях (не на соответствии состояний) и моделирующей тестовые возможности по управлению и наблюдению. Впервые появляется возможность единообразно определять не только семантики, задаваемые машинами Милнера и Ван Глаббека, но и многие другие семантики, применяемые в теории и на практике, в частности, семантику отношения ***ioco***.
   b. Вводится новое понятие разрушения, означающее специальное действие, которое моделирует запрещённое поведение системы. В отличие от запрещённого состояния, разрушение, как запрещённое действие, может быть введено на уровне машины тестирования и в дальнейшем использовано в трассовой модели, а не только в модели LTS.
2. **Введение новых взаимосвязанных концепций** на уровне трассовой и LTS-моделей :
   a. Безопасное тестирование как тестирование, избегающее ненаблюдаемых отказов и разрушения и не продолжающее тестовый эксперимент при дивергенции. Это является хорошей альтернативой полному запрету на поведение таких типов, что позволяет расширить домен конформности.
   b. Гипотеза о безопасности – гипотеза о реализации, которая позволяет безопасно тестировать реализацию для заданной спецификации.

с. Безопасная конформность ***saco*** (*safe conformance*), которая опирается на гипотезу о безопасности, параметризуется семантикой взаимодействия и проверяема при безопасном тестировании. Многие частные конформности, основанные на семантиках наблюдаемого поведения, такие как отношение ***ioco***, являются частными случаями отношения ***saco***.

3. **Метод генерации полного набора тестов для безопасной конформности** по заданной спецификации, который является обобщением аналогичных методов для частных конформностей (в том числе, отношения ***ioco***) и алгоритмизуем при достаточно слабых ограничениях на спецификацию (то есть ограничениях, обычно выполняемых в практических системах). В отличие от аналогичных методов спецификация может быть задана не только в форме LTS, но и в форме модели наблюдаемых трасс. Кроме того, обеспечивается безопасное тестирование даже в тех случаях, когда в реализации есть ненаблюдаемые отказы, дивергенция и разрушение при условии, что реализация удовлетворяет гипотезе о безопасности.
4. **Метод пополнения спецификаций**, сохраняющего класс конформных и не сужающего класс безопасных реализаций, с использованием специальных фиктивных действий «не-отказов». Этот метод позволяет перейти к семантике, в которой все отказы наблюдаемы и которая является отношением предпорядка (в частности, рефлексивна). Метод поддаётся алгоритмизации при достаточно слабых ограничениях на спецификацию. Частное применение этого метода решает проблему пополнения для отношения ***ioco***.
5. **Введение ϕ-трасс** и соответствующей модели ϕ-трасс как промежуточного уровня абстракции между моделью наблюдаемых трасс, достаточной для определения конформности, но не достаточной для определения композиции моделей, и уровнем LTS-моделей, где такая композиция определяется. Эта модель позволяет в единой трассовой теории формализовать как конформность, так и композицию, что является ключевым моментом в проблеме монотонности. В частности, по ϕ-модели восстанавливается модель

наблюдаемых трасс, а композиция ϕ-моделей обладает важным свойством аддитивности: ϕ-модель, соответствующая композиции LTS-моделей, совпадает с композицией ϕ-моделей, соответствующих LTS-операндам.

6. **Метод монотонного преобразования спецификаций** для семантики, в которой все отказы наблюдаемы. Этот метод позволяет сохранить при преобразовании классы конформных и безопасных реализаций и применим в случае асинхронного тестирования. Метод поддаётся алгоритмизации при достаточно слабых ограничениях на спецификацию. Частное применение этого метода решает проблему монотонности для семантики отношения ***ioco***.

Практическая значимость работы

Хотя диссертационная работа носит теоретический характер, она имеет практическую ценность, прежде всего, как теоретическая основа существующих и разрабатываемых методов спецификации, генерации тестов и верификации декомпозиции. Параметризация конформности позволяет единообразно использовать одни и те же понятия и методы в зависимости от тех или иных тестовых возможностей по управлению тестируемой системой и наблюдению её поведения. Предлагаемая теория позволяет яснее понимать, какие слабые места имеются у практических тестовых инструментов и какого рода ошибки могут оставаться в программах после тестирования этими инструментами. Также она формулирует те ограничения, которым должны удовлетворять спецификации, чтобы по ним можно было генерировать полный набор тестов или компоновать спецификацию составной системы по спецификациям компонентов. Наконец, теория выявляет те гипотезы о реализации, которые обычно подразумеваются, но редко формулируются точно и недвусмысленно, что иногда приводит к недоразумениям при тестировании или верификации композиции.

В диссертационной работе важное значение придавалось поиску тех, по возможности, как можно более слабых ограничений, которые нужно налагать

на исходные спецификации, чтобы их можно было алгоритмически задавать, автоматически генерировать по ним полные наборы тестов и выполнять пополнение спецификаций и их монотонное преобразование, а также алгоритмически компоновать преобразованные спецификации.

Основные практические результаты работы – это предлагаемые методы генерации тестов и преобразования спецификаций:

1. Общий метод генерации тестов позволяет создавать тестовые системы по единой схеме для разных семантик тестового взаимодействия.
2. Появляется возможность единообразного асинхронного тестирования с разными средами взаимодействия на базе предлагаемого преобразования спецификации и последующей её композиции со средой.
3. Метод композиция преобразованных спецификаций компонентов составной системы является основой для создания технологии автоматической верификации декомпозиции системных требований. При этом важно, что спецификации компонентов и системы в целом могут пониматься в разных семантиках тестового взаимодействия.

Апробация результатов исследования

Теория конформности, предлагаемая в диссертационной работе, создавалась в результате осмысления и обобщения практического опыта тестирования на основе формальных моделей, которое проводилось в ИСП РАН, начиная с 1994 г. Многие основные идеи, понятия и методы, предложенные в работе, находили своё применение в создании и дальнейшем развитии тестовых систем KVEST (Kernel VErification and Specification Technology) и UniTESK (**Uni**fied **TE**sting & **S**pecification **T**ool**K**it). В первую очередь это относится к понятиям разрушения и безопасного тестирования, а также к преобразованиям спецификации при асинхронном тестировании с различными средами взаимодействия.

Результаты диссертационной работы были доложены на следующих Российских и международных научных конференциях и семинарах:

- Всероссийская научная конференция "Научный сервис в сети интернет: технологии распределенных вычислений", Абрау-Дюрсо, 2005;
- Семинар Института проблем информационной безопасности при МГУ им. М.В. Ломоносова, 2005 г.;
- Европейская объединённая конференция по теории и практике программного обеспечения ETAPS, семинар «Тестирование на основе моделей», Вена, Австрия, 2006 г.;
- Пятая общероссийская научная конференция «Математика и безопасность информационных технологий» (МаБИТ-06) в рамках Международной научной конференции по проблемам безопасности и противодействия терроризму, Москва, 2006 г.;
- Семинар кафедры математической кибернетики ф-та ВМиК МГУ им. М.В. Ломоносова, Москва, 2006г.;
- Семинар «Тестирование дискретных управляющих систем на основе автоматных моделей», Томский Государственный Университет, 2006 г.;
- Научный семинар «Проблемы современных информационно-вычислительных систем», мех.-мат. фак-т МГУ им. М.В. Ломоносова, Москва, 2006 и 2007 г.г.

