И. Бурдонов, А. Косачев.
Развитие теории конформности: семантики, формальные модели, алгоритмы.
Труды Института системного программирования РАН, том 26-1, 2014 г., ISSN 2079-8156, с.27-72.
47 стр.

Развитие теории конформности: семантики, формальные модели, алгоритмы

Игорь Бурдонов <igor@ispras.ru>, Александр Косачев kos@ispras.ru

Аннотация. Статья посвящена теоретическим и практическим работам по тестированию конформности (conformance testing), которые выполнялись в ИСП РАН с 1994-го года и по настоящее время. Развитие теории конформности шло по нескольким направлением и в целом носило характер обобщения используемых семантик взаимодействия, моделей и конформностей. Необходимость такого обобщения диктовалась, прежде всего, требованиями практического тестирования. Это касается таких свойств систем как недетерминизм, частичная определённость, асинхронность, разнообразие тестовых воздействий и наблюдений над поведением реализации и т.п. При этом в центре внимания всегда находилась эффективность тестирования, определяемая как оптимизацией тестовых наборов, так и алгоритмами генерации тестов, в том числе on-fly. Мы рассматриваем основные вехи этого пути в кратком и неформальном обсуждении, уделяя внимание не деталям, а основным проблемам и способам их решения, пытаясь выявить общую тенденцию развития.

Ключевые слова: Семантика взаимодействия, конечные автоматы, LTS, IOLTS, трассы, конформность, приоритеты, симуляция, редукция, генерация тестов, оптимизация тестирования, пополнение спецификаций, композиция систем, медиаторы, моделирование, реализация, спецификация, безопасное тестирование, исследование графов, обход графа, параллельное тестирование.

1. Введение

Работы по тестированию конформности в ИСП РАН начались в 1994-ом году в рамках проекта по верификации ядра операционной системы реального времени для канадской телекоммуникационной компании Nortel Networks. Хотя цели этой работы были чисто практическими, ставилась задача разработки общей методологии и технологии тестирования конформности, пригодной для широкого класса приложений. Она получила название KVEST (Kernel VErification and Specification Technology) [1-4] и впоследствии легла в основу семейства методологий и технологий под общим названием UniTESK (Uniform Testing Kit) [5-57], развиваемого и широко используемого в настоящее время.

Тестирование конформности – это одно из основных направлений в области верификации систем на основе формальных моделей. Конформность – это отношение «соответствия» модели проверяемой системы (реализации) модели спецификационных требований к ней (спецификации). При тестировании конформность проверяется в процессе взаимодействия реализации с тестом, построенным по спецификации. Для этого спецификационные требования должны быть сформулированы в терминах такого взаимодействия, а семантика взаимодействия должна позволять доводить проверку до конца. Это значит, что тест должен заканчиваться за конечное время и выносить *вердикт*: ***pass*** или ***fail***. Набор тестов значимый, если он не ловит «ложных ошибок»: для конформной реализации все тесты выносят вердикт ***pass***. Набор тестов исчерпывающий, если он обнаруживает любую ошибку: для неконформной реализации хотя один тест вынесет вердикт ***fail***. Набор тестов полный, если он значимый и исчерпывающий.

Развитие теории конформности в ИСП РАН шло по нескольким направлением и в целом носило характер обобщения используемых семантик взаимодействия, моделей и

конформностей. В то же время исследовались возможности оптимизации тестирования для суженных, но практически полезных, классов реализаций и спецификаций. В данной статье мы рассмотрим основные вехи пути этого развития в кратком и неформальном обсуждении.

2. Конечные автоматы

2.1. Всюду определённые детерминированные конечные автоматы

Мы начинали с модели конечного автомата[1], в котором каждый переход помечается двумя символами: стимулом (input) и реакцией (output). В 1996-ом году вышел обзор [97] принципов и методов тестирования таких автоматов. По этому обзору хорошо видно, что к тому времени хорошо разработанная теория тестирования была создана только для класса детерминированных и всюду определённых конечных автоматов. Детерминированность означает, что в каждом состоянии автомата имеется не более одного перехода по каждому стимулу[2]. Всюду определённость означает, что в каждом состоянии автомата имеется не менее одного перехода по каждому стимулу. Тем самым, в каждом состоянии автомата должен быть ровно один переход по каждому стимулу и какой-нибудь реакции. Если число состояний автомата обозначить через n, число переходов – через m, а число стимулов – через p, то для таких автоматов $m=np$. Тестирование заключается в подаче на автомат последовательности стимулов и наблюдения получаемых в ответ реакций. Предполагается, что тестирование состоит из «одного сеанса» и занимает конечное время, т.е. на автомат подаётся только одна последовательность стимулов, имеющая конечную длину.

В [97] конформность опиралась на понятие эквивалентности состояний. Два состояния эквивалентны, если в этих состояниях любая последовательность стимулов порождает одну и ту же последовательность реакций. Иными словами, в эквивалентных состояниях определены одинаковые множества *трасс* как чередующихся последовательностей стимулов и реакций. Два автомата эквивалентны, если для каждого состояния одного автомата в другом автомате имеется эквивалентное ему состояние. Обычно имеются в виду только достижимые состояния, т.е. состояния, которых можно достигнуть по цепочке переходов из начального состояния.

На практике обычно рассматриваются автоматы с начальными состояниями, причём под начальным состоянием реализации понимается то состояние, с которого начинается её тестирование. Конформность таких автоматов подразумевает эквивалентность их начальных состояний. Для детерминированных и всюду определённых автоматов такая конформность совпадает с трассовой эквивалентностью и (слабой) бисимуляцией. Трассовая эквивалентность определяется как равенство множеств трасс, определённых в начальных состояниях автоматов. Слабая бисимуляция определяется как существование симметричного соответствия состояний, при котором начальные состояния соответствуют друг другу, и любая трасса, начинающаяся в состоянии a, начинается также в соответствующем ему состоянии $a`$, и, если состояние $b$ достижимо из $a$ по этой трассе, то хотя бы одно состояние $b`$, соответствующее b, достижимо из $a`$ по этой трассе. Трассовая эквивалентность и бисимуляция являются отношениями эквивалентности, порождаемыми соответствующими отношениями предпорядка: трассовым предпорядком (вложенность трасс) и симуляцией. Для всюду определённых детерминированных автоматов эти предпорядки совпадают с порождаемыми ими отношениями эквивалентности, но это не так в общем случае

1 Такой автомат называется автоматом Мили.
2 Мы называем такой детерминизм *сильным детерминизмом*.

недетерминированных и частично (т.е. не всюду) определённых автоматов. Также в общем случае не рассматривают эквивалентность автоматов, а трассовая эквивалентность (предпорядок) и бисимуляция (симуляция) различаются.

Проверка эквивалентности автоматов методом тестирования в общем случае невозможна: всегда найдётся такой реализационный автомат, который не эквивалентен спецификации, но внешне ведёт себя так же, как спецификация, т.е. выдаёт те же самые реакции в ответ на те же самые стимулы. Этот «тупик» можно обойти несколькими способами: 1) наложить ограничения на класс тестируемых реализаций, 2) наложить ограничения на спецификации, 3) использовать дополнительные тестовые возможности, а также сочетанием этих способов. Прежде всего, поскольку на автомат подаётся только одна последовательность стимулов и мы должны проверить каждый стимул в каждом состоянии, граф переходов спецификационного автомата должен быть сильно-связным.

Далее можно выделить два принципиально различных направления: тестирование с закрытым состоянием и тестирование с открытым состоянием.

При тестировании с закрытым состоянием реализация понимается как «чёрный ящик»: её состояния мы не видим. Если никаких дополнительных тестовых возможностей нет (мы можем только посылать в реализацию стимулы и наблюдать ответные реакции), то, прежде всего, налагаются ограничения на «размер» реализации. Спецификация «минимизируется», т.е. специальным алгоритмом преобразуется в эквивалентную ей спецификацию, все состояния которой попарно неэквивалентны. Тестируются только те реализации, число состояний которых не превосходит (или превосходит не более, чем на константу) числа состояний спецификации. В этом случае всегда существует алгоритм тестирования конформности, который, однако, может иметь экспоненциальную (относительно n) сложность. Для того, чтобы снизить сложность до полиномиальной (mn^2), приходится налагать ограничения на спецификацию: в ней должны существовать, так называемые, *различающие* (*distinguishing*) последовательности.

При тестировании с закрытым состоянием рассматривалась также такая тестовая возможность как *рестарт*, который гарантированно переводит реализацию из любого текущего состояния в начальное состояние. Прежде всего, при наличии рестарта граф переходов спецификации можно считать сильно-связным, если добавить в каждом состоянии переход по рестарту в начальное состояние. При наличии рестарта существует алгоритм тестирования конформности с полиномиальной сложностью (тоже mn^2).

Тестирование с открытым состоянием опирается на специальную тестовую возможность: операцию *status message*, которая позволяет узнать текущее состояние реализации без изменения его. Это самый «хороший» случай: тестирование конформности, фактически, сводится к обходу графа переходов реализационного автомата, который имеет длину $\boldsymbol{O}(mn)$, где m и n – это число переходов и состояний в реализации. Требуется сильно-связность этого графа или наличие рестарта. Заметим, что при тестировании с открытым состоянием не требуется минимизировать спецификацию. При тестировании устанавливается *естественное* соответствие состояний реализации и спецификации: проходя некоторую трассу, мы считаем соответствующими состояния реализации и спецификации после этой трассы. Для верификации конформности проверяется, что в соответствующих состояниях по стимулу получаются одинаковые реакции и соответствующие постсостояния (состояния после переходов по этим стимулу и реакции).

Развитие теории тестирования в мире поначалу шло, в основном, в направлении тестирования с закрытым состоянием, где были разработаны многочисленные алгоритмы, опирающиеся на различные последовательности стимулов в спецификации (установочные,

различающие, синхронизирующие, разделяющие, характеристические и др.), а также на те или иные гипотезы о реализации. Работы в этом направлении продолжаются и сегодня.

Для нас, однако, эти достижения оказались практически бесполезными, поскольку мы с самого начала пошли в другом направлении, положив в основу алгоритма тестирования обход графа [59]. При тестировании с открытым состоянием это был, как сказано выше, граф переходов реализации, а при тестировании с закрытым состоянием граф переходов спецификации. В последнем случае нам пришлось принимать следующую (довольно сильную) *гипотезу о реализационных состояниях*: любая ошибка – это выдача ошибочной реакции. Точнее, гипотеза говорит, что естественное соответствие состояний реализации и спецификации взаимно-однозначно. Если такая гипотеза выполнена, то, перебирая все стимулы в каждом состоянии спецификации, мы либо обнаружим ошибку (ошибочную реакцию), либо переберём все стимулы в каждом состоянии реализации. Эту гипотезу можно ослабить, разрешив нескольким спецификационным состояниям соответствовать одному состоянию реализации. В этом случае реализация может быть конформна только, если все состояния спецификации, соответствующие одному состоянию реализации, эквивалентны друг другу. Также гипотезу можно ослабить, разрешив нескольким состояниям реализации соответствовать одному состоянию спецификации. Но тогда может оказаться, что разные стимулы, подаваемые в одном состоянии спецификации, в реализацию поступают в разных её состояниях. Поэтому для полноты тестирования дополнительно нужно предположить, что в состояниях реализации, соответствующих одному состоянию спецификации, имеется одно и то же множество ошибок.

2.2. Частично определённые и недетерминированные конечные автоматы

Направление, в котором развивались наши исследования, определялось практическими потребностями. Мы начинали с детерминированных и всюду определённых автоматов, а потом постепенно снимали эти ограничения. Дело в том, что нам сразу нужно было тестировать системы, для которых в качестве адекватной модели приходилось выбирать недетерминированные и/или частично определённые конечные автоматы.

Частичная определённость и недетерминизм автомата имеют несколько различную прагматику для реализации и спецификации. Если стимул отсутствует в спецификации, то это значит, что не предъявляется никаких требований к поведению реализации при подаче этого стимула в соответствующем состоянии, т.е. стимул можно не подавать. Если же стимул отсутствует в состоянии реализации, то мы трактовали это как запрет на подачу стимула. Другую трактовку («блокировка стимула») мы стали рассматривать позже уже не для конечных автоматов. Недетерминизм реализации является, так сказать, её «природным» свойством: получая стимул, реализация действительно может выдавать в данном состоянии то одну, то другую реакцию и переходить то в одно, то в другое постсостояние недетерминированным образом. Недетерминизм же спецификации является следствием неоднозначности спецификационных требований, которые не столько предписывают реализации конкретное поведение, сколько налагают на такое поведение ограничения, тем самым, оставляя на выбор реализации, какую реакцию на стимул она будет выдавать.

Это привело к тому, что саму конформность пришлось понимать не как эквивалентность автоматов.Мы понимаем конформность как состоящую из двух частей: гипотезы о реализации и проверяемого условия. Гипотеза предъявляет к реализации требования, которые позволяют её тестировать, но сами эти требования при тестировании не проверяются и являются предусловием тестирования. Проверяемое условие – это как раз те требования к реализации, которые проверяются при тестировании.

В основном мы рассматриваем трассовые конформности, которые основаны на трассах реализации и спецификации[3]. Трассовую конформность мы называем *редукцией*, если она основана на разделении всех возможных трасс на правильные (конформные) трассы и ошибочные (неконформные) трассы (ошибки). Спецификация как раз и задаёт такое разделение. Реализация конформна, если в ней нет ошибок, т.е. множество трасс реализации вложено во множество конформных трасс, определяемое по спецификации. Тем самым, редукция является частичным (нестрогим) порядком (рефлексивное, симметричное и транзитивное отношение). Не являются редукциями трассовые конформности, которые требуют обязательного наличия в реализации тех или иных наблюдений после трасс, или которые разрешают, например, иметь в реализации любую из двух трасс, но не обе сразу.

Важно отметить, что множество конформных трасс (или его дополнение – множество ошибок) обычно задаётся спецификацией неявно: множество всех конформных трасс вычисляется по множеству трасс спецификации (трасс, определяемых ею явно), но вовсе не обязательно совпадает с ним. Поэтому редукция обычно определяется как отношение множества трасс реализации не с множеством всех конформных трасс, а с множеством трасс спецификации, т.е. трасс, явно задаваемых спецификацией, и в этом случае она уже не обязательно обладает свойствами рефлексивности, симметричности и транзитивности.

Мы использовали контрактные спецификации, основанные на пре- и постусловиях [5,6]. Предусловие абстрагируется в автоматной модели в наличие или отсутствие в том или ином состоянии перехода по тому или иному стимулу, иначе говоря, сразу требует частично определённых автоматов. Постусловие абстрагируется в предикат, которому должны удовлетворять реакция и постсостояние для данных стимула и пресостояния. Тем самым, пресостояние и стимул, вообще говоря, неоднозначно определяют реакцию и постсостояние, что и означает недетерминизм. Кроме того, контрактная спецификация задаёт спецификационный автомат имплицитно, и он эксплицируется в процессе тестирования. Это означает, что граф спецификации заранее неизвестен.