По тематике диссертационной работы автор участвовал в выполнении следующих грантов РФФИ:

- 96-01-01277-а «Методы тестирования программного обеспечения на основе формальных спецификаций» (1996-1998);
- 99-01-00207-а «Формальные методы в форвард- и реверс-инженерии программных систем» (1999-2001);
- 02-01-00959-а «Повторное использование формальных спецификаций в производстве и тестировании программных систем» (2002-2004);

- 04-07-90308-в «Верификация функций безопасности и мобильности протоколов IP» (2004-2006);
- 05-01-00999-а «Методы формальных спецификаций в тестировании Интернет–приложений» (2005-2007);
- 07-07-00243-а «Верификация функций безопасности протокола нового поколения IPsec v2» (2007-2009).

Содержание диссертационной работы легло в основу разработанного совместно с А.С. Косачевым учебного курса «Теория верификации соответствия программ», который читался на механико-математическом ф-те МГУ им. М.В. Ломоносова весной 2006 г. Отдельные элементы теории используются в учебном курсе «Тестирование на основе моделей», который читается на ф-те ВМиК МГУ им. М.В. Ломоносова, начиная с 2007 г. и в курсе «Методы формальной спецификации программ», который читался на 3-ем потоке ф-та ВМиК МГУ им. М.В. Ломоносова в 1995-2007 г.г.

Публикации

По теме диссертационной работы опубликовано 27 работ, полностью отражающих основные научные результаты работы, в том числе 7 в рецензируемых научных изданиях по списку ВАК 2007 года.

Структура и объём работы

Диссертационная работа состоит из введения, шести глав, заключения, списка литературы и приложения, содержащего доказательства утверждений, встречающихся в тексте. Каждая глава разбита на разделы, некоторые разделы разбиты на подразделы; каждый уровень имеет независимую нумерацию внутри единицы вышестоящего уровня. 135 определений, 84 леммы и 49 теорем имеют раздельную сквозную нумерацию по всему тексту. Содержание работы изложено на 436 страницах (без приложения), список библиографических ссылок включает 154 наименования (для удобства пользования сначала

перечисляются работы автора, затем другие работы на русском языке и, наконец, работы на других языках). В тексте содержится 87 рисунков.

Содержание работы

<u>В главе 1</u> вводятся встречающиеся в тексте общематематические понятия и обозначения. Такими понятиями являются классы, множества, числа (целые), соответствия, последовательности, деревья последовательностей и порождающие графы.

Автор опирается на NBG (Нейман, Бернайс, Гедель) аксиоматическую теорию множеств с праэлементами. Эта теория основана на понятиях класса и праэлемента. Обычные теоретико-множественные операции и отношения (конструктор, объединение, пересечение, вложенность, декартовое произведение) определяются для класса, который считается состоящим из элементов. Такими элементами могут быть праэлементы и некоторые классы, называемые множествами. Праэлемент – это объект, который является элементом класса, но сам классом не является. Множество – это класс, который является как элементом другого класса, так и сам состоит из элементов (множеств и праэлементов). Собственный класс – это класс, который не является множеством, то есть не является элементом какого-либо класса, хотя сам состоит из элементов (множеств и праэлементов).

Под деревом последовательностей понимается множество последовательностей, замкнутое по операции взятия префикса последовательности. Под порождающим графом понимается граф с выделенными начальными и конечными вершинами, дуги которого либо непомечены, либо помечены символами из некоторого алфавита. Граф порождает множество последовательностей в этом алфавите, которые являются последовательностями помеченных дуг маршрутов графа.

Глава 2 посвящена формализации тестового эксперимента с целью определения семантики взаимодействия теста и реализации, которая строится на наблюдениях внешних действий и отказов. Рассматриваются различные тестовые возможности, задаваемые с помощью предлагаемой автором машины тестирования. Отличие от реактивной машины Милнера и генеративной машины Ван Глаббека заключается в том, что вместо нажатия одной или нескольких кнопок, каждая из которых соответствует одному разрешаемому действию, нажимается одна кнопка, которой соответствует множество таких действий. Кроме того, для каждой кнопки указывается, соответствует ли её «кнопочному» множеству наблюдаемый отказ ($\mathfrak{R}$-кнопки) или нет ($\mathfrak{Q}$-кнопки). Набор семейств подмножеств алфавита внешних действий $\mathfrak{R},\mathfrak{Q}\subseteq\mathcal{P}(\mathsf{L})$, покрывающих весь алфавит $(\cup\mathfrak{R})\cup(\cup\mathfrak{Q})=\mathsf{L}$, как раз и является параметром машины. Такая машина названа $\mathfrak{R}/\mathfrak{Q}$-машиной, а соответствующая ей семантика взаимодействия – $\mathfrak{R}/\mathfrak{Q}$-семантикой. Параметризация семействами наблюдаемых и ненаблюдаемых отказов позволяет учитывать те или иные практические ограничения на (правильное) взаимодействие. Многие известные семантики тестирования оказываются вариантами $\mathfrak{R}/\mathfrak{Q}$-семантики при соответствующем определении семейств $\mathfrak{R}$ и $\mathfrak{Q}$.

Например, семантика отношения ***ioco*** для реактивной системы предполагает, что при тестировании можно посылать в реализацию один стимул, а принимать все реакции, отсутствие реакций (стационарность, традиционно обозначаемая символом δ) наблюдаемо, а блокировка стимула не наблюдаема. Такую семантику автор предлагает называть δ-семантикой, она совпадает с $\mathfrak{R}/\mathfrak{Q}$-семантикой, где $\mathfrak{R}$ состоит из одного множества всех реакций, а $\mathfrak{Q}$ это множество блокировок стимулов (блокировка стимула – это множество, состоящее только из этого стимула).

Рассматриваются различные «ответвления» от ***ioco***-семантики: наблюдаемость блокировок стимулов, посылка нескольких стимулов (отказ состоит из нескольких стимулов), приём части реакций (отказ состоит не из всех, а из части реакций) и совмещение посылки стимула с приёмом реакций (отказ содержит стимул и реакции). Здесь детально изучаются примеры, рассмотренные выше в подразделе «Выбор семантики взаимодействия и генерация тестов» раздела «Проблематика».

Вводится новое понятие разрушения как запрещённого действия, которое может быть определено в модели, но при правильном взаимодействии не должно быть достижимо. Дивергенция моделируется Δ-действием, которое возможно только в конце правильного взаимодействия (без продолжения).

Предлагаются основанные на такой семантике концепция безопасного тестирования, реализационная гипотеза о безопасности и безопасная конформность (***saco*** – safe conformance) типа редукции (сводимости реализации к спецификации).

Кратко рассмотрены проблемы несохранения конформности при асинхронном тестировании и композиции.

Формальное рассмотрение этих концепций представлено вдальнейших главах.