2.2.1. Частично определённые детерминированные автоматы

Сначала мы исследовали частично определённые, по, по-прежнему, детерминированные автоматы. Поскольку стимул может быть определён не в каждом состоянии реализации, мы вынуждены были использовать специальную гипотезу о реализации. Она называлась *гипотезой о допустимости* (стимулов) [59,61], а в дальнейшем такого рода гипотезы мы стали называть гипотезами о безопасности [64-67,71]. Гипотеза основана на естественном соответствии состояний (состояний после общих трасс) реализации и спецификации. Стимул называется допустимым (впоследствии они стали называться безопасными) после трассы, если он определён в состоянии спецификации после трассы. Гипотеза о допустимости предполагает, что такой стимул должен быть определён в соответствующем состоянии реализации. Заметим, что в реализации в состоянии могут быть определены также недопустимые стимулы, т.е. те, которые не определены ни в каком соответствующем состоянии спецификации. Недопустимые стимулы при тестировании конформности не подаются. Допустимые стимулы определяют допустимые трассы (впоследствии они стали называться безопасными), т.е. трассы, содержащие только допустимые стимулы. В рассматриваемом случае это просто все трассы спецификации.

[3] Конформность, основанная на соответствии состояний реализации и спецификации, рассматривается ниже в разделе 7.

Частичная определённость автомата вынудила нас вместо трассовой эквивалентности автоматов использовать конформность типа редукции: во-первых, в реализации после имеющейся в ней допустимой трассы должны быть определены все допустимые после этой трассы стимулы (гипотеза о допустимости), и, во-вторых, на каждый такой стимул реализация должна выдавать такую же реакцию, как в спецификации (проверяемое условие). Для детерминированных автоматов такая конформность эквивалентна вложенности множества трасс спецификации во множество трасс реализации. Для частично определённых детерминированных автоматов тестирование по-прежнему основано на обходе графа, но подаются не любые, а только допустимые стимулы. При тестировании с открытым состоянием строится не граф реализации, а его подграф, порождаемый допустимыми трассами.

Частичная определённость впоследствии породила две идеи: идею о безопасности тестирования и идею об отказах. Это определялось выбором той или иной интерпретации отсутствия перехода по стимулу в состоянии реализации.

В одной интерпретации отсутствие в состоянии перехода по стимулу понимается как запрет на подачу стимула в состоянии. Иначе говоря, раз переход по стимулу в состоянии не определён, значит не определено поведение автомата после подачи стимула в состоянии. Обычно это понималось так, что подавать стимул нет смысла: нечего проверять, поскольку любое поведение возможно (*хаотическое* поведение [102103]). Мы предложили расширенное понимание хаотического поведения, включив в него «опасное» поведение, при котором автомат может «сломаться», «разрушиться». Тем самым, стимул, не определённый в состоянии, не только нет смысла, но и нельзя подавать в состоянии. Так возникло понятие разрушения [64-67], впоследствии оно изображалось переходом по специальному символу γ. Тестирование безопасно, если не возникает разрушения, т.е. мы не оказываемся в состоянии, где есть γ-переход. Именно эта интерпретация используется в описанной выше гипотезе о допустимости: если стимул не определён в состоянии реализации, это понимается как наличие перехода по стимулу в состояние, где есть γ-переход.

В другой интерпретации отсутствие в состоянии перехода по стимулу понималось как вполне определённая ситуация «блокировки стимула» [64-67]. Если эту ситуацию нельзя распознавать при тестировании, то мы можем бесконечно долго ждать реакции, которой не будет. Поскольку мы хотим, чтобы тестирование заканчивалось за конечное время, такой стимул естественно считать тоже «опасным». Однако, если блокировка стимула наблюдаема, то это новый вид наблюдения, который порождает новую конформность. Возможность такого распознавания определяется интерфейсом между тестовой и тестируемой системой. Например, как это бывает в протоколах, подаваемый стимул – это сообщение, посылаемое в тестируемую систему, и предполагается получение ответного сообщения «стимул принят». Если такое ответное сообщение должно придти в течение заданного тайм-аута, то отсутствие ответа в течении тайм-аута как раз и означает блокировку стимула. Блокировку стимула следует отличать от игнорирования стимула, которая иногда также используется как интерпретация отсутствия перехода по стимулу. Например, в автомате, выдающем кофе, стимул игнорируется, если автомат «глотает» монету, но кофе не выдаёт, и стимул блокируется, если щель для приёма монет перекрыта. Наблюдаемая блокировка стимула не используется для конечных автоматов, мы использовали её впоследствии при переходе от автоматной модели к модели LTS, о чём речь пойдёт ниже.

2.2.2. Частично определённые детерминированные реализации и слабо-детерминированные спецификации

Далее мы исследовали общий случай частично определённых и недетерминированных автоматов. Поначалу мы ограничивались только детерминированными реализациями. Почему же для детерминированных реализаций спецификация может быть недетерминированной? Причина в том, что спецификация описывает не конкретную реализацию, а требования к реализации, т.е., фактически, класс возможных (конформных) реализаций. В контрактной форме задания спецификации постусловие, как сказано выше, определяет не единственную «правильную» реакцию (и постсостояние) при заданных пресостоянии и стимуле, а множество таких «правильных» реакций (и постсостояний). В графе переходов это изображается, так называемой, Δ-переходом – множеством переходов, выходящих из одного состояния и помеченных одним стимулом [61].

Поскольку тестирование происходит по трассам спецификации, нам нужно в спецификации вычислять постсостояние перехода по пресостоянию, стимулу и реакции, полученной от реализации. Спецификация, в которой такое вычисление даёт однозначный результат, называется *слабо-детерминированной*[4]. Для конечных автоматов мы ограничились только такими спецификациями, чтобы не переусложнять тестовую систему.

Недетерминизм спецификации (даже слабый) вызывает изменение конформности. От реализации требуется, чтобы она выдавала на данный стимул только «правильные» реакции, но не обязательно все. Теперь уже конформность не эквивалентна вложенности множества трасс спецификации во множество трасс трасс реализации.

Сравним тестирование детерминированной реализации для сильно- и слабо-детерминированной спецификации. Гипотезы о реализации одинаковы в обоих случаях и различаются для тестирования с открытым и с закрытым состоянием.

Тестирование с открытым состоянием, по-прежнему, опирается на гипотезы о допустимости и сильно-связности (с учётом рестарта) реализации и основано на обходе подграфа реализации, порождаемого допустимыми трассами. От случая детерминированной спецификации оно отличается только тем, что изменяется проверяемое условие, поскольку меняется конформность.

Тестирование с закрытым состоянием, по-прежнему, опирается на гипотезу о допустимости и гипотезу о реализационных состояниях реализации. Мы, по-прежнему, в каждом состоянии спецификации должны попробовать каждый определённый в нём стимул. Но теперь это отличается от обхода графа, поскольку таким образом мы пройдём хотя бы по одному, а не по каждому переходу в каждом Δ-переходе. Такой обход мы назвали *обходом по стимулам*. После подачи стимула в состоянии мы получаем от реализации какую-то реакции. Если она ошибочная, тестирование заканчивается. Если реакция «правильная», мы проходим в спецификации соответствующий ей переход из Δ-перехода. Чтобы гарантировать тестирование любой (удовлетворяющей принятым гипотезам) реализации, вводится понятие Δ-*маршрута*, который определяется как множество маршрутов, «ветвящееся» в каждом проходимом Δ-переходе, определяемом состоянием и стимулом: во множестве есть маршруты, соответствующие всем имеющимся в спецификации (и, тем самым, возможным в конформных реализациях) реакциям для этих состояния и стимула. Если тестирование

4 В [61] это называется наблюдаемым детерминизмом, но мы предпочитаем использовать этот термин для случая, когда пресостояние и стимул однозначно определяют реакцию и, быть может, неоднозначно постсостояние, т.е. когда получаемая последовательность реакций однозначно определяется последовательностью подаваемых стимулов.

полное, то все маршруты Δ-маршрута – это обходы по стимулам; такой Δ-маршрут называется Δ-*обходом* [61]. Генерация теста реализует алгоритм построения Δ-обхода.

Δ-обход существует не для всякого графа. Для слабо-детерминированной спецификации достаточным условием существования Δ-обхода является сильно-Δ-связность графа спецификации: для любых двух состояний существует Δ-маршрут, все маршруты которого начинаются в первом состоянии и заканчиваются во втором состоянии. В [61] предложен оригинальный алгоритм Δ-обхода, время работы которого равно $\boldsymbol{O}(mn^2)$ или $\boldsymbol{O}(mn\log_2 n)$ в зависимости от того, можно ли сравнивать идентификаторы состояний только на равенство или также на больше/меньше. Требуемая память равна $\boldsymbol{O}(mX+nI+m\log_2 m)$ бит, где X и I – размер в битах, соответственно, стимула и идентификатора состояния. В [19] предложен другой алгоритм Δ-обхода для подкласса графов, в которых синглетонные Δ-переходы порождают сильно-связный суграф.

2.2.3. Частично определённые недетерминированные реализации и слабо-детерминированные спецификации

В дальнейшем мы стали рассматривать также тестирование недетерминированных реализаций с той же конформностью. Тестирование, по-прежнему, основано на обходе графа, но нам требуется дополнительная гипотеза о реализации, которую назовём Δ-*гипотезой*. Поскольку реализация недетерминирована, она может выдавать разные реакции на один и тот же стимул в одном и том же состоянии. Для гарантии обнаружения ошибки мы предположим, что, если на данный стимул в данном состоянии может быть ошибочная реакция, то мы её получим сразу при первой подаче этого стимула в этом состоянии.

При тестировании с открытым состоянием, по-прежнему, нужно подать в каждом состоянии реализации каждый допустимый в нём стимул. Для этого мы обходим по стимулам подграф реализации, порождённый допустимыми трассами, который должен быть сильно-Δ-связным.

При тестировании с закрытым состоянием, по-прежнему, нужно совершить Δ-обход спецификации. Спецификация должна быть сильно-Δ-связна.

2.2.4. Факторизация спецификации

Другой подход к проблеме недетерминизма спецификации заключается в том, чтобы этот недетерминизм «устранить», тем самым сведя задачу тестирования к случаю детерминированной спецификации. Такое «устранение» возможно с помощью факторизации спецификации, основанной на классах эквивалентности состояний, переходов и/или стимулов [58]. Фактор-спецификация может оказаться слабо- или даже сильно-детерминированной, тогда как исходная спецификация была недетерминированной (даже не слабо-детерминированной).

Кроме того, факторизация является основным методом решения проблемы слишком «больших» графов спецификации. Например, подача стимула часто понимается как вызов той или иной процедуры с параметрами. Среди таких параметров могут быть числа (целые и вещественные), и понятно, что полный перебор всех значений таких параметров практически невозможен. Разумеется, чтобы такой подход был оправданным, нужна мотивированная *фактор-гипотеза* о том, что классы эквивалентности подобраны «правильно», т.е. все интересующие нас ошибки, которые возможны в реализации, обнаруживаются при тестировании по факторизованной спецификации. Факторизация используется на одном из

этапов методологии UniTESK, а именно на этапе разработки тестовых сценариев, являющихся представлениями конечных спецификационных автоматов [6].

3. Модели ввода-вывода

3.1. Асинхронные автоматы

На каком-то этапе наших практических работ мы столкнулись с тестированием систем, которые не могли адекватно моделироваться конечными автоматами. В качестве первого такого примера была многопроцессная программная система, интерфейс с которой основан на обмене сообщениями. В ответ на входное сообщение система могла выдать несколько выходных сообщений, причем, если в процессе их выдачи поступало следующее входное сообщение, оно могло изменить выходной поток.

Для таких систем мы сначала придумали автоматы с отложенными реакциями (AOP), которые похожи, но не совпадают, с хорошо известными автоматами ввода-вывода (IOSM – Input/Output State Machines), называемыми также взаимодействующими конечными автоматами (CFSM - Communicating Finite State Machines). В обоих автоматах переход из одного состояния в другой происходит либо как прием стимула – принимающий переход, либо как выдача реакции – посылающий переход, либо как пустой переход, не сопровождающийся ни приемом стимула, ни выдачей реакции. Состояния, в которых определены только принимающие переходы называются стационарными.

Дальнейшим обобщением AOP и IOSM стал *асинхронный автомат* (*AA*) [60]. Название «асинхронный» связано с тем, что выдача реакции происходит, вообще говоря, асинхронно с приемом стимула. Отметим, что наше понятие AA отличается от понятия асинхронно выполняющейся сети взаимодействующих автоматов.

На самом деле AA – это семейство автоматов, систематизируемое по нескольким критериям: *смешанные состояния*, *e-переходы*, *императивность* или *факультативность*, *финальная допустимость стимулов* и *приоритеты*. В смешанном состоянии имеются как принимающие, так и посылающие или пустые переходы. E-переход – это введённый нами переход по пустому стимулу, что понимается как переход по отсутствию стимула на входе автомата. В императивном автомате (в частности, в AOP) принимающий переход по поступившему на автомат стимулу приоритетнее посылающих и пустых переходов, а в факультативном автомате (в частности, в IOSM) они равноприоритетны. Если стимул не может быть принят, то это понимается как, так называемая, ошибка неспецифицированного ввода. Императивный автомат не может принять стимул, если стимул поступает тогда, когда автомат находится в состоянии, где нет перехода по этому стимулу. Факультативный автомат не обязан сразу принимать поступивший стимул (даже если он может это сделать): автомат может совершать посылающие и пустые переходы, а стимул остаётся ждать на входе автомата. Далее возможны три варианта: 1) стимул принимается, 2) автомат переходит в стационарное состояние, где нет перехода по данному стимулу, что приводит к ошибке неспецифицированного вида, 3) автомат бесконечно совершает посылающие и пустые переходы («зацикливается»). Если вариант 2 невозможен, то стимул считается финально допустимым. Соответственно, факультативные автоматы подразделяются на те, в которых обычная и финальная допустимости стимулов совпадают, и те, в которых это не так. Кроме того, для всех типов автоматов могут быть различные варианты взаимных приоритетов e-переходов, с одной стороны, и посылающих и пустых переходов, с другой стороны.

По указанным критериям определяются 18 типов AA. Эти типы автоматов сравнивались по реализуемым ими словарным функциям: если множество словарных функций, реализуемых автоматами одного типа, вложено во множество словарных функций,

реализуемых автоматами второго типа, то считалось, что второй тип моделирует первый тип. Было показано, что все АА разбиваются на три группы эквивалентных (моделирующих друг друга и, тем самым, реализующих одно и то же множество словарных функций) автоматов. В частности, AOP относится к группе, которая моделирует все АА, в то время как IOSM реализует более узкий класс словарных функций. Кроме словарных функций АА изучались, так называемые, сериализации – последовательности, в которых подпоследовательности стимулов/реакций совпадают с входным/выходным словом. Такие сериализации впоследствии назывались трассами, как это и принято в мировой литературе. Было показано, что классификация АА по словарным функциям совпадает с классификацией АА по их трассам. Также изучались конечные трассы АА и тестирование АА по конечным трассам.

Для АА рассматривалось стационарное и нестационарное тестирование. При стационарном тестировании стимулы подаются на автомат только в стационарных состояниях; иначе говоря, после того как на автомат подаётся стимул, ожидаются все поступающие в ответ реакции, и только после этого подаётся следующий стимул. Стационарное тестирование АА использовалось ещё в KVEST, а в UniTESK ему посвящены работы [7,16,17,19,23]. При нестационарном тестировании стимулы также подаются в стационарных состояниях, но можно подать не один стимул, а последовательность стимулов. Тем самым, только первый стимул принимается в стационарном состоянии, а последующие стимулы могут приниматься и в нестационарных состояниях.

Основная особенность тестирования АА заключалась в том, что мы не управляли поступлением реакций от реализации: получая поток реакций и подавая стимул, мы не знали, в какой точке этого потока реализация получит стимул. По сути, мы подразумевали, что реализация снабжена двумя очередями: во входную очередь поступают стимулы, а в выходную очередь – реакции. E-переход срабатывал, если входная очередь оказывалась пустой. В качестве наблюдения мы получали входное слово и выходное слово, по которым строили все возможные сериализации, т.е. трассы. Следует отметить, что после подачи стимула (или последовательности стимулов при нестационарном тестировании) мы ожидали реакций до тех пор, пока реализация не перейдёт в стационарное состояние. По сути, это означает, что в трассы включались наблюдения такой стационарности (сравни с δ-наблюдением в следующем п. 3.2). Проверялось, что хотя бы одна такая трасса имеется в спецификации, такая конформность относится к типу редукции.

Также рассматривались АА с несколькими входными и выходными очередями, когда можно параллельно послать несколько стимулов в разные (не обязательно все) входные очереди и принимать реакции из нескольких (не обязательно всех) выходных очередей. По сути идентификация стимула (реакции) уточняется номером входной (выходной) очереди так, что очереди осуществляют разбиение алфавита уточнённых стимулов (реакций). Поскольку стимулы, по крайней мере, в разные очереди можно было подавать параллельно, фактически, выполнялось нестационарное тестирование. Проблемы сериализации для АА с несколькими входными и/или выходными очередями изучались в работах [20,22]. Такого рода проблемы исследовались и в работе [106].

3.2. ioco и $ioco_{\beta\gamma\delta}$

С современной точки зрения тестирование АА – это, так называемое, *асинхронное* тестирование или тестирование *в контексте*, роль которого играли входные и выходные очереди. Наблюдение стационарности – это то же самое, что, так называемое, δ-наблюдение или *quiescence*, которое ввёл в 1991 г. F.Vaandrager [94], а в 1996 г. J.Tretmans [98] использовал в предложенной им конформности ***ioco*** (*Input Output Conformance*). Эта конформность (тоже типа редукции) та же самая, что при тестировании IOSM

(факультативный АА без e-переходов), но предполагает *синхронное* тестирование, когда никакого контекста нет и реализация находится под полным управлением теста. Именно поэтому удаётся получить при тестировании не пару входное/выходное слово, а сразу трассу наблюдений как последовательность стимулов, реакций и δ-наблюдений. Аналогично, IOSM с несколькими входными и выходными очередями соответствуют конформности ***mioco***, которую предложил ученик Tretmans`a L.Heerink [99].

Ещё одно отличие тестирования АА от ***ioco*** – это предположения о допустимости стимулов в реализации. Мы применили к АА тот же подход, что и к конечным автоматам. Частично определённые реализации разрешались, но стимулы, не определённые в реализации, считались «опасными», что впоследствии породило понятие разрушения. Гипотеза о допустимости предполагала, что после общей трассы стимул, допустимый в спецификации, допустим и в реализации. В то же время Tretmans требовал от реализаций всюду определённости, но допускал частично определённые спецификации.

Интересно, что четырьмя годами ранее, в 1992 г., в своей докторской диссертации Tretmans рассматривал частично определённые реализации, в которых отсутствие стимула трактовалось как его блокировка, правда, только при тестировании в контексте (входная очередь стимулов), но потом не возвращался к этой идее. Тем не менее, в мировой литературе стали появляться работы, в которых блокировка стимулов допускалась и рассматривалась как особый вид наблюдения.

Всё это подвигло нас на исследование систем, в которых возможно как разрушение, так и блокировка стимулов. Примером тестирования в ИСП РАН систем с блокировкой стимулов может служить работа [10]. Мы предложили конформность типа редукции, которую назвали ***ioco***$_{\beta\gamma\delta}$ [64-67]. От ***ioco*** она отличалась: 1) трактовкой отсутствия перехода по стимулу в стабильном состоянии, т.е. состоянии, где нет τ-переходов (аналог пустых переходов в АА), как блокировки стимула (блокировка может входить в трассу наблюдений), 2) наличием γ-переходов, т.е. переходов по разрушению. Кроме того, Tretmans рассматривал только системы, в которых нет бесконечной последовательности (например, цикла) τ-переходов, которая называется *дивергенцией*. Мы же допускали дивергенцию, которую в то время трактовали как разрушение. Вместо всюду определённости реализации и отсутствия дивергенции мы предложили понятия безопасного тестирования, безопасной трассы и соответствующую гипотезу о безопасности. Тестирование безопасно, если не возникают разрушение и дивергенция; гипотеза о безопасности ограничивает класс реализаций, которые могут безопасно тестироваться для данной спецификации. Отношение ***ioco*** оказывается частным случаем ***ioco***$_{\beta\gamma\delta}$ – для спецификаций без блокировок стимулов, разрушения и дивергенции.

Кроме введения блокировок стимулов, мы преследовали цель решить две важнейшие проблемы теории конформности: 1) нерефлексивность и нетранзитивность конформности, и 2) немонотонность конформности.

Нерефлексивность отношения ***ioco*** с практической точки зрения очень неудобна: казалось бы, если реализация написана как «калька» со спецификации, то всё должно быть правильно, но для частично определённой спецификации это не так. Кроме того, рефлексивность и транзитивность конформности необходимы для последовательного повышения уровня абстракции спецификации. В частности, такое повышение происходит, когда при тестировании в технологии UniTESK вместо спецификации используется более «грубая», так называемая, тестовая модель. Рефлексивность гарантирует, что домен спецификаций вложен в домен реализаций, что даёт возможность построить такую тестовую модель, которой конформна исходная спецификация. А транзитивность гарантирует, что при

тестировании по тестовой модели не будет «ложных» ошибок с точки зрения спецификации: реализация конформная спецификации конформна тестовой модели.

Причина нерефлексивности и нетранзитивности ***ioco*** в различии допустимости стимулов в реализации и спецификации: спецификация может быть частично определенной, а реализация должна быть всюду определённой. Поэтому возникла «естественная» задача о *пополнении* спецификации – её преобразовании в эквивалентную ей всюду определенную спецификацию. Кроме того, решение проблемы немонотонности, о которой скажем ниже, опирается на пополнение спецификации.

До нас обе указанные проблемы не были решены для ***ioco***. Предлагались различные пополнения, однако все они не сохраняют отношение ***ioco***. Нам удалось решить проблему пополнения, выйдя «за пределы» ***ioco***, т.е. перейдя к более общей конформности $\boldsymbol{ioco}_{\beta\gamma\delta}$. Мы предложили пополнение спецификации [67], которое сохраняет отношение ***ioco***. Более того, мы расширили реализационный домен ***ioco***, допустив реализации с блокировками, разрушением и дивергенцией, если для них выполнена гипотеза о безопасности. В дальнейшем мы оптимизировали алгоритмы такого пополнения специально для ***ioco*** [81].

Пополнение спецификации – это первый шаг к решению проблемы немонотонности конформности. Суть этой проблемы в том, что композиция реализаций, конформных своим спецификациям, оказывается, вообще говоря, не конформна композиции этих спецификаций. Эта проблема имеет самые разные «обличья». В асинхронном тестировании она известна как несохранение конформности: при тестировании в контексте обнаруживаются «ложные» ошибки, не обнаруживаемые при синхронном тестировании. В общем случае без решения этой проблемы невозможно «вычислить» спецификацию системы, если известны спецификации её компонентов и «схема компоновки». Соответственно, невозможна верификация декомпозиции системных требований, т.е. проверка того, что спецификация системы правильно «разложена» на спецификации её компонентов.

Мы предложили алгоритм *монотонного* преобразования пополненных спецификаций [67], после которого композиция реализаций, конформных своим спецификациям, конформна композиции монотонно преобразованных пополненных спецификаций компонентов. Это преобразование работает для общей конформности $\boldsymbol{ioco}_{\beta\gamma\delta}$, а для частного случая ***ioco*** (пополненные спецификации без блокировок и разрушения) предложен упрощённый алгоритм преобразования [67], усовершенствованный в [81].

4. *R*/*Q*-модель

Наши работы по ***R***/***Q***-модели см. в [65,68,70-75,77-79,84,86-87].

4.1. LTS

Модели, лежащие в основе как AA, так и отношений ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$, относятся к классу LTS (*Labelled Transition System*). В LTS переход между состояниями помечается *действием* из алфавита действий или символом τ (в AA вместо τ-перехода пустой переход). Для AA без e-переходов и конформностей ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$ алфавит действий LTS разбивается на два подалфавита: стимулов и реакций; такая LTS называется IOLTS (Input-Output LTS). Наблюдения стационарности (δ-наблюдение) и блокировок стимулов относятся к, так называемым, отказам. Отказ возникает при deadlock`e, когда реализация не может выполнять те действия, которые ей разрешены. Такой deadlock возможен только в стабильном состоянии, поскольку τ-переходы считаются всегда разрешёнными – эти переходы как бы «самопроизвольно» меняют состояние реализации, независимо от внешнего вмешательства.

Отказ – это то множество «отвергаемых» действий (*refusal set*), которые разрешены, но не могут выполняться, что и вызывает deadlock. Например, если тест при синхронном тестировании посылает стимул, а в текущем стабильном состоянии реализации нет приёма этого стимула, возникает его блокировка Если тест не посылает стимулы, но принимает все реакции, а реализация не выдаёт реакции, то возникает deadlock в стационарном состоянии.

В дальнейшем мы сосредоточились на исследовании LTS общего вида. Для LTS взаимодействие формализуется в операции параллельной композиции двух LTS. Она имеет две основные модификации: CSP и CCS, которые по-разному понимают отношение *синхронности* на алфавитах операндов. В CSP синхронные действия одинаковые, а в CCS – противоположные (на универсуме действий задано инволюционное соответствие). Состояние композиционной LTS – это пара состояний LTS-операндов. Паре переходов в операндах по синхронным действиям в композиции соответствует в синхронный τ-переход (в CSP после операции ***hide***, превращающей часть синхронных действий в τ), меняются состояния обоих операндов. Переходы по асинхронным действиям, т.е. действиям, для которых нет парного синхронного действия, а также τ- и γ-переходы в композиции сохраняются, меняется состояние только одного операнда.

Тестовый эксперимент понимается как композиция LTS реализации с LTS теста, определённой в том же (CSP) или «противоположном» (CCS) алфавите. Для наблюдения отказа, в тесте используется θ-переход, который срабатывает при обнаружении deadlock`а (в композиции превращается в τ-переход). В композиции реализации и теста есть только τ-переходы, в терминальных состояниях теста выносится вердикт.

4.2. *R*/*Q*-семантика

В наших исследованиях нас интересовали конформности типа редукции, основанные на трассах, состоящих из действий и отказов. Анализируя опыт тестирования AA и отношения ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$, мы пришли к выводу, что как модель, так и отношение конформности, прежде всего, определяются тем, какие тестовые воздействия мы можем производить над реализацией и какие наблюдения можем получить в ответ, в частности, какие отказы наблюдаемы, а какие нет. Скажем, для ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma}$ можно посылать стимул, но блокировку стимула можно наблюдать только для $\boldsymbol{ioco}_{\beta\gamma\delta}$.

Набор тестовых воздействий и наблюдений формализуется в понятии семантики взаимодействия. Эта семантика описывается с помощью, так называемой, машины тестирования, предложенной R.Milner`ом [92] и van Glabbeek`ом [93,95]. Машина представляет собой «чёрный ящик», внутри которого находится реализация. Управление сводится к тому, что оператор машины, выполняя тест, нажимает кнопки на клавиатуре машины, разрешая реализации выполнять те или иные действия, которые могут наблюдаться на дисплее машины. В *реактивной* машине Milner`а нажимается только одна кнопка (разрешается только одно действие), в ответ поступает ровно одно наблюдение, после чего можно нажимать следующую кнопку. В *генеративной* машине van Glabbeek`а нажимается сразу несколько кнопок (разрешается множество действий), машина выполняет последовательность разрешённых действий, пока не будут нажаты другие кнопки. Отсутствие выполняемых действий может наблюдаться как отказ.

Van Glabbeek систематизировал возможные наблюдения, что легло в основу его классификации отношений конформности. Однако не все известные в литературе и используемые на практике отношения попали в эту классификацию. В частности, там нет отношения ***ioco***. Причина в том, что, по van Glabbeek`у, оператор может разрешать любое (или, как вариант, любое конечное) множество действий. Кроме того, наблюдаются либо все

возможные отказы, либо никакой отказ не наблюдаем. В то же время для ***ioco*** и ***ioco***$_{\beta\gamma\delta}$ множество разрешаемых действий не любое: это либо синглетон стимула, либо множество всех реакций. Множеству всех реакций соответствует наблюдаемый отказ (δ-наблюдение), а для стимула его блокировка не наблюдаема для ***ioco*** и наблюдаема для ***ioco***$_{\beta\gamma\delta}$.

Это натолкнуло нас на мысль ввести машину тестирования, параметризуемую той или иной семантикой взаимодействия. Наша машина похожа на машину Milner`a, но кнопкой разрешается не одно действие, а множество, как у van Glabbeek`a. Набор кнопок определяет набор тестовых воздействий. Отказы, соответствующие кнопкам, разделяются на наблюдаемые (***R***-семейство) и ненаблюдаемые (***Q***-семейство). В LTS наблюдаемые отказы обычно изображаются дополнительными переходами-петлями, помеченными отказами. Такую семантику мы назвали ***R***/***Q***-семантикой [65,71]. Трассы наблюдений могут содержать действия и ***R***-отказы.

Заметим, что факультативный АА с e-переходами, равноприоритетными с посылающими и пустыми переходами, не моделируемый IOLTS, моделируется LTS общего вида. Соответствующая ***R***/***Q***-семантика содержит для каждого стимула (включая пустой стимул) кнопку, разрешающую реализации принимать этот стимул или выдавать любую реакцию. Альтернативой этому мог бы служить θ-переход в IOLTS-реализации, который в стационарном состоянии срабатывает тогда, когда тест не посылает стимулов. Однако e-переход в нестационарном состоянии не моделируется θ-переходом. Кроме того, обычно θ-переход допускается только в тесте, но не в реализации.