Также во второй главе рассматриваются другие тестовые возможности, формализуемые в машине тестирования: приоритеты, репликация, глобальное тестирование, бесконечные и отрицательные наблюдения, трассы готовности. В диссертационной работе, как в других работах по тестированию конформности, предполагается, что в реализации отсутствуют приоритеты между действиями: если действие разрешено и определено в реализации, оно может выполняться независимо от других разрешённых и определённых действий. Репликация означает «копирование» машины с независимым наблюдением её поведения в разных копиях. Если репликация допустима в любой момент времени, это даёт возможность отслеживать состояния реализации (с точностью до их эквивалентности, как множество дальнейших поведений) и определять

конформности типа симуляций. В диссертационной работе такая возможность не рассматривается, и предполагается, что репликация возможна только перед тестированием, то есть моделируется рестарт системы для нескольких прогонов одного и того же или разных тестов. Глобальное тестирование означает теоретическую возможность получить наблюдение любого возможного поведения реализации, что необходимо для полноты тестирования (то есть обнаружения всех ошибок). Бесконечные наблюдения не рассматриваются, поскольку они практически не реализуемы и избыточны для конформностей, основанных на конечных трассах наблюдений (тесты заканчиваются через конечное время). Отрицательное наблюдение «вычисляется» по множеству всех наблюдений. Например, если после трассы ни разу не наблюдалось действие из отказа P, то трасса продолжается отказом P. Однако для конформности типа редукции это ничего нового не даёт. Поэтому отрицательные наблюдения в работе не рассматриваются.

В этой главе также расматриваются трассы готовности, основанные на наблюдении множества всех действий, которые реализация могла бы выполнить в данный момент времени, если бы они были разрешены. Такая тестовая возможность встречается на практике гораздо реже, чем наблюдение действий и отказов, и поэтому в дальнейших главах работы не рассматривается. В то же время отмечается, что при наличии такой возможности трассы готовности полезны, и можно определять соответствующие гипотезы о безопасности и безопасную конформность для трасс готовности. Эти трассы похожи на ϕ-трассы, которые вводятся в главе 6 и являются основным инструментом в решении проблемы монотонности.

Глава 3 содержит теорию трасс наблюдений. Определяется модель наблюдаемых трасс ($\mathfrak{R}$-трасс) для заданной $\mathfrak{R}/\mathfrak{Q}$-семантики взаимодействия, которые состоят из действий и $\mathfrak{R}$-отказов и могут заканчиваться дивергенцией или разрушением. Даже без учёта дивергенции и разрушения, такое

определение впервые даётся интенсионально, а не генетически (через LTS), что позволяет строить теорию трассовых моделей безотносительно к теории LTS-моделей. Допущение дивергенции позволяет, в частности, решить проблему, характерную для отношений типа ***ioco***, которая заключается в выходе за пределы домена конформности при композиции (появление дивергенции как результата бесконечного взаимодействия компонентов).

В диссертации внешние действия, разрушение и дивергенция считаются праэлементами, что позволяет не путать их в трассах с отказами как множествами действий.

Вводится полная трассовая модель (***F***-модель), соответствующая семантике, в которой любому подмножеству внешних действий соответствует своя $\mathfrak{R}$-кнопка. Доказывается, что проекция полной модели на $\mathfrak{R}$-трассы является $\mathfrak{R}$-моделью, и любая $\mathfrak{R}$-модель является проекцией полной модели. Также показывается, что объединение ***F***-моделей является ***F***-моделью, а пересечение ***F***-моделей не является, вообще говоря, ***F***-моделью. Доказывается утверждение о разложении ***F***-модели на, так называемые, стабильные и контрстабильные деревья, что соответствует состояниям LTS и в следующей главе используется для установления эквивалентности трассовой и LTS-теорий с точки зрения конформности.

В этой главе показано, что трассовая теория достаточна для формулировки отношения безопасности и безопасной конформности. Определяются отношения безопасности в реализации ***safe in*** и в спецификации ***safe by***, означающие «кнопка безопасна после трассы». На этой основе определяются безопасные трассы реализации $\boldsymbol{SafeIn}(\mathrm{I})$ и спецификации $\boldsymbol{SafeBy}(\Sigma)$.

Гипотеза о безопасности – это отношение ***safe for*** – «реализация безопасно-тестируема для спецификации»: 1) если спецификация не разрушается с самого начала, то и реализация не разрушается с самого начала; 2) после трассы, которая безопасна как в реализации, так и в спецификации, кнопка, безопасная в спецификации, должна быть безопасна в реализации.

I ***safe for*** Σ $=_{\text{def}}$ $(\langle\gamma\rangle\notin\Sigma \Rightarrow \langle\gamma\rangle\notin\mathrm{I})$ & $\forall\sigma\in\boldsymbol{SafeBy}(\Sigma)\cap\boldsymbol{SafeIn}(\mathrm{I})$

$\forall \mathrm{P}\in\mathfrak{R}\cup\mathfrak{Q}$ (P ***safe by*** Σ ***after*** $\sigma \Rightarrow$ P ***safe in*** I ***after*** σ).

Безопасная конформность ***saco*** – «реализация безопасно-конформна спецификации» – применима только к безопасным реализациям и означает: любое наблюдение, которое возможно над реализацией, возможно в спецификации в той же самой безопасной ситуации, то есть после общей безопасной трассы при нажатии кнопки, безопасной в спецификации.

I ***saco*** Σ $=_{\text{def}}$ I ***safe for*** Σ & $\forall\sigma\in\boldsymbol{SafeBy}(\Sigma)\cap\boldsymbol{SafeIn}(\mathrm{I})$

$\forall \mathrm{P}$ ***safe by*** Σ ***after*** σ $\boldsymbol{obs}(\sigma,\mathrm{P},\mathrm{I})\subseteq\boldsymbol{obs}(\sigma,\mathrm{P},\Sigma)$,

где ***obs*** означает множество возможных наблюдений, когда после трассы σ нажимается кнопка P.

Для конформности ***saco*** определяется понятие трассового теста и выясняются условия конечности времени выполнения теста. Также введено понятие управляемого (без лишнего недетерминизма) теста. Далее на трассовом уровне определено отношение «реализация проходит тест» и полные наборы тестов. Выделен подкласс строгих тестов (найденная ошибка всегда фиксируется) и доказано утверждение о полном наборе строгих управляемых тестов. Для примитивных тестов, каждый из которых строится ровно по одной безопасной трассе спецификации, этот набор содержит только ограниченные (по времени выполнения) управляемые строгие тесты. После этого рассматриваются вопросы оптимизации тестов и набора тестов, которые могут использоваться на практике. Глава завершается кратким рассмотрением вопроса об алгоритмическом задании трассовой модели и алгоритмической генерации полного тестового набора. Метод генерации полного набора тестов для безопасной конформности по заданной спецификации является обобщением аналогичных методов для частных конформностей (в том числе, отношения ***ioco***). Отличие, во-первых, в том, что спецификация может быть задана в виде модели наблюдаемых трасс, а не в виде LTS, и, во-вторых, в том, что обеспечивается безопасное тестирование, даже если в реализации есть

ненаблюдаемые отказы, дивергенция и разрушение, при условии, что реализация удовлетворяет гипотезе о безопасности.