Что касается АА, в которых имеются приоритеты между переходами, в частности императивных АА (например, АОР), то они не моделируется обычными LTS, в которых никаких приоритетов нет: любое разрешённое действие может выполняться независимо от того, какие ещё действия разрешены. Этот факт отмечал уже van Glabbeek. В дальнейшем мы исследовали проблему приоритетов, и нам удалось определить не только LTS с приоритетами и соответствующие конформности, но и композицию таких LTS [72]. Такие LTS уже могут моделировать любые АА.

4.3. Гипотеза о безопасности и конформность *saco*

Гипотеза о безопасности в ***R***/***Q***-семантике обобщает гипотезы о допустимости стимулов в конечных автоматах, АА и IOLTS (для ***ioco***) и гипотезу о безопасности для ***ioco***$_{\beta\gamma\delta}$. Тестовое воздействие (нажатие кнопки на клавиатуре машины тестирования) считается опасным, если оно может привести к возникновению разрушения или бесконечному ожиданию наблюдения, что препятствует продолжению тестирования.

Разрушение моделирует любое нежелательное поведение системы, в том числе и ее реальное разрушение, которого нельзя допускать при тестировании. Бесконечное ожидание наблюдений бывает при отказе и при дивергенции. Отказ возникает при deadlock`e, когда никакие действия не могут выполняться и, следовательно, наблюдаться. Если отказ тоже ненаблюдаем, то никаких наблюдений не будет. Аналогичная ситуация возникает при тестовом воздействии во время дивергенции: реализация может бесконечно долго выполнять τ-переходы, и никаких наблюдений не будет. Заметим, что в отличие от семантики ***ioco***$_{\beta\gamma\delta}$ опасной считается не сама дивергенция, а попытка выхода из дивергенции.

Это определение даёт отношение ***safe in***: «кнопка безопасна в реализации после трассы». Для спецификации вводится отношение ***safe by***: «кнопка безопасна в спецификации после трассы», которое отличается для ***Q***-кнопок. Если такая кнопка безопасна после трассы, то трасса должна продолжаться каким-нибудь действием, разрешаемым этой кнопкой, хотя может продолжаться и соответствующим отказом. Кроме того, если трасса продолжается

действием, которое разрешается кнопкой, не вызывающей разрушения или попытки выхода из дивергенции, то этой действие должно разрешаться некоторой безопасной кнопкой. Эти условия определяют отношение ***safe by*** неоднозначно, поэтому при задании ***R***/***Q***-семантики и спецификации дополнительно указывается отношение ***safe by***.

Безопасность кнопок определяет безопасные трассы, в которых каждое наблюдение может быть разрешено безопасной (после префикса трассы) кнопкой. Только безопасные по ***safe in*** трассы должны проходиться при безопасном тестировании. Требование безопасности тестирования выделяет класс безопасных реализаций – тех, которые можно безопасно тестировать для проверки их конформности или неконформности заданной спецификации. Этот класс определяется следующей *(трассовой) гипотезой о безопасности*: реализация безопасна для спецификации, если 1) в реализации нет разрушения с самого начала (до нажатия первой кнопки), если этого нет в спецификации, 2) после общей трассы, безопасной как в реализации (по ***safe in***), так и в спецификации (по ***safe by***) любая кнопка, безопасная (по ***safe by***) в спецификации, безопасна (по ***safe in***) после этой трассы в реализации.

Следует отметить, что гипотеза о безопасности не проверяема при тестировании и является его предусловием. После этого определяется отношение безопасной конформности ***saco*** [65]: реализация конформна спецификации, если она безопасна и выполнено *тестируемое условие*: после общей трассы, безопасной как в реализации (по ***safe in***), так и в спецификации (по ***safe by***) любое наблюдение, возможное в реализации в ответ на нажатие безопасной в спецификации (по ***safe by***) кнопки, разрешается, т.е. имеется в спецификации.

Большинство отношений конформности типа редукции оказываются частными случаями отношения ***saco*** при выборе соответствующей семантики. Для трассового предпорядка ***R*** пусто, а ***Q*** состоит из одной кнопки, разрешающей все действия. Для ***ioco R*** состоит из одной кнопки приёма всех реакций, а ***Q*** состоит из кнопок-синглетонов для каждого стимула. Для $\boldsymbol{ioco}_{\beta\gamma\delta}$ ***R*** состоит из кнопки приёма всех реакций и кнопок-синглетонов для каждого стимула, а ***Q*** пусто.

Важно отметить, что конформность ***saco*** определяется для данной семантики только через трассы наблюдений, которые можно получить с помощью машины тестирования, маскирующей состояния LTS-реализации. Иными словами, ***saco*** использует только трассы реализации и спецификации и не зависит от их состояний. Поэтому наряду с основной моделью LTS мы использовали в наших исследованиях также *трассовую модель* как множество трасс LTS. В [67] для $\boldsymbol{ioco}_{\beta\gamma\delta}$ и в [71] для ***R***/***Q***-семантики даны необходимые и достаточные условия того, что множество трасс является трассовой моделью. В трассовой теории LTS – это способ конечного представления регулярных трассовых моделей.

В общем случае тестирование конформности ***saco*** основано на гипотезе о *глобальном тестировании* [95]: любое поведение реализации можно воспроизвести в тестовом эксперименте через конечное число попыток. Иногда используются более сильные гипотезы, например, указанные выше гипотеза о реализационных состояниях (для детерминированной реализации) или Δ-гипотеза. Если гипотеза о глобальном тестировании верна, существует полный (не обязательно конечный) набор тестов. Предложены алгоритмы генерации такого набора. При наличии рестарта набор конечных тестов эквивалентен одному (быть может, бесконечному) тесту, в котором нет бесконечной части без рестарта. Иногда удаётся построить конечный полный набор тестов (или один конечный тест, быть может, с использованием рестарта).

4.4. Полное тестирование с открытым состоянием ограниченно недетерминированных реализаций

Алгоритмы тестирования с открытым состоянием, предложенные для конечных автоматов и опирающиеся на обход (под)графа реализации при соответствующих гипотезах о реализации, легко переносятся на случай LTS в ***R***/***Q***-семантике. Они опираются на гипотезу о допустимости стимулов, которой в ***R***/***Q***-семантике соответствует гипотеза о безопасности, Δ-гипотезу и сильно-Δ-связность реализации (с учетом рестарта)[5]. При этом в ***R***/***Q***-семантике обходу по стимулам соответствует обход по кнопкам: после трассы, безопасной в спецификации, в каждом состоянии реализации нужно нажать каждую кнопку, безопасную в спецификации после этой трассы. Δ-переход определяется состоянием и кнопкой и означает множество переходов (включая добавленные переходы-петли по наблюдаемым отказам), выходящих из этого состояния и помеченных наблюдениями, разрешаемыми этой кнопкой.

Δ-гипотеза предполагает, что, если после нажатия кнопки в состоянии возможно ошибочное наблюдение, то оно возникает при первом нажатии этой кнопки в этом состоянии. Вместо этой довольно сильной гипотезы мы предложили для LTS в ***R***/***Q***-семантике *гипотезу об ограниченном недетерминизме*: для заданной константы t первые t нажатий кнопки в состоянии дают все возможные переходы Δ-перехода[6]. Если есть ошибка, она возникнет не обязательно после первого, а после одного из t нажатий. Если эта гипотеза выполнена, вместо сильно-Δ-связности реализации требуется более слабая сильно-связность.

Предложенный в [73,74,86] алгоритм тестирования выполняет параллельно две работы: исследование (learning) реализации и верификацию конформности. Оценка числа тестовых воздействий: $\boldsymbol{O}(nbt^n)$ для $t>1$, $\boldsymbol{O}(bn^2)$ для $t=1$, где b – число кнопок, а n – число состояний реализации. При наличии рестарта в каждом состоянии реализации для $t>1$ оценка равна $\boldsymbol{O}(bt^n)$. Оценка объёма вычислений (без тестовых воздействий): $\boldsymbol{O}(bn^2t^n)+\boldsymbol{O}(b^2tn^2)+\boldsymbol{O}(btn2^k)$, где k – число состояний спецификации, или первое слагаемое заменяется на $\boldsymbol{O}(bnt^n)$, если есть рестарты, или $\boldsymbol{O}(bn^3)$ для $t=1$. Для сильно-Δ-связных реализаций предложена модификация алгоритма с оценками $\boldsymbol{O}(btn^2)$ для числа тестовых воздействий и $\boldsymbol{O}(btn^3)+\boldsymbol{O}(b^2tn^2)+\boldsymbol{O}(btn2^k)$ для объёма вычислений. Заметим, что если LTS-спецификация детерминирована (каждая трасса заканчивается в одном состоянии), то сомножитель 2^k заменяется на k. Детерминированность LTS соответствует слабо-детерминированности конечных автоматов. Любую LTS можно превратить в детерминированную LTS (правда, переходы по наблюдаемым отказам будут не обязательно петлями) с помощью процедуры детерминизации [84,86,87][7].

4.5. Спецификационные тройки и отношения на них

Гипотеза о безопасности и конформность задаются спецификационной тройкой: семантика взаимодействия ***R***/***Q***, спецификационная модель (LTS или трассовая модель), отношение безопасности ***safe by***. Семантика определяет алфавит действий как все действия, разрешаемые всеми кнопками: $\boldsymbol{L} = (\cup\boldsymbol{R})\cup(\cup\boldsymbol{Q})$. Каждая тройка определяет в этом алфавите класс безопасных (по гипотезе о безопасности) реализаций и класс конформных (по ***saco***)

[5] Если реализация детерминирована, то для неё автоматически выполнена Δ-гипотеза, а сильно-Δ-связность эквивалентна сильно-связности.

[6] $t=1$ соответствует детерминированному случаю.

[7] Справедливости ради, нужно отметить, что детерминизация LTS с k состояниями может дать LTS с 2^k-1 состояниями, хотя обычно до этого не доходит.

реализаций. На одном из направлений наших исследований мы изучали вопрос о соотношении троек через соотношение определяемых ими классов безопасных и конформных реализаций. Вложенность троек определяется как равенство классов конформных реализаций и вложенность классов безопасных реализаций. Это отношение является предпорядком и индуцирует соответствующее отношение эквивалентности троек. Преобразование троек назовём вложенным (эквивалентным), если тройка-аргумент вложена в (эквивалентна) тройке-результату. Вложенное преобразование даёт тройку, которую при тестировании можно использовать вместо исходной тройки.

Эти вопросы исследовались в работе [70], где особое внимание было уделено эквивалентности троек, отличающихся только отношением ***safe by*** (нормализация отношения) или имеющих общую спецификационную модель. Последнее порождает отношение «не сильнее» для семантик: первая семантика не сильнее второй семантики, если для каждой тройки с первой семантикой найдётся эквивалентная ей тройка со второй семантикой и той же самой спецификационной моделью (но, возможно с другим отношением ***safe by***). Соответственно определяется эквивалентность семантик. На классе эквивалентных семантик изучались условия существования минимальных и наименьших семантик по отношению вложенности семейств (***R*** и ***Q***) кнопок.

Классы реализаций, определяемые тройкой, являются подклассами реализаций в одном алфавите. Если алфавиты спецификационных троек разные, то классы реализаций также рассматриваются в разных алфавитах и, чтобы их сравнивать, нужно привести их к «единому знаменателю». Таким «знаменателем» может служить любой «целевой» класс интересующих нас реализаций: вместо классов конформных и безопасно-тестируемых реализаций рассматриваются их пересечения с этим «целевым» классом реализаций. Для наших целей нам было достаточно в качестве такого «целевого» класса брать класс всех моделей в некотором «целевом» алфавите ***L***. Мы определили «приведение» троек к целевому алфавиту, что дало возможность рассматривать отношения троек в разных алфавитах: ***L***-вложенность (и ***L***-эквивалентность), при которых сохраняется подкласс конформных и не сужается (сохраняется) подкласс безопасных реализаций в алфавите ***L***.

Для тестирования особый интерес представляет случай, когда две разные семантики определяют одинаковые тестовые возможности для тестирования реализаций в алфавите ***L***: если в кнопках оставить только действия из алфавита ***L***, то семантики совпадут. Такие семантики мы назвали ***L***-эквивалентными. Для ***L***-эквивалентных семантик в [84] рассматривались алгоритмические ***L***-вложенные и ***L***-эквивалентные преобразования.

4.6. Пополнение: удаление из спецификации ненаблюдаемых отказов

Исторически конформность ***saco*** возникла как обобщение ***ioco*** на общую ***R***/***Q***-семантику. Были наследованы и две основные проблемы конформности: 1) нерефлексивность и нетранзитивность, и 2) немонотонность.

Нерефлексивность ***saco*** следует из наличия ненаблюдаемых ***Q***-отказов, так же как нерефлексивность ***ioco*** следует из ненаблюдаемости блокировок стимулов. Эта проблема решается пополнением: вложенным преобразованием спецификационной тройки. Рефлексивность гарантируется тем, что пополнение строит модель спецификации без ***Q***-отказов, а транзитивность – тем, что пополнение определяет отношение ***safe by*** = ***safe in***.

В [71] показано существование преобразования, которое сохраняет класс конформных реализаций, определяет отношение ***safe by*** = ***safe in***, и не меняет семантики взаимодействия. Однако класс безопасных реализаций может сузиться, то есть такое преобразование не всегда вложенное. Показано, что без изменения семантики пополнение существует не для всякой спецификационной тройки. Поэтому было предложено ***L***-вложенное пополнение,

которое изменяет семантику на ***L***-эквивалентную ей. Такое пополнение определено для трассовой модели, т.е. как преобразование множества трасс. Показано, что при конечном алфавите это преобразование алгоритмизуемо, исходная спецификация может быть задана в виде LTS. Строится при этом тоже LTS, но, вообще говоря, не конечная.

В [84] предложен усовершенствованный алгоритм пополнения (там оно называется ~пополнением), который по конечной LTS в конечном алфавите строит конечную LTS пополненной спецификации. Правда, этот алгоритм работает не для любого, а для, так называемого, ограниченного отношения ***safe by***: после безопасных трасс, заканчивающихся в одном множестве состояний LTS-спецификации, безопасность кнопок одинакова.

Пополненная спецификация в ***R***/***Q***-семантике эквивалентна пополненной спецификации в семантике, где все отказы наблюдаемы, т.е. в (***R***∪***Q***)/∅-семантике.

4.7. Монотонное преобразование

Как и для ***ioco*** проблема ***saco*** возникает в связи с композицией системы и заключается в том, что композиция реализаций, конформных своим спецификациям, оказывается, вообще говоря, не конформна композиции этих спецификаций. Для LTS композиция задаётся оператором параллельной композиции в духе CCS или CSP.