В главе 4 в качестве модели рассматривается система помеченных переходов (LTS). Новым является введение перехода по разрушению. Трассы LTS – это трассы её маршрутов, причём при проходе стабильного состояния в трассу может добавляться любая конечная последовательность $\mathfrak{R}$-отказов, соответствующих состоянию (отказ как множество действий, по которым нет переходов из состояния). Дивергенция как бесконечная цепочка τ-переходов, которой продолжается маршрут, допускается в LTS и моделируется Δ-действием в конце трассы маршрута. Устанавливается эквивалентность трассовой и LTS-моделей в том смысле, что множество трасс наблюдений, задаваемых LTS, является моделью наблюдаемых трасс, и каждая модель наблюдаемых трасс может быть задана некоторой LTS. Это позволяет развить LTS-теорию, опираясь на трассовую теорию: определить гипотезу о безопасности и безопасную конформность.

Для LTS определяется операция объединения множества LTS с помощью добавления нового начального состояния и проведения из него τ-переходов в начальные состояния объединяемых LTS. Доказано, что множество трасс такого объединения LTS совпадает с объединением множеств трасс этих LTS. В то же время в диссертации отмечается, что класс всех LTS, подкласс LTS в данном алфавите (и многие другие подклассы LTS) не являются множествами, поскольку не налагается никаких ограничений на природу состояний LTS. Такие собственные классы LTS нельзя объединять, поскольку в объединении будет получаться класс состояний, не являющийся множеством.

С точки зрения безопасности, конформности и генерации тестов LTS-модель более «избыточна», чем трассовая модель. Однако она имеет ряд преимуществ. LTS-модель более наглядна и удобна для человеческого восприятия. Многие вещи можно увидеть в ней, так сказать, непосредственным

созерцанием (когда пишут, что это «очевидно»). Трассовая модель не обладает такой наглядностью. В частности, генетическое определение $\mathfrak{R}$-трасс через LTS просто и понятно, а интенсиональное определение $\mathfrak{R}$-модели более сложно и далеко не очевидно. LTS более естественно «укладывается» в чёрный ящик машины тестирования, чем трассовая модель.

Но самым главным преимуществом LTS-модели является возможность определения композиции составной модели из моделей-компонентов, используя оператор параллельной композиции алгебры процессов (в данной работе оператор $\wr$ аналогичный оператору композиции в CCS – (Calculus of Communicating Systems). Соответственно, взаимодействие в LTS-теории выглядит как композиция LTS-реализации и LTS-окружения, а тестирование – как композиция LTS-реализации и LTS-теста. С этой точки зрения $\mathfrak{R}/\mathfrak{Q}$-машина тестирования интерпретируется как ограничение на LTS-окружение/тест, то есть на способ (правильного) взаимодействия с LTS-реализацией.

Для того, чтобы метод генерации трассовых тестов можно было использовать для генерации LTS-тестов, во-первых, показывается соответствие отношений «реализация проходит тест» в трассовой и LTS-теориях, и, во-вторых, определяется преобразование трассового теста в LTS-тест.

Таким образом, конформность и композиция определяются, фактически, на разных уровнях абстракции. Это становится причиной ряда проблем, главными из которых являются проблема рефлексивности спецификации и монотонности (сохранения конформности при композиции). Эти проблемы рассматриваются в следующих главах.

<u>Глава 5</u> посвящена решению проблемы пополнения спецификации. Задачей такого пополнения является достижение рефлексивности конформности, что, в свою очередь, является первым шагом к решению проблемы монотонности (сохранения конформности при композиции).

Сначала проблема пополнения рассматривается для частного случая δ-семантики и отношения ***ioco***, отличающегося от отношения ***ioco***$_\delta$ (***saco*** в δ-семантике) только более узким доменом реализаций и спецификаций. Показано, что отношение ***ioco***, вообще говоря, нерефлексивно, транзитивно и немонотонно. В частности, показывается на примерах несохранение ***ioco***-конформности при асинхронном тестировании. Также для δ-семантики существует проблема выхода за пределы домена отношения ***ioco*** (и ***ioco***$_\delta$) при композиции (возникновение дивергенции как бесконечной цепочки синхронных переходов). Эта проблема решается в предлагаемом отношении ***ioco***$_{\gamma\delta}$ (***saco*** в γδ-семантике, то есть δ-семантике с разрушением).

Рассмотрены различные виды допущений полноты, используемые в литературе по тестированию конформности, и основанные на пополнении состояний: доопределении отсутствующего в состоянии перехода по стимулу с тем или иным дальнейшим поведением. Предлагается новое гамма-пополнение: после приёма неспецифицированного стимула определяется разрушение. Показано, что отношение ***ioco***, вообще говоря, не сохраняется при пополнении состояний LTS-модели. При этом только два вида пополнения состояний не усиливают конформность (не появляется дополнительных требований к реализации): гамма-пополнение, предлагаемое автором диссертации, и предлагаемое Яном Тритмансом, так называемое, демоническое пополнение (после неспецифицированного стимула допустимо любое поведение, кроме блокировок, дивергенции и разрушения). При этом гамма-пополнение удобнее тем, что сохраняется информация о стимулах, которыми не продолжались трассы: теперь эти стимулы принимаются с дальнейшим разрушением, в то время как при демоническом пополнении эта информация теряется. Тем самым, при гамма-пополнении мы избегаем лишнего тестирования (как в синхронном,

так и в асинхронном случае), которое неизбежно возникает при демоническом пополнении. В целом делается вывод о необходимости трассового гамма-пополнения, когда трасса, не продолжавшаяся стимулом, продолжается им и далее разрушением.

Далее исследован вопрос о рефлексивности и транзитивности конформности, и определяется необходимое и достаточное условие рефлексивности и достаточное условие транзитивности. Показано, что при отсутствии $\mathfrak{Q}$-кнопок (ненаблюдаемых отказов) эти условия выполнены и в $\mathfrak{R}/\varnothing$-семантике отношение ***saco*** является предпорядком.

Наконец, доказывается существование пополнения, сохраняющего класс конформных реализаций и такого, что для пополненной спецификации классы конформных реализаций в $\mathfrak{R}/\mathfrak{Q}$- и $\mathfrak{R}\cup\mathfrak{Q}/\varnothing$-семантиках совпадают. Это пополнение строится в трассовой теории как объединение конформных трассовых реализаций. В LTS-теории такое пополнение определяется как LTS, множество трасс которой совпадает с объединением множеств трасс конформных LTS.

В диссертации отмечается, что LTS-пополнение нельзя определять как объединение конформных LTS, поскольку класс таких LTS не является, вообще говоря, множеством. Если бы этот класс считался множеством, это привело бы к своеобразному варианту известного парадокса наивной теории множеств. Дело в том, что тогда объединение конформных LTS было бы собственным операндом, и, поскольку из его начального состояния ведут τ-переходы в начальные состояния всех объединяемых LTS, это привело бы к бесконечной цепочке внутренних переходов, то есть дивергенции. А наличие дивергенции в самом начале неконформно для большинства спецификаций, что противоречит совпадению множества трасс объединения конформных LTS с объединением множеств трасс этих LTS.[2]

[2] С той же целью обхода этих парадоксов иногда вместо «система» говорят «пространство» (labeled transition space), а для задания отдельной LTS указывают её начальное состояние как

После этого изучается проблема сохранения класса безопасных реализаций. Показано, что этот класс может как расширяться (что допустимо и даже полезно), так и сужаться (что плохо) при пополнении спецификации. Исследованы причины сужения класса безопасных реализаций и предложено решение проблемы, основанное на введении фиктивных действий – не-отказов. С помощью не-отказов удаётся сделать такое пополнение, которое сохраняет класс конформных и не сужает класс безопасных реализаций. Кратко рассмотрены вопросы алгоритмизации этого пополнения и его частные случаи для δ-, γδ- и βγδ-семантик (последняя допускает блокировки стимулов).