Первым шагом решения проблемы является пополнение спецификаций. Второй шаг – это монотонное преобразование спецификаций, при котором композиция реализаций, конформных своим спецификациям, конформна композиции монотонно преобразованных пополненных спецификаций компонентов. Кроме этого, монотонное преобразование, как и пополнение, должно: 1) не изменять тестовые возможности для тестирования реализаций в исходном алфавите ***L***, 2) сохранять конформность, т.е. сохранять подкласс конформных реализаций в алфавите ***L***, 3) не сужать возможность безопасного тестирования, т.е. не сужать подкласс безопасных реализаций в алфавите ***L***. Это означает, что монотонное преобразование должно быть ***L***-вложенным преобразованием в ***L***-эквивалентную семантику.

Монотонное преобразование позволяет решить две задачи: 1) генерация спецификации системы по спецификациям компонентов, и 2) верификация декомпозиции системных требований. Для решения первой задачи выполняется пополнение и затем монотонное преобразование спецификаций компонентов, после чего они компонуются обычным образом. Вторая задача – это верификация «согласованности» уже имеющейся спецификации системы со спецификациями компонентов. Её решение опирается на тот факт, что спецификация системы, сгенерированная для решения первой задачи, во-первых, согласована со спецификациями компонентов, а, во-вторых, среди всех таких «согласованных» спецификаций системы предъявляет к системе наибольшие требования. Поэтому (по транзитивности конформности для пополненных спецификаций) достаточно проверить, что сгенерированная спецификация конформна заданной спецификации системы. Такая проверка может выполняться аналитически[8], в том числе, моделируя тестирование сгенерированной спецификации, рассматриваемой как реализация системы, по полному набору тестов, сгенерированному из заданной спецификации системы.

Особым случаем композиции является асинхронное тестирование или тестирование в контексте, который понимается как некая среда, с которой компонуется реализация по общим правилам композиции LTS. Предполагается, что среда известна и ней нет ошибок. При асинхронном тестировании возникают две проблемы: 1) «вседозволенность»

[8] При аналитической верификации гипотезу о безопасности можно проверять так же, как проверяемое условие конформности.

(*permissiveness*), когда асинхронные тесты не ловят ошибку, обнаруживаемую синхронными тестами, и 2) «несохранение соответствия» (*non preservation of conformance*), когда асинхронные тесты ловят «ложную» ошибку. С первой проблемой, по-видимому, приходится мириться: асинхронное (и вообще композиционное) тестирование – это более «косвенное» (через контекст) тестирование реализации. Оно может не позволить создать все те режимы взаимодействия и наблюдать всё то поведение реализации, которые возможны в синхронном тестировании, когда тест и реализация взаимодействуют непосредственно друг с другом. Вторая проблема более серьёзная – это частный случай общей проблемы монотонности. Решение – монотонное преобразование, которое применяется только к спецификации реализации, а среда остаётся неизменной.

Глубинной причиной проблемы монотонности является различие в уровнях абстракции конформности и композиции. Конформность опирается на трассовую модель, а композиция – на более детальную (учитываются ещё и состояния) модель LTS. Дело в том, что на трассах наблюдений невозможно определить такую композицию, по отношению к которой композиция LTS обладала бы свойством *аддитивности*: множество трасс композиции LTS совпадает с множеством всех попарных композиций трасс LTS-операндов. Это объясняется тем, что не всякое композиционное наблюдение вычисляется по паре наблюдений в операндах: отказы так не вычисляются.

Для решения этой проблемы вводятся, так называемые, ϕ-трассы (в [67] для $\boldsymbol{ioco}_{\beta\gamma\delta}$ и в [71] для ***R***/***Q***-семантики). Кроме действий, в ϕ-трассы входят, так называемые, ϕ-символы, а ϕ-символ содержит информацию о всех действиях, переходы по которым определены в стабильном состоянии[9]. Модель ϕ-трасс занимает промежуточный уровень абстракции между моделью трасс наблюдений и LTS, достаточный для определения композиции ϕ-трасс со свойством аддитивности. Кроме того, ϕ-трассы обладают свойством *генеративности*: по ϕ-трассам LTS вычисляются все её трассы наблюдений (обратное неверно). На базе ϕ-трасс строилась теория монотонности и разрабатывались алгоритмы монотонного преобразования в [81] для ***ioco***, в [67] для $\boldsymbol{ioco}_{\beta\gamma\delta}$ и в [71] для общей ***R***/***Q***-семантики.

4.8. Удаление из спецификации неконформных трасс

Генерация тестов происходит по безопасным трассам спецификации. После такой трассы нажимается безопасная кнопка и верифицируются полученные наблюдения. Однако, если эта трасса *неконформна*, т.е. не встречается в конформных реализациях, то ошибку можно обнаружить раньше, как только получена сама эта трасса. Поэтому возникает задача оптимизации генерации тестов с помощью удаления из спецификации неконформных трасс. Существование безопасных неконформных трасс следует из нерефлексивности отношения ***saco***. Пример таких трасс для отношения ***ioco*** смотри в [65,78,81,84], есть даже пример спецификации, в которой все трассы неконформны [78,81,84]. В других семантиках спецификации могут содержать *неактуальные* трассы, которые не встречаются не только в конформных, но и в безопасных реализациях [78,84].

Пополнение (см. п.4.6) удаляет из спецификации ***Q***-отказы, а в такой спецификации все трассы конформны. Однако это пополнение может менять семантику и, соответственно, расширять исходный алфавит действий ***L***. Из-за этого могут появиться новые конформные реализации, хотя и не в алфавите ***L***. Назовём трассу ***L***-конформной, если она встречается в конформных реализациях в алфавите ***L***. Понятно, что ***L***-конформная трасса конформна, но

[9] Это похоже на *множества готовности* (ready sets) в *трассах готовности* (ready traces) [58,59].

обратное не верно. Следовательно, для тестирования «старых» реализаций (т.е. в алфавите ***L***) нужны только ***L***-конформные трассы, а остальные трассы, в том числе конформные, можно удалить. В [84] преобразование спецификации, выполняющее эту операцию, названо ∇-пополнением. Оно выполняется после ~пополнения (удаления ***Q***-отказов). Разработаны алгоритмы ∇-пополнения; для конечных LTS-спецификаций в конечном алфавите с ограниченным отношением ***safe by*** строится ∇-пополнение в виде конечной LTS. Для ***ioco*** в [81] предложен упрощённый вариант такого алгоритма.

4.9. Финальные модели спецификации

Рассмотренный выше способ задания конформности ***saco*** в общей ***R***/***Q***-семантике обладает двумя существенными недостатками, усложняющими генерацию тестов по спецификации: 1) недетерминизм LTS-спецификации, 2) необходимость кроме описания спецификационной модели давать отдельное описание отношения ***safe by***.

Мы уже говорили выше, что для конформности ***saco*** в общей ***R***/***Q***-семантике достаточной трассовой модели. LTS – это порождающий граф трассовой модели, т.е. является компактным способом представления моделей, в частности, позволяет конечным образом описывать регулярные трассовые модели. Этот способ, однако, обладает существенным недостатком: LTS, вообще говоря, недетерминирована: трасса может заканчиваться не в одном, а в нескольких состояниях. Работать с такими трассами на LTS неудобно. В то же время этот недетерминизм вовсе не является неизбежным следствием недетерминизма моделируемой системы. Причина недетерминизма LTS в том, что в ***R***/***Q***-семантике наблюдения делятся на два вида: действия и отказы, которые существенно различным образом отображаются в LTS. Если трасса продолжается как отказом R, так и действием $z \in R$, то эти два продолжения не могут быть определены в одном и том же состоянии LTS. С другой стороны, любой порождающий граф можно сделать детерминированным с помощью процедуры детерминизации. Правда, при этом переходы по отказам не обязательно будут петлями. Кроме того, при обычной детерминизации теряется информация о дивергенции (бесконечном маршруте τ-переходов).

Это натолкнуло нас на мысль использовать новую модель, которую мы назвали RTS (Refusal Transition System) и которая представляется собой детерминированную LTS в объединённом алфавите действий и ***R***-отказов. Кроме того, допускаются γ-переходы, означающие разрушение, и Δ-переходы, означающие дивергенцию (вместо бесконечного маршрута τ-переходов). LTS преобразуется в RTS модифицированным алгоритмом детерминизации, который добавляет Δ-переходы из дивергентных состояний. RTS-модель мы использовали в [84] как вспомогательное средство для удаления неконформных трасс.

Что касается отношения ***safe by***, то оно однозначно определяется по LTS-спецификации для ***R***-кнопок и тех заведомо опасных ***Q***-кнопок, нажатие которых приводит к разрушению или попытке выхода из дивергенции. Остальные опасные ***Q***-кнопки задаются множеством трасс вида $\sigma \cdot Q$, где σ безопасная трасса спецификации, а Q опасная после неё кнопка. Если это множество регулярно, ***safe by*** называется регулярным.

В [87] предложена, так называемая, *финальная* RTS, задающая одновременно спецификационную модель и отношение ***safe by***, для чего допускаются переходы по ***Q***-отказам. Есть два выделенных состояния: терминальное состояние ω и состояние γ, из которого выходит один переход, помеченный γ и ведущий в ω. В остальных состояниях нет γ-переходов, а все переходы по Δ и по ***Q***-отказам ведут в ω. Финальная RTS обладает рядом полезных для генерации тестов свойств, выгодно отличающих ее от LTS в алфавите действий:

1. **Детерминизм**. RTS детерминирована, следовательно, каждая трасса, по которой нужно генерировать тесты, соответствует одному маршруту в графе спецификации и заканчивается в одном состоянии.
2. **Безопасные кнопки.** Кнопка *P* безопасна после трассы, если конечное состояние трассы отлично от ω и γ, в нём нет Δ-перехода и, если *P* – это ***Q***-кнопка, нет перехода по *P*.
3. **Безопасные трассы**. Безопасные трассы спецификации – это все ее трассы, заканчивающиеся в состояниях, отличных от ω и γ.

Любую LTS-спецификацию с любым отношением ***safe by*** можно преобразовать в эквивалентную ей (по классам безопасных и конформных реализаций) финальную RTS-спецификацию. Предложен алгоритм такого преобразования, который строит конечную RTS по конечной LTS и конечному порождающему графу регулярного отношения ***safe by***.

5. Расширения *R*/*Q*-модели

В этом разделе мы рассмотрим три важнейших расширения ***R***/***Q***-теории: медиаторы, приоритеты и симуляцию.

5.1. Медиаторы: тестирование с преобразованием семантик

Теория конформности строится в предположении, что реализация и спецификация заданы в одной семантике. Однако на практике, как правило, требуется некоторое преобразование спецификационных тестовых воздействий (кнопок) в реализационные тестовые воздействия (кнопки) и обратное преобразование реализационных наблюдений в спецификационные наблюдения. Программа, осуществляющая эти преобразования, называется медиатором. По сути, это означает, что реализация и спецификация заданы в разных семантиках, и медиатор осуществляет преобразование семантик.

В простейшем случае различие семантик только в разных способах представления одних и тех же кнопок и наблюдений в реализации и спецификации. Медиатор выполняет взаимно-однозначное преобразование алфавитов реализации и спецификации (кнопок – в кнопки, действий – в действия и отказов – в отказы). В общем случае такого преобразования недостаточно, поскольку спецификация обычно определяется на более высоком уровне абстракции. Медиатор может оказаться довольно сложной программой, осуществляющей медиативные функции между реализацией в одной семантике и тестом, генерируемым по спецификации в другой семантике. В частности, одному нажатию спецификационной кнопки может соответствовать сеанс взаимодействия медиатора с реализацией.

Тестирование через медиатор похоже на тестирование в контексте. Там тоже тест взаимодействует с реализацией не напрямую, а через промежуточную среду взаимодействия. Отличие в том, что при тестировании в контексте спецификация задаётся в той же семантике, что и реализация. При тестировании по фактор-спецификации (см. п.2.2.4) также выполняется преобразование спецификационных кнопок в реализационные кнопки и реализационных наблюдений в спецификационные наблюдения. Но при этом используется фактор-гипотеза (эквивалентность состояний и переходов). При тестировании через медиатор, вообще говоря, фактор-гипотеза не нужна, но, если медиаторное преобразование зависит от реализационного состояния, то требуется возможность опроса реализационного состояния. Другое дело, что тестирование через медиатор обычно совмещается с факторизацией спецификации и тогда предполагает выполнение фактор-гипотезы.

В [75] определяется медиаторное преобразование и соответствующим образом модифицируются гипотеза о безопасности и конформность ***saco***. Для тестирования через медиатор предлагается модификация алгоритма полного тестирования с открытым

состоянием ограниченно недетерминированных реализаций (см. п.4.4). Отличие в том, что число *t* нажатий кнопки, достаточное для получения всех наблюдений в состоянии, зависит от медиаторного преобразования, т.е. перестаёт быть константой. Вместо него вводится ответ медиатора «все наблюдения получены».

Для тестирования с открытым состоянием также рассматривается медиаторное преобразование состояний, когда по состоянию реализации «вычисляется» соответствующее ему спецификационное состояние. Это означает, что на множестве состояний реализации существует отношение эквивалентности, и состояние спецификации взаимно-однозначно соответствует классу эквивалентных состояний реализации. В отличие от факторизации это соответствие является не предусловием тестирования (реализационной гипотезой), а частью проверяемого условия. По сути, меняется сама конформность, проверяемая таким тестированием: вместо редукции – частный случай слабой симуляции (см. ниже п. 5.3).

5.2. Приоритеты

Математические модели взаимодействия, как правило, абстрагируются от приоритетов: считается, что любое разрешённое действие может выполняться независимо от того, какие ещё действия разрешены или не разрешены. Тем самым, вводится излишний недетерминизм в описание поведения систем. В то же время на практике приоритеты широко используются. Приведём только три примера. 1) Приоритетная обработка запросов или сообщений, в том числе аппаратных прерываний. В IOLTS этому соответствуют взаимные приоритеты переходов по разным стимулам. 2) Выход из дивергенции, когда запрос, поступающий извне, прерывает внутреннюю активность системы, хотя без этого запроса она могла бы продолжаться бесконечно. В LTS этому соответствует приоритет перехода по действию над τ-переходом. 3) Прерывание цепочки действий – операция *cansel*. Этому соответствует приоритет перехода по cansel над всеми остальными переходами.

В нашей работе [72] предложена модель LTS с приоритетами, в которой переход помечается не только действием, но и множеством разрешаемых действий (МРД). Такой переход выполняется только тогда, когда разрешено выполнять именно это множество действий. В системах с приоритетами понятия стабильности и дивергенции становятся условными – в зависимости от нажатой кнопки. Состояние стабильно, если никакой переход не может выполняться при нажатии этой кнопки, и дивергентно, если в нём начинается бесконечный τ-маршрут, все переходы которого помечены этой кнопкой. Также появляется возможность *переключения кнопок*, т.е. нажатие следующей кнопки, не дожидаясь наблюдения. Дело в том, что такое переключение меняет МРД и, тем самым, выполняемые переходы в реализации. В частности, можно выйти из дивергенции по кнопке, нажав другую кнопку. Также можно нажимать кнопку, которая может вызвать ненаблюдаемый отказ, если после этого, не дожидаясь наблюдения (его может и не быть), нажать другую кнопку.

Если приоритетов нет, возможность наблюдения действия не зависит от того, какая именно нажимается кнопка, разрешающая это действие. При наличии приоритетов это становится важным, поскольку для одной такой кнопки действие может наблюдаться, а для другой – нет, и это может отражаться в спецификационных требованиях. Поэтому теперь нужно запоминать не только наблюдения, но также нажимаемые кнопки: результатом тестового эксперимента становится трасса как последовательность не только наблюдений (действий и отказов), но также и кнопок. Предложены основанные на таких трассах модификации гипотезы о безопасности и конформности ***saco*** для систем с приоритетами как без переключения кнопок, так и с переключением кнопок.

В композиции LTS с приоритетами переход из состояния одного операнда участвует в композиции только тогда, когда он помечен таким МРД, что его подмножество синхронных

действий совпадает с множеством синхронных действий, по которым определены переходы в парном состоянии другого операнда. МРД асинхронного перехода композиции приводится к алфавиту композиционной LTS, т.е. из него удаляются синхронные действия. Синхронный переход композиции соответствует паре переходов в операндах и помечается объединением МРД этих переходов, из которых удалены синхронные действия.

Для сокращения записи предложено также другое представление LTS с приоритетами, в котором множество кратных переходов по одному и тому же действию, но разным МРД, заменяется одним переходом по этому действию, помеченному предикатом от МРД. Тождественно ложный предикат соответствует отсутствию перехода, а тождественной истинный предикат – множеству кратных переходов по всем возможным МРД.

Для заданной $\mathfrak{R}/\mathfrak{Q}$-семантики LTS теста, как и раньше, определяется в том же (CSP) или «противоположном» (CCS) алфавите. Множество действий, по которым определены переходы из состояния теста, совпадает с одной из кнопок. Для обнаружения отказов используется θ-переход. Все переходы теста помечены тождественно истинным МРД.

5.3. Симуляция

Кроме конформностей, основанных на трассах наблюдений, существуют также конформности, основанные на соответствии состояний реализации и спецификации. Такие конформности называются симуляциями и требуют, чтобы правильным было не только наблюдаемое внешнее поведение реализации, но и изменение ее состояний. Выбор симуляции в качестве конформности наиболее естественен, когда состояния реализации доступны для их наблюдения, т.е. при тестировании с открытым состоянием.

Однако все рассматриваемые в литературе симуляции либо не учитывают безопасности тестирования, предполагая отсутствие дивергенции и ненаблюдаемых отказов, либо предполагают возможность прямого наблюдения дивергенции и всех отказов. Также они не учитывают возможность разрушения. В [76,80,85] мы исследовали расширение ***R***/***Q***-теории на симуляции, выбрав наиболее практический вариант слабой или наблюдаемой симуляции, которая основана на принципиальной ненаблюдаемости τ-переходов.

Слабая симуляция определяется как существование соответствия[10] состояний реализации и спецификации, при котором начальные состояния соответствуют друг другу, и если в реализации в состоянии *i* наблюдается действие *z* (до и после которого возможны τ-переходы), а среди постсостояний есть состояние *i`*, то в спецификации в любом состоянии *s*, соответствующем *i*, также наблюдается действие *z* (до и после которого возможны τ-переходы), а среди постсостояний найдётся хотя одно состояние *s`*, соответствующее *i`*. В ***R***/***Q***-семантике наблюдаться могут не только действия, но и ***R***-отказы, поэтому в определении слабой симуляции под *z* понимается любое наблюдение: действие или ***R***-отказ.

Сложнее обстоит дело с безопасностью тестирования. Для трассовой конформности ***saco*** безопасность кнопки в реализации определялась после трассы. Это значит, что кнопка безопасна в каждом состоянии после трассы, т.е. в состоянии нет дивергенции и нажатие кнопки в этом состоянии не приводит к разрушению и ненаблюдаемому отказу. Поскольку для симуляции мы предполагаем наблюдаемость состояний реализации, мы можем нажимать кнопку, которая безопасна не в каждом состоянии после трассы, а в наблюдаемом состоянии.

Соответственно, гипотеза о безопасности строится на соответствии состояний реализации и спецификации, которое мы назвали *H-соответствием*. Начальные состояния

[10] Если соответствие симметричное, то это бисимуляция.

H-соответствуют друг другу, если в них нет разрушения (в том числе после цепочки τ-переходов). Неначальные состояния *H*-соответствуют друг другу, если они могут быть достигнуты из начальных состояний нажатием кнопок, которые безопасны как реализации, так и в спецификации. Основанную на *H*-соответствии гипотезу о безопасности для симуляции мы назвали *H-гипотезой*. Она требует: 1) в реализации нет разрушения с самого начала (до нажатия первой кнопки), если этого нет в спецификации[11], 2) если кнопка безопасна в состоянии спецификации, то она безопасна в *H*-соответствующем ему состоянии реализации. Отметим, что *H*-гипотеза предъявляет к реализации более сильные требования, чем трассовая гипотеза о безопасности.

Конформность, которую мы назвали *безопасной симуляцией* и обозначили ***ss***, имеет в качестве предусловия тестирования *H*-гипотезу. Проверяемое при тестировании условие означает существование такого соответствия $R \subseteq H$, что 1) начальные состояния *R*-соответствуют друг другу, если в начальном состоянии спецификации нет разрушения (в том числе после цепочки τ-переходов), 2) если кнопка безопасна в состоянии *s* спецификации и в *R*-соответствующем ему состоянии *i* реализации её нажатие вызывает наблюдение *z* (до и после которого возможны τ-переходы) с постсостоянием *i`*, то в состоянии *s* тоже есть наблюдение *z* (до и после которого возможны τ-переходы) и одно из постсостояний *s`* *R*-соответствует *i`*. Можно также рассматривать безопасную симуляцию, основанную не на *H*-гипотезе, а на трассовой гипотезе о безопасности; мы обозначили такую конформность ***sst***. Показано, что $\boldsymbol{ss} \subset \boldsymbol{sst} \subset \boldsymbol{saco}$.

Тестирование безопасной симуляции, также как для ***saco***, основано на гипотезе о глобальном тестировании. Однако полное тестирование не всегда возможно: если для ***saco*** неконформность любой реализации можно определить за конечное время, то для ***ss*** это не так. Тем не менее предложен общий алгоритм значимого тестирования, которое на некотором подклассе спецификаций исчерпывающее и, следовательно, полное (определяет неконформность за конечное время). В частности, этот подкласс содержит все конечные спецификации в конечном алфавите.

На этом подклассе спецификаций для конечных и ограниченно недетерминированных реализаций предложены алгоритмы полного тестирования за конечное время[12] в предположении, что рестарт возможен в любой момент времени. Один алгоритм является модификацией общего алгоритма, а другой, более быстрый, алгоритм аналогичен алгоритму тестирования таких реализаций для ***saco*** (см. п.4.4). Сначала выполняется обход (под)графа реализации, потом (без тестирования) строится *R*-соответствие и, если его удаётся построить, выносится вердикт ***pass***, иначе – вердикт ***fail***.

Для быстрого алгоритма число тестовых воздействий такое же, как в п.4.4, т.е. $\boldsymbol{O}(bt^n)$ для $t>1$, и $\boldsymbol{O}(bn^2)$ для $t=1$. Объём вычислений равен $\boldsymbol{O}(bnt^n)+\boldsymbol{O}(b^2n^2t)+\boldsymbol{O}(bntk^2)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $\boldsymbol{O}(bn^3)$.

6. Модель наблюдений и модель событий

6.1. Критика *R*/*Q*-модели

R/***Q***-семантика обобщила многие известные конформности, что позволило единообразно ставить и решать многие общие проблемы, основными из которых являются

[11] Первое требование совпадает с первым требованием трассовой гипотезы о безопасности.
[12] Эти ограничения на спецификацию и реализацию такие же, как для ***saco*** (5.4).

проблемы оптимизации генерации тестов, композиции систем и учёта приоритетов. В то же время в решении этих проблем выявились серьёзные трудности: алгоритмы пополнения спецификации (удаления ненаблюдаемых отказов), удаления из спецификации неконформных трасс и монотонного преобразования оказались довольно сложными и громоздкими, что затрудняет их применение на практике. Анализ этих трудностей показал, что их причины лежат достаточно глубоко: в самой семантике взаимодействия и выбранных моделях реализации и спецификации. При всей их общности ***R***/***Q***-семантика и соответствующие ей модели недостаточно общи: в них существуют внутренние зависимости, которые и приводят к отмеченным усложнениям и трудностям.

Во-первых, нет «просто» наблюдений: они делятся на действия и отказы с существенно разной семантикой. Отказ – не просто «другое» по сравнению с действиями наблюдение, а именно множество разрешаемых, но не выполнимых действий. После отказа P не может наблюдаться действие $z \in P$. Кроме того, отказы возникают только в стабильных состояниях, где нет τ- и γ-переходов.

Во-вторых, нет «просто» тестовых воздействий: каждая моделирующая тестовое воздействие кнопка машины тестирования соответствует множеству действий, а само тестовое воздействие (нажатие кнопки) – это разрешение реализации выполнять действия именно из этого множества. Соответственно, после тестового воздействия возможно не любое (но, конечно, зависящее от реализации) наблюдение: после нажатия кнопки P может наблюдаться только действие $z \in P$ или отказ P, если он наблюдаем. Отказы могут быть не только наблюдаемыми (***R***-отказы), но также и ненаблюдаемыми (***Q***-отказы). Тем самым, кнопки тоже подразделяются на ***R***- и ***Q***-кнопки, что влияет как на безопасность тестирования, так и на конформность, и приводит к проблеме пополнения.

В-третьих, представление спецификационных требований в виде той же модели, что и для тестируемой системы, обосновано только исторически – оно возникло из идеи об эквивалентности автоматов. Однако такое представление затрудняет генерацию тестов по спецификации. Прежде всего, спецификация может содержать неконформные трассы, которые не нужны для генерации тестов, что и вызывает необходимость их удаления из спецификации. Это, однако, не решает всей проблемы оптимизации тестов.

Возможность и необходимость оптимизации тестов объясняется наличием зависимостей между ошибками, определяемыми спецификацией. Спецификация, описывая «правильное» поведение реализации, задаёт (хотя и неявно) множество A всех ошибок: реализация конформна, если в ней нет ни одной ошибки (из A). Зависимость между ошибками означает, что существует строгое подмножество ошибок $B \subset A$ такое, что любая реализация, содержащая ошибку из A, содержит также какую-то ошибку из B. Поэтому для полного тестирования вместо тестов, которые ловят все ошибки (из A), достаточно сгенерировать тесты, которые ловят все ошибки из B.

Конформность в ***R***/***Q***-семантике основана на трассах наблюдений или, при наличии приоритетов, на более общих трассах наблюдений и кнопок. Поэтому ошибка – это трасса. Отсюда следуют две тривиальные зависимости между ошибками. Во-первых, после наблюдения трассы σ всегда можно нажать кнопку[13] P, следовательно, если σP ошибка, то σ тоже ошибка. Во-вторых, если наблюдается трасса σ, то перед этим наблюдается каждый её

[13] Если кнопка опасна, её нельзя нажимать при безопасном тестировании, но это не значит, что её вообще нельзя нажимать. Иными словами, ошибкой считается любая трасса, которая не наблюдается при взаимодействии с конформными реализациями, а не только безопасная.

префикс μ, следовательно, если μ ошибка, то σ тоже ошибка. Тривиальные зависимости определяют тривиальную оптимизацию тестов, которая не создаёт проблем.

К сожалению, в ***R***/***Q***-семантике тривиальными зависимостями дело не ограничивается. Даже после удаления из спецификации ненаблюдаемых отказов и неконформных трасс всё равно остаются нетривиальные зависимости между ошибками [84]. Глубинная причина наличия таких зависимостей лежит в наличии зависимостей внутри семантики и модели при том, что спецификационная модель совпадает с реализационной.

Поскольку спецификационная модель того же типа, что модель реализации, композиция спецификаций традиционно выполняется так же, как композиция любых LTS. Из-за этого возникает несохранение конформности при композиции и необходимость монотонного преобразования спецификаций-операндов, которая выполняется перед их композицией. На самом деле, нет никаких причин ни для того, чтобы спецификация задавалась моделью того же типа, что реализация, ни для того, чтобы компоновать спецификации так же, как реализации. Возникло предположение, что при другой модели спецификации и при другой композиции спецификаций проблема сохранения конформности при композиции будет решена гораздо проще.

Что касается приоритетов, то они вносятся в ***R***/***Q***-модель как чистая «добавка», что усложняет модель и соответствующие алгоритмы. Кроме того, проблема композиции до сих пор не решена для систем с приоритетами. Возникло предположение, что всё было бы проще, если бы приоритеты были естественной частью самой семантики взаимодействия.

Всё это натолкнуло нас на мысль исследовать семантику более общего вида, в которой нет внутренних зависимостей, но есть приоритеты. Первая попытка обобщения ***R***/***Q***-семантики была предпринята в работе [69]. Там введена, так называемая, ***P***-семантика, в которой разорвана жёсткая связь между тестовым воздействием (кнопкой) и наблюдаемым отказом: наблюдаться может не только отказ, совпадающий с множеством разрешаемых действий. Теперь кнопка – это множество наблюдений, которые разрешаются этой кнопкой: как действий, так и отказов. При нажатии кнопки P отказ $r \in P$ возникает в стабильном состоянии, если в этом состоянии не определены переходы по действиям $z \in r$ (а не $z \in P$ как в случае ***R***/***Q***-семантики). В ***P***-семантике удалось определить безопасное тестирование конформности и разработать алгоритмы генерации тестов, а также ввести приоритеты, но тоже как «добавку» аналогично ***R***/***Q***-семантике. Однако в ***P***-семантике остались как разделение наблюдений на действия и отказы, так и понимание тестового воздействия как множества разрешаемых наблюдений. Поэтому она не получила дальнейшего развития.