В <u>главе 6</u> решается проблема монотонности конформности: сохранение конформности при композиции и при асинхронном тестировании. Суть этой проблемы в том, что композиция реализаций компонентов, конформных своим спецификациям, может оказаться неконформной композиции спецификаций компонентов, если последнюю делать по тем же правилам, по которым делается композиция реализаций компонентов (прямая композиция):

$\mathbf{I_1}$ ***saco*** $\mathbf{S_1}$ & $\mathbf{I_2}$ ***saco*** $\mathbf{S_2}$ $\Rightarrow$ $\mathbf{I_1 \between I_2}$ ***saco*** $\mathbf{S_1 \between S_2}$.

Поскольку с помощью пополнения спецификации можно перейти к семантике без Ω-кнопок с сохранением классов безопасных и конформных реализаций, в данной главе рассматриваются только семантики без Ω-кнопок, когда конформность ***saco*** является предпорядком.

Для решения проблемы монотонности создана общая теория монотонности, в рамках которой вводятся следующие понятия. Корректная спецификация системы – это такая спецификация, которая удовлетворяет

одно из состояний LTS-пространства. Однако в этом случае возникают трудности при определении различных преобразований LTS. В частности, если строится LTS, состояния которой – все подмножества состояний исходной LTS, то возрастает мощность множества состояний LTS, и, в конце концов, она перестаёт «помещаться» в LTS-пространстве.

условию монотонности: прямая композиция компонентов, конформных своим спецификациям, конформна спецификации системы:

$\mathbf{I_1}$ ***saco*** $\mathbf{S_1}$ & $\mathbf{I_2}$ ***saco*** $\mathbf{S_2}$ $\Rightarrow$ $\mathbf{I_1} \between \mathbf{I_2}$ ***saco*** $\mathbf{S}$.

Далее определяется специальная – косая – композиция спецификаций компонентов $\mathbf{S_1} /\!/ \mathbf{S_2}$ как такая корректная спецификация системы, которая конформна любой корректной спецификации. Решение проблемы монотонности предложено искать как монотонное преобразование $\mathcal{T}$, обладающее тем свойством, что косая композиция спецификаций совпадает с прямой композицией монотонных (монотонно преобразованных) спецификаций: $\mathbf{S_1} /\!/ \mathbf{S_2} = \mathcal{T}(\mathbf{S_1}) \between \mathcal{T}(\mathbf{S_2})$.

Общая теория монотонности базируется на понятии ϕ-трасс, занимающем промежуточный уровень абстракции между моделями трасс наблюдений в $\mathfrak{R}$-семантике и LTS-моделями. В частности, для ϕ-трасс определяется оператор композиции. Сформулированы восемь достаточных условий монотонности. Шесть из них не связаны с самим монотонным преобразованием, и описывают связь между конформностью ***saco***, трассами наблюдений ($\mathfrak{R}$-трассами), ϕ-трассами и оператором прямой композиции $\between$. Два условия налагаются на преобразование для того, чтобы оно было монотонным.

Для $\mathfrak{R}$-трасс определяется отношение мажорирования и доказывается его эквивалентность конформности (первое условие монотонности).

ϕ-трассы аналогичны $\mathfrak{R}$-трассам, но вместо $\mathfrak{R}$-отказа в стабильном состоянии s в трассу может быть добавлена полная информация о состоянии в виде ϕ-символа состояния как пары (***ref***-множество, ***gamma***-множество), где ***ref***-множество состоит из действий, по которым нет переходов из s, а ***gamma***-множество – из действий, после перехода по которым есть переход по разрушению. Поскольку в диссертации внешние действия, разрушение и

дивергенция считаются праэлементами, в ϕ-трассах они отличимы от ϕ-символов как элементов декартового произведения.

Рассматриваются не только конечные, но и бесконечные ϕ-трассы. Для ϕ-трасс определяется оператор их композиции, дающий множество ϕ-трасс.

Для ϕ-трасс доказаны два условия монотонности. 1) Генеративность: по ϕ-трассам модели однозначно восстанавливаются её $\mathfrak{R}$-трассы. 2) Аддитивность: множество ϕ-трасс композиции LTS совпадает с объединением композиций ϕ-трасс LTS-операндов $\mathbf{\Phi}^{\omega}(\mathbf{S_1}\!\upharpoonright\!\downharpoonright\!\mathbf{S_2})=\cup(\mathbf{\Phi}^{\omega}(\mathbf{S_1})\!\upharpoonright\!\downharpoonright\!\mathbf{\Phi}^{\omega}(\mathbf{S_2}))$, где оператор $\mathbf{\Phi}^{\omega}$ даёт множество ϕ-трасс LTS. Первое свойство даёт возможность учитывать конформность в теории ϕ-трасс. Второе свойство позволяет рассматривать композицию, не выходя за рамки теории ϕ-трасс. Таким образом, теория ϕ-трасс оказывается замкнутой как в смысле конформности, так и в смысле композиции, что и является ключом к успеху в решении проблемы монотонности, связывающей эти два понятия: конформность и композиция.

Далее рассматриваются два монотонных преобразования. Первое преобразование определяется как множество $\boldsymbol{Conf}(\mathbf{S})$ конформных ϕ-трасс, то есть объединение ϕ-трасс всех реализаций, конформных спецификации $\mathbf{S}$. По сути, здесь доказывается существование монотонного преобразования в самом общем случае. Однако такое преобразование даёт самую большую монотонную спецификацию из всех возможных, и оно, вообще говоря, не алгоритмизуемо.

Остальная часть главы посвящена поиску меньших по объёму монотонных подмоделей. Для этого вводится отношение “$\preccurlyeq$” мажорирования ϕ-трасс, отвечающее двум важным условиям. 1) Генеративность: мажорирование ϕ-трасс влечёт мажорирование генерируемых ими $\mathfrak{R}$-трасс. 2) Композиционность: мажорирование ϕ-трасс операндов влечёт мажорирование ϕ-трасс композиции, то есть $\mathbf{\Phi}^{\omega}(\mathbf{I_1})\preccurlyeq\mathbf{\Phi}^{\omega}(\mathbf{S_1})$ и $\mathbf{\Phi}^{\omega}(\mathbf{I_2})\preccurlyeq\mathbf{\Phi}^{\omega}(\mathbf{S_2})$ влечёт

$\cup(\mathbf{\Phi}^{\omega}(\mathbf{I_1})\!\upharpoonright\!\mathbf{\Phi}^{\omega}(\mathbf{I_2}))\preccurlyeq\cup(\mathbf{\Phi}^{\omega}(\mathbf{S_1})\!\upharpoonright\!\mathbf{\Phi}^{\omega}(\mathbf{S_2}))$. Также показано, что мажорирование ϕ-трасс является предпорядком.

Такое отношение мажорирования позволяет удалить из наибольшей монотонной спецификации «лишние» ϕ-трассы, если они мажорируются оставшимися ϕ-трассами. К сожалению, далеко не всегда можно получить наименьшую (и даже минимальную) по вложенности монотонную спецификацию. Предложены три вложенные монотонные подмодели: $\boldsymbol{Conf}(\mathbf{S}) \supseteq$ финальная $\supseteq$ однородная $\supseteq$ сингулярная.

Сингулярная монотонная подмодель $\mathcal{T}(\mathbf{S})$ наибольшей монотонной спецификации $\boldsymbol{Conf}(\mathbf{S})$ не всегда существует. Для её существования налагаются ограничения на ветвимость исходной LTS-спецификации $\mathbf{S}$. Спецификации, удовлетворяющие этим ограничениям, названы конечно-доминируемыми. Частично эти ограничения совпадают с тестовыми ограничениями, то есть ограничениями, требуемыми для алгоритмической генерации тестов по исходной спецификации $\mathbf{S}$. Дополнительные ограничения совпадают с теми, которые нужны для того, чтобы преобразованная сингулярная модель $\mathcal{T}(\mathbf{S})$ также удовлетворяла тестовым ограничениям, то есть чтобы по ней также можно было алгоритмически генерировать тесты. Таким образом, эти ограничения «не лишние». Метод монотонного преобразования спецификаций поддаётся алгоритмизации при достаточно слабых ограничениях на спецификацию (то есть ограничениях, обычно выполняемых в практических системах). Частное применение этого метода решает проблему монотонности для отношения ***ioco***.

После определения монотонного преобразования изучается проблема композиции преобразованных спецификаций. Суть её в том, что в общем случае композиция спецификаций, удовлетворяющих всем указанным ограничениям, может не удовлетворять тестовым ограничениям. В этом случае спецификация композиционной системы не годится для алгоритмической генерации системных тестов. Для решения этой проблемы определяются

дополнительные ограничения на преобразованные, а затем и исходные спецификации компонентов системы. Также исследована проблема сохранения безопасности при композиции. Общий вывод: без дополнительных гипотез о реализациях безопасное системное тестирование либо невозможно, либо сводится к автономному тестированию отдельных компонентов.

Подводя итоги, можно сказать, что косая композиция спецификаций компонентов составной системы, выполняемая как прямая композиция монотонно преобразованных спецификаций компонентов, позволяет решить две задачи: 1) верификацировать согласованность имеющейся спецификации системы со спецификациями компонентов, и 2) при отсутствии спецификации системы сгенерировать такую спецификацию по спецификациям компонентов.

Первая задача – это классическая задача верификации декомпозиции системных требований, то есть проверка правильности декомпозиции требований к системе в требования к компонентам системы. Для её решения строится косая композиция системы и проверяется её конформность заданной спецификации системы. Такую проверку можно делать аналитически, в том числе, моделируя тестирование косой композиции спецификаций компонентов, рассматриваемой как реализация системы, по полному набору тестов, сгенерированному из имеющейся спецификации системы.

Решение второй задачи позволяет достигнуть три цели. 1) Построенная спецификация системы может рассматриваться как описание системы с точки зрения её пользователей. 2) Такую спецификацию можно считать техническим заданием для разработчика системы, в том числе, при возможных дальнейших модификациях системы (её компонентов) с сохранением внешней функциональности и схемы компоновки. 3) По спецификации системы можно генерировать системные тесты при дополнительных гипотезах о реализациях.

Важно также отметить, что косая композиция не зависит от того, в какой семантике рассматривается составная система. Иными словами, она «выполняет свою работу» для любой семантики. Также важно, что для системы, состоящей не из двух, а большего числа компонентов, монотонное

преобразование требуется только для спецификаций исходных, атомарных компонентов, а не промежуточных подсистем, составленных из таких компонентов. Так происходит в том случае, когда не меняется схема компоновки (число компонентов и порядок применения оператора композиции), то есть когда реализации и спецификации компонентов компонуются по одной схеме. В противном случае метод работает, если выполнять монотонное преобразование промежуточных результатов композиции.

Заключение

В настоящее время повышение качества, безопасности и надёжности ПО, как вновь создаваемого, так и давно находящегося в эксплуатации, стало важнейшей проблемой как для разработчиков реальных программных систем в индустрии, так и для исследователей в области информатики. Ключевой частью решения этой проблемы является тестирование программ, доля которого во времени и других ресурсах от общих расходов на разработку ПО составляет около 50%, а для критических по безопасности приложений – до 90%. Противоречивая задача повышения качества программ и снижения затрат на тестирование диктует необходимость поиска путей автоматизации тестирования. Одним из важнейших методов решения этой задачи является тестирование на основе формальных (математических) моделей, целью которого является проверка того, что реализационная модель (модель тестируемой системы) соответствует (конформна) спецификации (модель требований).

В диссертационной работе предлагается теория конформности для широкого класса семантик взаимодействия теста с реализацией, основанных только на наблюдаемом поведении, с параметризацией конформности типа редукции той или иной семантикой из этого класса. **Область применения расширяется допущением в тестируемых реализациях ненаблюдаемых отказов, дивергенции и разрушения,**

моделирующего запрещённое поведение, в рамках, определяемых спецификацией. Это важно для моделирования предусловий вызова программ, для построения безопасных систем из небезопасных компонентов, для учёта дивергенции, возникающей при композиции и т.п. **Для практического применения теория развита до уровня алгоритмов генерации тестов;** показано, при каких ограничениях на спецификацию такие алгоритмы можно построить. И наконец, **в предлагаемой теории решается проблема монотонности (сохранение конформности при композиции и при асинхронном тестировании)** для рассматриваемого класса конформностей. Для этого предлагается монотонное преобразование спецификации, которое, при определённых ограничениях, может быть реализовано алгоритмически.

Получены следующие основные результаты:

1. **Метод формализации тестового эксперимента** с помощью параметризуемой машины тестирования. Целью формализации является определение класса семантик взаимодействия, основанных только на наблюдениях (не на соответствии состояний) и моделирующих те или иные тестовые возможности по управлению и наблюдению поведения реализации. Также вводится понятие *разрушения*, моделирующее поведение системы, запрещённое при тестировании.
2. **Введение на уровне трассовой и LTS-моделей концепции безопасного тестирования**, избегающего разрушения и ненаблюдаемых отказов и не продолжающего тестовый эксперимент в том случае, когда возможна дивергенция. Введение гипотезы о безопасности реализации, которая позволяет её безопасно тестировать для заданной спецификации, и безопасной конформности типа редукции, которая опирается на гипотезу о безопасности и параметризуется семантикой взаимодействия.
3. **Метод генерации полного набора тестов для безопасной конформности** по спецификации, заданной в виде модели наблюдаемых трасс или LTS-модели, обобщающий аналогичные методы для некоторых частных семантик на произвольную семантику взаимодействия, основанную на

наблюдаемом поведении. Метод алгоритмизуем при достаточно слабых ограничениях на спецификации (то есть ограничениях, обычно выполняемых в практических системах).

4. **Метод пополнения спецификаций** с использованием специальных фиктивных действий «не-отказов», который позволяет сохранять класс конформных и не сужать класс безопасных реализаций и поддаётся алгоритмизации при достаточно слабых ограничениях на спецификацию. В частности, решается проблема пополнения для семантики отношения ***ioco***.
5. **Разработка концепции ϕ-трасс** и модели ϕ-трасс, которые позволяют в единой трассовой теории формализовать как конформность, так и композицию, что является ключевым моментом в проблеме монотонности.
6. **Метод монотонного преобразования спецификации**, который при достаточно слабых ограничениях на исходную спецификацию строит монотонную спецификацию. При этом сохраняются классы безопасных и конформных реализаций, а для преобразованных спецификаций конформность сохраняется при композиции. Метод поддаётся алгоритмизации при достаточно слабых ограничениях на спецификацию. Частное применение метода решает проблему монотонности для семантики отношения ***ioco***.

Теория конформности даёт общую методологию уточнения и формализации процесса создания тестовой системы для того или иного конкретного случая. Общие алгоритмы генерации тестов и преобразования спецификаций, определяемые теорией, могут служить базовым шаблоном для разработки эффективных частных алгоритмов и реализующих их тестовых программ в практически значимых областях.

В практическом плане результаты диссертационной работы сводятся к предлагаемым методам генерации тестов и преобразования спецификаций. Это даёт, во-первых, возможность создавать тестовые системы для разных семантик тестового взаимодействия на базе единого подхода с использованием общего метода генерации тестов. Во-вторых, в сочетании с

преобразованием спецификации метод применим для асинхронного тестирования с разными средами взаимодействия. В-третьих, заложена основа для построения инструментов автоматической верификации декомпозиции системных требований, причём спецификации компонентов и системы в целом могут пониматься в разных семантиках тестового взаимодействия.

В приведённой ниже таблице показаны методы тестирования, которые применялись в системах тестирования KVEST и UniTESK, разработанных в Институте Системного Программирования (ИСП РАН). Спецификации записывались в виде пред- и постусловий. Из них извлекались вручную или автоматическим способом модели в виде автоматов или LTS. Под разрушением понималось поведение программы после обращения к ней с нарушением её предусловия. Это учитывалось при тестировании, то есть оно было безопасным. Эта таблица свидетельствует о следующем.

Во-первых, теория строилась как осмысление и обобщение того практического тестирования на основе формальных моделей, которое проводилось, в частности, в ИСП РАН.

Во-вторых, построенная теория покрывает все эти частные случаи. Конечно, общий метод генерации тестов, описанный в работе, в каждом из этих практических случаев оптимизирован с учётом указанных в таблице ограничений на реализации и спецификации.

В-третьих, теория показывает пути решения проблем тестирования для других семантик взаимодействия и других контекстов при асинхронном тестировании. Некоторые из них продемонстрированы на простейших примерах выше в разделе «Проблематика».

ТЕСТИРОВАНИЕ В ИСП РАН

<table>
<tr><th>Модель</th><th>Тестирование</th><th>Контекст</th><th>$\mathfrak{R}$</th><th>$\mathfrak{Q}$</th><th>Метод</th><th>Ограничения</th><th>Инструмент</th><th>Проекты</th></tr>
<tr><td rowspan="3">Автомат</td><td rowspan="3">Синхронное</td><td rowspan="3">–</td><td rowspan="5">$\varnothing$</td><td rowspan="5">Блокировки стимулов
δ</td><td>Обход графа</td><td>Детерминированный граф сильно-связный</td><td>KVEST
UniTESK</td><td>Группа 1</td></tr>
<tr><td rowspan="2">Обход графа
по стимулам</td><td>Недетерминиров. граф с сильно-связным полным остовным детерминиров. подграфом</td><td>UniTESK</td><td>Группа 2</td></tr>
<tr><td>Недетерминиров. граф сильно-Δ-связный</td><td>[UniTESK]</td><td>–</td></tr>
<tr><td>Автомат /LTS</td><td rowspan="3">Асинхронное</td><td rowspan="3">Неограниченные входные очереди и выходные очереди 1,2,…
+
Непосредственные реакции 0</td><td>Вызов из параллельных процессов</td><td>Дополнение к синхронному тестированию</td><td>KVEST</td><td>Группа 3</td></tr>
<tr><td rowspan="2">LTS</td><td rowspan="2">δ_1
δ_2
...</td><td rowspan="2">Блокировки стимулов
δ_0</td><td rowspan="2">Стационарное тестирование[2]</td><td rowspan="2">В стационарном состоянии[1] все стимулы принимаются (возможно, с разрушением)</td><td>KVEST</td><td>Группа 4</td></tr>
<tr><td>UniTESK</td><td>Группа 5</td></tr>
<tr><td>Синхронное/ Асинхронное</td><td>Нестационарное тестирование</td><td>Дополнение к стационарн. тестированию</td><td>[UniTESK]</td><td>Группа 6</td></tr>
</table>

[1]*Стационарное состояние – состояние, в котором есть переходы только по стимулам (нет переходов по реакциям и τ-переходов).*

2*Стационарное тестирование – тестирование, при котором стимулы посылаются в реализацию только в стационарных состояниях.*

№	Название проекта	Годы	Заказчик
1	Kernel Verification	1994-8	Nortel
	FPE	1998-9	Nortel
	XA-Core	2000	Nortel
2	OLVER	2005-6	Федеральное Агентство по науке и инновациями (Роснаука)
	ОС 2000	2005-8	НИИСИ
	Java Infrastructure	2004	Intel
	Hardware Design Testing	2006-8	НИИСИ
	Germany Banking Software	2004-5	Luxoft
3	Kernel Verification	2000	Nortel
	XA-Core	2000	Nortel
	ORB	2000	Nortel
4	GWC	2000	Nortel
5	MSR IPv6	2000-1	Microsoft Research
	Mobile IPv6	2001-2	Microsoft Research, РФФИ
	OLVER	2005-6	Федеральное Агентство по науке и инновациями (Роснаука)
	ОС 2000	2005-8	НИИСИ
	Hardware Design Testing	2006-8	НИИСИ
6	Верификация распределенных систем	2003-5	Программа Президиума РАН «Разработка фундаментальных основ создания научной распределённой информационно-вычислительной среды на основе технологий GRID»
	Проблемно-ориентированные методы автоматизированной верификации распределённых систем	2006-8	

Несколько слов следует сказать о перспективах дальнейшего развития теории конформности. Во второй главе уже были указаны два таких основных направления: 1) использование трасс готовности (*ready traces*) и соответствующей безопасной конформности ***resaco***, 2) использование приоритетов.

Трассы готовности предполагают более мощные тестовые возможности и, как следствие, имеют более узкую область применимости. Но тогда, когда такие возможности есть (множества готовности наблюдаемы) и их правомерно использовать (не возникает переспецификации), тестирование по трассам готовности оправдано. Поскольку трассы готовности схожи с ϕ-трассами, можно предположить, что в рамках одной модели трасс готовности удастся соединить как определение безопасной конформности и генерацию тестов, так и решение проблемы композиции с помощью монотонного преобразования.

Использование приоритетов можно считать чрезвычайно важной и актуальной задачей ближайшего развития теории и практики конформности. Это решило бы целый ряд проблем тестирования: 1) выход из дивергенции при наличии более приоритетного «задания» для реализации, в частности, при асинхронном тестировании реактивных систем выход из цикла осцилляции (бесконечной цепочки выдачи реакций) при поступлении стимула; 2) определение альтернативного поведения реализации при возникновении тупиков (θ-переход в реализации); 3) проблема совмещения посылки стимула с приёмом реакций в реактивных системах; 4) возможность для реализации распознавать отсутствие стимулов и др.

Важное значение имеет также развитие теории конформности в сторону специальных версий теории для ограниченных классов семантик, спецификаций и реализаций. На этом пути за счёт сужения области применимости можно получить большую эффективность. Хотя это кажется возвратом к хаосу всевозможных моделей и конформностей, на самом деле этот процесс закономерен и упорядочен, поскольку идёт в рамках общей теории как её конкретизация в частных случаях с сохранением единых понятий и

методологии. В частности, весьма многообещающим кажется соединение общей теории конформности с методами тестирования конечных автоматов (в последнее время также и LTS), основанными на обходе графа переходов. Также полезные результаты получаются в области асинхронного тестирования, когда рассматриваются специальные типы среды взаимодействия, например, многоканальные входные и выходные очереди, очереди с приоритетами и т.п. По сути здесь применяются специальные разновидности монотонного преобразования спецификаций, ориентированные на класс сред. Особенностью таких тестовых систем является преобразование спецификации и композиция со средой «на лету», в процессе тестирования.

Последняя проблема, на которой хотелось бы остановиться, это проблема «разрыва» между явными моделями конечных автоматов или LTS, обычно применяемыми в теории конформности, и тем формализмом, в котором часто на практике задаются спецификации. В частности, если спецификации записываются в виде пред- и постусловий, то, хотя в основе такой спецификации лежит LTS-модель, однако она задана неявно – как решение системы уравнений. В некоторых случаях удаётся построить LTS-модель спецификации в процессе самого тестирования, точнее, ту её часть, которая оказывается необходима и достаточна для тестирования данной реализации. Этот подход оказался весьма полезным и успешно применяется в системе UniTESK. Дальнейшее развитие может быть связано с расширением области применимости этого подхода.

Публикации автора по теме диссертации

1. Burdonov I., Kossatchev A., Petrenko A., Cheng S., Wong H. Formal Specification and Verification of SOS Kernel. BNR/NORTEL Design Forum, June 1996.
2. Баранцев А.В., Бритвина Е.Н., Бурдонов И.Б., Косачев А.С., Гоманюк С.В., Демаков А.В., Иванов А.В., Максимов А.В., Петренко А.К., Сазанов Ю.Л.,

Сортов А.А., Стефанов В.П., Сумар Г.М. Архитектура системы генерации и пропуска тестов**.** Вопросы кибернетики, Москва, 1998.

3. Burdonov I., Kossatchev A., Petrenko A., Galter D. KVEST: Automated Generation of Test Suites from Formal Specifications. Proceedings of Formal Method Congress, Toulouse, France, 1999, LNCS, No. 1708.
4. Бурдонов И.Б., Косачев А.С., Демаков А.В., Петренко А.К., Максимов А.В. Формальные спецификации в технологиях обратной инженерии и верификации программ. Труды Института системного программирования РАН, No 1, 1999.
5. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. «Программирование». 2000. No. 2.
6. Bourdonov I.B., Kossatchev A.S., Petrenko A.K., Kuliamin V.V. Experiences in using testing tools and technology in real-life applications. Proceedings of SETT'01, India, Pune, 2001.
7. Burdonov I., Kossatchev A., Demakov A., Jarov A., Petrenko A., Kuliamin V., Zelenov S. Java specification extension for automated test development. Proc. of Andrei Ershov Fourth International Conference Perspectives of System Informatics (preliminary proceedings). Novosibirsk. 2001.
8. Burdonov I., Kossatchev A., Demakov A., Jarov A., Petrenko A., Kuliamin V., Zelenov S. Java Specification Extension for Automated Test Development. Proceedings of 4-th Intl. Conf. on Perspectives of System Informatics, LNCS 2244, Springer-Verlag, 2001.
9. Bourdonov I.B., Kossatchev A.S., Petrenko A.K., Kuliamin V.V. UniTesK Test Suite Architecture. Proceedings of 11-th Symposium on Formal Methods Europe, LNCS 2391, Springer-Verlag, 2002.
10. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Асинхронные автоматы: классификация и тестирование. Труды ИСП РАН, т. 4, 2003.
11. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. «Программирование». 2003. No. 5.

12. Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов. «Программирование», 2003, No. 6.
13. Kuliamin V.V., Kossatchev A.S., Petrenko A.K., Pakoulin N.V., Bourdonov I.B. Integration of Functional and Timed Testing of Real-Time and Concurrent Systems. Perspectives of System Informatics // LNCS. No. 2890, Springer-Verlag, 2003.
14. Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A. UniTesK: Model Based Testing in Industrial Practice. Proc of 1-st European Conference on Model-Driven Software Engineering, Nurnberg, December 2003.
15. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. «Программирование». 2004. No. 1.
16. Бурдонов И.Б. Обход неизвестного ориентированного графа конечным роботом. «Программирование», 2004, No. 4.
17. Бурдонов И.Б. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. «Программирование», 2004, No. 6.
18. Бурдонов И.Б. Исследование одно/двунаправленных распределённых сетей конечным роботом. Труды Всероссийской научной конференции "Научный сервис в сети ИНТЕРНЕТ". 2004.
19. Баранцев А.В., Бурдонов И.Б., Демаков А.В., Зеленов С.В., Косачев А.С., Кулямин В.В., Омельченко В.А., Пакулин Н.В., Петренко А.К., Хорошилов А.В. Подход UniTesK к разработке тестов: достижения и перспективы. Труды ИСП РАН, т. 5, М. ИСП РАН, 2004
20. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Мета-модель функциональной спецификации распределенной системы, пригодная для тестирования. Труды Всероссийской научной конференции "Научный сервис в сети ИНТЕРНЕТ". 2004.

21. Бурдонов И.Б., Косачев А.С. Тестирование компонентов распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
22. Бурдонов И.Б., Косачев А.С. Верификация композиции распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
23. Bourdonov I., Kossatchev A., Kuliamin V. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proc. Of MBT 2006, Vienna, Austria, March 2006.
24. Бурдонов И.Б., Косачев А.С., Пономаренко В.Н., Шнитман В.З. Обзор подходов к верификации распределенных систем. ИСП РАН, препринт 16, М., 2006.
25. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, No. 5.
26. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Безопасность, верификация и теория конформности. Материалы Второй международной научной конференции по проблемам безопасности и противодействия терроризму, Москва, МНЦМО, 2007.
27. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.