6.2. Модель наблюдений

В [88,89] введена семантика общего вида, в которой нет внутренних зависимостей и есть приоритеты. Она задаётся двумя непересекающимися множествами: ***B*** – кнопок и ***O*** – наблюдений, без каких-либо дополнительных ограничений. Мы назвали её ***B***/***O***-семантикой. В ***B***/***O***-семантике LTS реализации содержит явные переходы как по наблюдениям, так и по кнопкам, а также, как обычно, ненаблюдаемые τ-переходы. Её можно назвать *LTS наблюдений – OTS* (от *observation* – наблюдение) в отличие от LTS в алфавите действий, которая используется в ***R***/***Q***-семантике и которую можно назвать *LTS действий – ATS* (от *action* – действие). Предполагается, что в процессе взаимодействия с реализацией мы получаем поток наблюдений, а тестовое воздействие (нажатие кнопки) лишь регулирует этот поток. Если тестового воздействия нет (никакая кнопка не нажата), реализация может выполнить любую цепочку переходов по наблюдениям и τ-переходов, начинающуюся в её

текущем состоянии. При понимании тестового воздействия в ***B***/***O***-семантике мы исходили из двух, обычно противоречащих друг другу, предположений.

Первое предположение – это приоритет тестовых воздействий над поведением реализации: не только наблюдаемым, но и ненаблюдаемым, что даёт возможность учёта приоритетов. В [88,89] показано, как в ***B***/***O***-семантике моделируются различные системы с приоритетами, в том числе системы из примеров п. 5.2. В качестве курьёза можно отметить, что модель наблюдений с таким предположением больше всего похожа на императивный АА, если стимул понимать как кнопку, а реакцию – как наблюдение. Однако императивный АА при наличии стимула не мог сделать ни одного τ-перехода, а в ***B***/***O***-семантике это разрешено вторым предположением.

Второе предположение – это основное допущение о ненаблюдаемом поведении: реализация всегда может выполнять τ-переходы независимо от тестовых воздействий.

В совокупности эти два предположения определяют следующий протокол взаимодействия: нажатие кнопки блокирует наблюдения, реализация может выполнять τ-переходы, но только конечное их число, после чего должна выполнить любой переход по нажатой кнопке. Этот переход по кнопке означает, что реализация «приняла к сведению» тестовое воздействие через конечное время после нажатия кнопки. В то же время мы никак не оговариваем, что означает это «принятие к сведению», допускается и простое игнорирование тестового воздействия, что в LTS моделируется переходом-петлёй по кнопке.

Для того, чтобы реализация всегда имела возможность «принять к сведению» тестовое воздействие, отсутствие перехода по кнопке в состоянии интерпретируется как наличие перехода-петли по этой кнопке. Разрушение γ понимается как одно из наблюдений, но при безопасном тестировании его не должно быть.

В новой модели мы по-прежнему рассматриваем трассовые конформности типа редукции. Как уже было сказано в п.2.2, редукция является предпорядком, если множество трасс спецификации – это множество все конформных трасс. Для генерации тестов наиболее удобно, чтобы спецификация явно задавала множество всех ошибок, т.е. дополнение множества конформных трасс. В этом случае тесты строятся по каждой трассе спецификации или строится один адаптивный тест, по сути, совпадающий со спецификацией. Тест выносит вердикт ***fail***, если получена ошибка, т.е. одна из трасс спецификации.

В то же время наличие тривиальных зависимостей, присущих любой трассовой конформности, позволяет задавать в спецификации не все ошибочные трассы, а только те из них, которые, во-первых, не заканчиваются кнопкой (заканчиваются наблюдением), и, во-вторых, все строгие префиксы которых не являются ошибками. Такую спецификацию мы назвали *нормализованной* и определили простейший алгоритм *нормализации* спецификации.

Для компактного представления спецификаций и тестов используются порождающие графы, которые конечны для регулярных множеств трасс. Такие графы всегда можно сделать детерминированными с помощью процедуры детерминизации. По сути, это детерминированные LTS в алфавите кнопок и наблюдений. Поскольку множество ошибок не является префикс-замкнутым (оно постфикс-замкнуто), состояния, в которых заканчиваются трассы-ошибки, помечаются как конечные.

Новая модель позволила по-новому поставить вопрос о зависимостях между ошибками. В ***B***/***O***-семантике нет нетривиальных зависимостей на классе всех реализаций, однако если ограничиться тем или иным подклассом реализаций, то на нём такие зависимости уже могут появиться просто потому, что реализации, которые содержат одну данную ошибку и не содержат остальных ошибок, могут оказаться вне этого подкласса.

Подклассы реализаций возникают, прежде всего, из-за тех или иных гипотез о безопасности, для которых тестируемые реализации ограничиваются подклассом безопасных

реализаций. В то же время для ***B***/***O***-семантики без каких-либо дополнительных ограничений тоже существуют естественные гипотезы о безопасности, определяемые двумя требованиями: 1) время ожидания наблюдения конечно – λ-гипотеза, 2) при тестировании не должно возникать разрушение реализации – γ-гипотеза.

λ-гипотеза означает, что если в префикс-замыкании спецификации трасса продолжается наблюдением, то при тестировании реализации после этой трассы гарантированно должно быть какое-нибудь наблюдение. Такая гарантия имеет место, если в реализации трасса не заканчивается в дивергентных состояниях и в каждом стабильном состоянии после трассы есть переход по какому-нибудь наблюдению. λ-гипотеза вводит новую зависимость между ошибками: если для каждого наблюдения *u* трасса μ*u* является ошибкой, то трасса μ неконформна, т.е. тоже ошибка. Соответствующая процедура λ-нормализации дополнительно к обычной нормализации добавляет в спецификацию ошибку μ и удаляет все ошибки μ*u*.

γ-гипотеза означает, что при тестировании по данной спецификации в реализации не возникает разрушение. Это сужает класс тестируемых реализаций до подкласса реализаций, удовлетворяющих γ-гипотезе, но не вводит дополнительных зависимостей между ошибками. Совокупность λ- и γ-гипотез как раз и определяют естественный для ***B***/***O***-семантики подкласс безопасных реализаций, что учитывается в относительно простой процедуре λ-нормализации спецификации.

Полезность ***B***/***O***-семантики определяется тем, что ею моделируются все другие семантики с трассовыми конформностями типа редукции, описанные в [95], а также класс ***R***/***Q***-семантик, включая ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$. Это моделирование определяется алгоритмом преобразования ATS в OTS, при котором класс всех ATS отображается в строгий подкласс класса всех OTS. Тем самым, в ***B***/***O***-семантике исходная конформность рассматривается не на классе всех LTS-реализаций, допускаемых ***B***/***O***-семантикой, а на подклассе преобразованных LTS-реализаций. Из-за этого и возникают дополнительные зависимости между ошибками, требующие дополнительной оптимизации тестов.

Так мы показали, что проблема оптимизации тестов для различных семантик оказывается частным случаем общей проблемы оптимизации тестов при ограничении тестируемых реализаций тем или иным подклассом. Вот лишь несколько общих примеров подклассов реализаций, которые применяются на практике и не связаны с гипотезами о безопасности: 1) подкласс LTS-реализаций с ограниченным числом состояний; 2) конечный (с точностью до изоморфизма) подкласс тестируемых реализаций; 3) конечный подкласс неконформных реализаций класса тестируемых реализаций (в [105] такой подкласс называется *классом неисправностей*); 4) подкласс реализаций, в котором каждая неконформная реализация содержит ошибку из некоторого заранее заданного конечного множества ошибок. В последнем случае обычно говорят, что набор тестов нацелен на поиск ошибок из указанного конечного множества, а другие ошибки могут не обнаруживаться.

6.3. Модель событий

Как было сказано в п. 4.7, проблема композиции возникает из-за неаддитивности трасс наблюдений. «Виноваты» в этом отказы, поскольку трассы действий аддитивны. Монотонное преобразование в ***R***/***Q***-семантике удалось сделать при помощи ϕ-трасс, которые обладают свойствами аддитивности и генеративности.

B/***O***-семантика тоже основана на трассах наблюдений (только общего вида), которые, вообще говоря, тоже не аддитивны. Поэтому в [90] предложена ***E***/***O***-семантика и основанная на ней *модель событий* (*ETS* – от event – событие). Вводится универсум событий ***E***, и вместо

переходов по наблюдениям в ETS используются переходы по событиям. Генеративность обеспечивается универсальной частично-определённой однозначной функцией $\boldsymbol{f}:\boldsymbol{E}\to\boldsymbol{O}$. Разрушение γ считается как событием, так и наблюдением, и $\boldsymbol{f}(\gamma)=\gamma$. Переходы совершаются только по *наблюдаемым событиям*, т.е. событиям из домена функции $\boldsymbol{f}$. Ненаблюдаемые события, однако, необходимы для аддитивности композиции; композиция событий, одно или оба из которых ненаблюдаемы, может быть наблюдаемым событием. Для композиции события разделяются на синхронные и асинхронные и задаётся коммутативная операция композиции событий, превращающая два события-операнды в событие-результат. Разрушение считается асинхронным, а кнопки всегда синхронны. На этом строится композиция ETS в духе CCS (с последующим сокрытием части синхронных символов операцией ***hide***), которая обладает свойством аддитивности.

Если бы спецификация задавалась множеством всех конформных (встречающихся в конформных реализациях) трасс событий, то свойство аддитивности гарантировало бы сохранение конформности при композиции. При этом множество трасс событий композиции спецификаций определялось бы просто как множество всех попарных композиций трасс событий спецификаций-операндов. Однако спецификация в новой модели – это нормализованное (или λ-нормализованное, если используется λγ-гипотеза о безопасности) множество ошибок. Поэтому перед композицией сначала 1) в OTS-операндах выполняется «разнормализация», дающая множество всех ошибок, 2) потом строится дополнение этого множества, т.е. множество всех конформных трасс наблюдений, 3) потом выполняется преобразование в OTS→ETS (задаваемой обратной функцией $\boldsymbol{f}^{-1}$). Получившиеся ETS-операнды компонуются, после чего нужно сделать обратные преобразования: 1) ETS→OTS (задаваемое функцией $\boldsymbol{f}$), 2) построить дополнение получившегося множества всех конформных трасс наблюдений композиции, т.е. построить множество всех ошибок, определяемых композицией, 3) выполнить нормализацию (или λ-нормализацию). Заметим, что эта процедура композиции спецификаций определяется на трассах, но разработан алгоритм её выполнения для регулярных спецификацией, задаваемых конечными OTS.

В [90] также предложен способ моделирования в $\boldsymbol{E}/\boldsymbol{O}$-семантике семантик с трассовыми конформностями типа редукции, описанные в [95], включая семантику трасс готовности (ready trace semantics), а также класса $\boldsymbol{R}/\boldsymbol{Q}$-семантик, включая ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$. Для этого предложены алгоритмы преобразования ATS-реализаций с соответствующей семантикой в ETS. Показано, что такое моделирование согласовано с композицией: композиция ETS, полученных в результате моделирования исходных ATS с исходными семантиками, эквивалентна ETS, полученной в результате моделирования композиции этих ATS. Здесь эквивалентность понимается как совпадение множеств трасс событий.

7. Параллельное тестирование

В заключении этой статьи мы хотим отметить, что почти все теоретические и все практические алгоритмы тестирования, разработанные нами для различных семантик и моделей, основаны на исследовании (обходе) (под)графа: (под)графа спецификации при тестировании с закрытым состоянием или (под)графа реализации при тестировании с открытым состоянием. Задача исследования (learning) графа, тем самым, ключевая для полноты тестирования. Ей была посвящена серия наших работ [58,59,61-63].

В то же время за последние годы размер реально используемых систем и сетей и, следовательно, размер их моделей и, следовательно, размер исследуемых графов непрерывно растёт. Проблемы возникают тогда, когда исследование графа одним автоматом (компьютером) либо требует недопустимо большого времени, либо граф не помещается в

памяти одного компьютера, либо и то и другое. Поэтому возникает задача параллельного и распределённого исследования графов. Эта задача формализуется как задача исследования графа коллективом автоматов, т.е. несколькими параллельно работающими компьютерами с достаточной суммарной памятью. Автоматы могут обмениваться между собой сообщениями для синхронизации их действий по исследованию графа.

В работах [82,83] упор сделан на ускорении тестирования за счёт распараллеливания. При этом считается, что память каждого компьютера достаточна для хранения всего исследуемого графа. Это формализуется как обход графа коллективом автоматов, которым разрешено не только обмениваться сообщениями, но и писать/читать пометки в вершинах графа. В частности, в [82,83] выполнялось функциональное тестирование различных подсистем модели процессора: кэш третьего уровня, управление прерываниями и пр. Модельные графы содержали от нескольких тысяч до нескольких миллионов узлов и несколько миллионов дуг. Тест выполнялся максимально на 150 компьютерах.

Если исследуемый граф помещается только в суммарную память всех компьютеров, но не помещается в память одного компьютера, задача резко усложняется. Она формализуется как обход графа коллективом автоматов, которым разрешено обмениваться сообщениями, но не разрешено оставлять какие-то пометки на графе (что и означает распределённое размещение графа на нескольких компьютерах). Нами предложен эффективный алгоритм такого исследования детерминированного графа, подробно изложенный в нашей работе «Обход неизвестного графа коллективом автоматов», которая в настоящее время подготовлена к печати. Более ранняя (и менее эффективная) версия этого алгоритма в сокращённом виде изложена в работе [91].

В дальнейшем предполагается сосредоточить усилия на решении задачи исследования коллективом автоматов 1) недетерминированных графов, 2) с использованием внешней памяти, 3) на подклассах графов, которые практически значимы и на которых можно разработать более быстрые алгоритмы.

ЛИТЕРАТУРА:

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. Bourdonov 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, № 1708, pp.608-621.
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 ИСП РАН, № 1, 1999, стр. 31-43.
5. Bourdonov I.B., Kossatchev A.S., Petrenko A.K., Kuliamin V.V. UniTESK Test Suite Architecture // Proceedings of FME'2002 conference, Copenhagen, Denmark, LNCS, № 2391, 2002, pp. 77-88.
6. Бурдонов И.Б., Косачев А.С., Кулямин В.В., Петренко А.К. Подход UniTESK к разработке тестов // «Программирование»,-2003, №6, стр.25-43.
7. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V., Petrenko A.K., Pakoulin N.V. Integration of Functional and Timed Testing of Real-Time and Concurrent Systems // Perspectives of System Informatics // LNCS. № 2890, Springer-Verlag, 2003, pp.450-461.

8. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. UniTESK: Model Based Testing in Industrial Practice // Proceedings of 1-st European Conference on Model-Driven Software Engineering, Nurnberg, December 2003, pp. 55-63.
9. Бурдонов И.Б., Баранцев А.В., 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, № 5, 2004, стр.121-156.
10. Калинов А.Я., Косачев А.С., Посыпкин М.А,, Соколов А.А. Автоматическая генерация тестов для графического пользовательского интерфейса по *UML* диаграммам действий // Труды ИСП РАН, № 6, 2004, стр.7-84.
11. V.Kuliamin, A.K.Petrenko. Applying Model Based Testing in Different Contexts // Proceedings of Seminar on Perspectives on Model Based Testing, Dagstuhl, Germany, September 2004.
12. V.Kuliamin. Multi-paradigm Models as Source for Automated Test Construction // Proceedings of Workshop on Model Based Testing, Barcelona, Spain, March 2004. Electronic Notes in Theoretical Computer Science 111:137-160, 2005, Elseveir.
13. А.А.Сортов, А.В.Хорошилов. Функциональное тестирование Web-приложений на основе технологии UniTESK. //Труды ИСП РАН, №8, 2004, стр.77-97.
14. V.Kuliamin. Model Based Testing of Large-scale Software: How Can Simple Models Help to Test Complex System. //Proc of ISOLA 2004, Cyprus, October 2004, pp. 311-316.
15. В.П. Иванников, А.С. Камкин, В.В. Кулямин, А.К. Петренко. Применение технологии UniTESK для функционального тестирования моделей аппаратного обеспечения // Препринт ИСП РАН, 2005.
16. V.Kuliamin, A.Petrenko, N.Pakoulin. Extended Design-by-Contract Approach to Specification and Conformance Testing of Distributed Software. Proc. of 9-th WMSCI, Orlando, USA, July 2005, v. VII. Model Based Development and Testing, pp. 65-70.
17. V.Kuliamin, A.Petrenko, N.Pakoulin. Practical Approach to Specification and Conformance Testing of Distributed Network Applications. Proc. of 2-nd ISAS 2005, Berlin, Germany, April 2005, pp. 60-73 M. Malek, E. Nett, N. Suri, eds. Service Availability. LNCS 3694, pp. 68-83, Springer-Verlag, 2005.
18. С.Грошев. Применение технологии UniTESK для тестирования систем с различной конфигурацией активных потоков управления. //Труды ИСП РАН, №9, 2006, стр. 67-81.
19. А.В.Хорошилов. Спецификация и тестирование систем с асинхронным интерфейсом. //Препринт ИСП РАН, №12, 2006.
20. А.И.Гриневич, В.В.Кулямин, Д.А.Марковцев, А.К.Петренко, В.В.Рубанов, А.В.Хорошилов. Использование формальных методов для обеспечения соблюдения программных стандартов.// Труды ИСП РАН, №10, 2006.
21. В.С. Мутилин. Паттерны проектирования тестовых сценариев. //Труды ИСП РАН, №9, 2006.
22. A.Grinevich, A.Khoroshilov, V.Kuliamin, D.Markovtsev, A.Petrenko, V.Rubanov. Formal Methods in Industrial Software Standards Enforcement. Proc. of PSI'2006, Novosibirsk, Russia, June 2006.
23. Н.В.Пакулин, А.В.Хорошилов. Разработка формальных моделей и тестирование соответствия для систем с асинхронными интерфейсами и телекоммуникационных протоколов. Программирование, №6, 2007.
24. В.П.Иванников, А.С.Камкин, А.С.Косачев, В.В.Кулямин, А.К.Петренко. Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры.// Программирование, №5, 2007, стр. 47-61.
25. Kamkin. Coverage-Directed Verification of Microprocessor Units Based on Contract Specifications. EWDTS 2008, pp. 84-87.
26. А. Камкин. Генерация тестовых программ для микропроцессоров. // Труды ИСП РАН, 2008.
27. А.К. Петренко. Унификация в автоматизации тестирования. Позиция UniTESK // Препринт ИСП РАН, т. 14, ч. 1, 2008, стр. 7-22.
28. A.Petrenko. Formal Methods and Innovation Economy: Facing New Challenges. //Proceedings of the 6th IEEE International Conference on Software Enginering and Formal Methods, Cape Town, South Africa, 10-14 November 2008.
29. S.G.Groshev. Bug localization by constructing reduced traces.// Programming and Computer Software, Volume 35, Number 3, pp. 145-157, may 2009.

И. Бурдонов, А. Косачев.
Развитие теории конформности: семантики, формальные модели, алгоритмы.
Труды Института системного программирования РАН, том 26-1, 2014 г., ISSN 2079-8156, с.27-72.
47 стр.

---

30. S. Frenkel, A.Kamkin. Verification Methodology Based on Algorithmic State Machines and Cycle-Accurate Contract Specifications. // East-West Design & Test Symposium, September 18-21, 2009, pp. 39-42.
31. В. В. Кулямин. Интеграция методов верификации программных систем. // Программирование, 35(4):41-55, 2009.
32. В. В. Кулямин. Перспективы интеграции методов верификации программного обеспечения. // Труды ИСП РАН, 16:73-88, 2009.
33. M. Chupilko. Constructing Test Sequences for Hardware Designs with Parallel Starting Operations Using Implicit FSM Models. // EWDTS-2009 (East-West Design and Test Symposium 2009). Proceedings of IEEE East-West Design & Test Symposium (EWDTS 2009), 393-396 pp.
34. M. Chupilko, A. Kamkin. Specification-Driven Testbench Development for Synchronous Parallel-Pipeline Designs. // NorChip-2009, pp.1-4.
35. Я.С. Губенко, А.С. Камкин, М.М. Чупилко. Сравнительный анализ современных технологий разработки тестов для моделей аппаратного обеспечения. // Труды ИСП РАН 2009, том17, стр.133-143.
36. С.А. Смолов. Разработка тестового набора для функционального тестирования инфраструктурного программного обеспечения Грид с применением технологии UniTESK. // Сборник научных трудов научно-практической конференции «Актуальные проблемы программной инженерии», 2009, стр.183-189.
37. Н.В.Пакулин, А.Н.Тугаенко. Разработка тестовых наборов для тестирования соответствия почтовых протоколов. // Конференция АППИ-2009, стр. 154-160.
38. В.В. Кулямин. Архитектура среды тестирования на основе моделей, построенная на базе компонентных технологий. // Труды ИСП РАН, N 18, 2010, стр. 9-44.
39. С.Г. Грошев. Технология создания гетерогенных трасс, их анализа и генерации из них отчётов. // Труды ИСП РАН, N 18, 2010, стр. 45-66.
40. М.М. Чупилко. Автоматизация системного тестирования моделей аппаратуры на основе формальных спецификаций. // Труды ИСП РАН, N 18, 2010, стр. 115-128.
41. А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Разработка тестового набора для верификации реализаций протокола безопасности IPsec v2. // Труды ИСП РАН, № 18, 2010, стр. 151-182.
42. M. Chupilko. Models of Synchronous Hardware Designs Based on FSM at Different Abstraction Levels: Application to Functional Verification // EWDTS-2010 (East-West Design and Test Symposium 2010). Proceedings of IEEE East-West Design & Test Symposium (EWDTS 2010), 127-130 pp.
43. M. Chupilko, A. Kamkin. Developing cycle-accurate contract specifications for synchronous parallel-pipeline hardware: application to verification. // BEC-2010 (Baltic Electronics Conference 2010). Proceedings of the 12th Biennial Baltic Electronics Conference (BEC 2010), 185-188 pp.
44. N. Pakulin, A. Tugaenko. Specification Based Conformance Testing for Email Protocols. // Proceedings of ISoLA 2010, pp.371-382. Heraclion, Greece, 2010.
45. В. В. Кулямин. Компонентная архитектура среды для тестирования на основе моделей. // Программирование, 36(5):54-75, 2010.
46. Н.В. Пакулин, А.Н. Тугаенко. Тестирование протоколов электронной почты Интернета с использованием моделей. // Труды ИСП РАН, том 20, 2011 г.
47. Kamkin. Simulation-Based Verification with Time-Abstract Models. EWDTS, 2011.
48. M. Chupilko, A. Kamkin. A TLM-based approach to functional verification of hardware components at different abstraction levels. LATW, 2011.
49. А. Камкин, М. Чупилко. Механизмы поддержки функционального тестирования моделей аппаратуры на разных уровнях абстракции, Труды ИСП РАН, 2011.
50. M.Chupilko. C++TESK-SystemVerilog united approach to simulation-based verification of hardware designs. EWDTS-2011.
51. V. Kuliamin, N. Pakulin, A. Tugaenko. Case Studies of Summer Model-based Testing Framework, Model-based Testing User Conference, October 18-20, 2011.
52. В.П. Иванников, А.К. Петренко. Модели в разработке и анализе программных систем. Ломоносовские чтения, 2011.

И. Бурдонов, А. Косачев.
Развитие теории конформности: семантики, формальные модели, алгоритмы.
Труды Института системного программирования РАН, том 26-1, 2014 г., ISSN 2079-8156, с.27-72.
47 стр.

---

53. Y. Gerlits, A. Khoroshilov. «Model-Based Testing of Safety Critical Real-Time Control Logic Software» // In Proceedings of the Seventh Workshop on Model-Based Testing (MBT 2012), Tallin, Estonia, March 25, 2012.
54. N. Pakulin. Integrated Modular Avionics: New Challenges for МВТБ. // ETSI TTCN-3 User Conference and Model Based Testing Workshop, Bangalore, India, 11-14 June 2012.
55. Шнитман В.З., Никешин А.В., Пакулин Н.В. Применение моделей для тестирования протоколов безопасности. // Всероссийская конференция «Инфокоммуникационные технологии в научных исследованиях», ИКИ РАН, 14-16 ноября 2012 г.
56. Чупилко М.М. Разработка тестовых систем для многомодульных моделей аппаратуры. Программирование, 2012, №1, с.47-58
57. http://www.UniTESK.com
58. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. «Программирование». 2000. № 2.
59. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. «Программирование». 2003. № 5.
60. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Асинхронные автоматы: классификация и тестирование. // Труды Института системного программирования РАН, №4, 2003, стр.7-84.
61. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. «Программирование». 2004. № 1.
62. И.Б.Бурдонов. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. Программирование, №6, 2004.
63. И.Б.Бурдонов. Обход неизвестного ориентированного графа конечным роботом. Программирование, №4, 2004.
64. I.B.Bourdonov, A.S.Kossatchev, V.V.Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proceedings of the Workshop on Model Based Testing (MBT 2004), Elsevier, 2006.
65. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Формализация тестового эксперимента.// Программирование, №5, 2007, стр. 3-32.
66. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Безопасность, верификация и теория конформности. «Материалы второй международной научной конференции по проблемам безопасности и противодействия терроризму. МГУ 2006», М., МЦНМО, 2007, стр. 135-158.
67. И.Б.Бурдонов, Косачев А.С., В.В.Кулямин. Теория соответствия для систем с блокировками и разрушением. // «Физ-мат лит» Наука, Москва, 2008. 412 с.
68. И.Б.Бурдонов, Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. // Труды ИСП РАН, N 14.1, 2008, стр.23-54.
69. И.Б.Бурдонов, Косачев А.С. Обобщённые семантики тестового взаимодействии. // Труды ИСП РАН, N 15, 2008, стр.69-106.
70. И.Б.Бурдонов, Косачев А.С. Эквивалентные семантики взаимодействия. // Труды ИСП РАН, N 14.1, 2008, стр.55-72.
71. И.Б.Бурдонов. Теория конформности (функциональное тестирование программных систем на основе формальных моделей). LAP Lambert Academic Publishing, 2011, 428 стр. (докторская диссертация И.Б.Бурдонова 2008 г.)
72. И.Б. Бурдонов, А.С. Косачев. Системы с приоритетами: конформность, тестирование, композиция. // «Программирование», -2009, №4–стр. 24-40.
73. И.Б. Бурдонов, А.С. Косачев. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. // Труды ИСП РАН, N 17, 2009, стр.161-192.
74. И.Б. Бурдонов, А.С. Косачев. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. // «Программирование», -2009, №6–стр 3-18.
75. И.Б. Бурдонов, А.С. Косачев. Тестирование с преобразованием семантик. // Труды ИСП РАН, N 17, 2009, стр.193-208.
76. И.Б. Бурдонов, А.С. Косачев. Тестирование конформности на основе соответствия состояний. // Труды ИСП РАН, N 18, 2010, стр. 183-220.

77. Kossachev, I.Burdonov. Formal Conformance Verifcation, Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil, pp.1-6.
78. А.С. Косачев, И.Б.Бурдонов. Семантики взаимодействия с отказами, дивергенцией и разрушением. // «Программирование», -2010, №5–стр 3-23.
79. И.Б.Бурдонов, А.С.Косачев, Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 1. Гипотеза о безопасности и безопасная конформность, «Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №4, 2010, стр.124-133
80. И.Б.Бурдонов, А.С.Косачев, Безопасное тестирование симуляции систем с отказами и разрушением, «Моделирование и анализ информационных систем», Том 17, Номер 4, 2010. стр. 27—40.
81. И.Б.Бурдонов, А.С.Косачев. Пополнение спецификации для ***ioco***. // Программирование, №1, 2011, с. 3-18.
82. Бурдонов И.Б., Грошев С.Г., Демаков А.В., Камкин А.С., Косачев А.С., Сортов А.А., Параллельное тестирование больших автоматных моделей, Вестник ННГУ, №3, 2011 г., стр. 187-193
83. Demakov, A. Kamkin, A. Sortov. High-Performance Testing: Parallelizing Functional Tests for Computer Systems Using Distributed Graph Exploration. Open Cirrus, 2011.
84. И.Б.Бурдонов, А.С.Косачев. Удаление из спецификации неконформных трасс // Препринт ИСП РАН, Препринт 23, 2011 г, с. 1-219.
85. I.B.Bourdonov, A.S.Kossatchev. Safe simulation testing of systems with refusals and destructions // Automatic Control and Computer Sciences, Vol. 45, №7, 2011, pp. 380-389.
86. И.Б.Бурдонов, А.С.Косачев. Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования // Вестник Томского Государственного Университета, № 2(15), 2011.
87. И.Бурдонов, А.Косачев, Финальные модели спецификации, Труды ИСП РАН, том 22, 2012 г., ISSN 2079-8156, с. 233-276
88. И.Бурдонов, А.Косачев, Зависимости между ошибками на классах тестируемых реализаций, Труды ИСП РАН, том 23, 2012 г., ISSN 2079-8156
89. И.Б.Бурдонов, А.С.Косачев. Формализация тестового эксперимента –II. // Программирование, №4, 2013, с. 3-27.
90. И.Б.Бурдонов, А.С.Косачев. Согласование конформности и композиции // Программирование, №6, 2013, с. 3-15.
91. И.Бурдонов, А.Косачев, Обход неизвестного графа коллективом автоматов, Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: все грани параллелизма», 2013, изд. МГУ, стр. 228-232.
92. Milner R. Modal characterization of observable machine behaviour. In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer, pp. 25-34.
93. van Glabbeek R.J. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, CONCUR’90, Lecture Notes in Computer Science 458, Springer-Verlag, 1990, pp 278–297.
94. Vaandrager F. On the relationship between process algebra and Input/Output Automata. In Logic in Computer Science, pp. 387-398. Sixth Annual IEEE Symposium, IEEE Computer Society Press, 1991.
95. van Glabbeek R.J. The linear time - branching time spectrum II; the semantics of sequential processes with silent moves. Proceedings CONCUR ’93, Hildesheim, Germany, August 1993 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993, pp. 66-81.
96. De Nicola R., Segala R. A process algebraic view of Input/Output Automata. Theoretical Computer Science, 138:391-423, 1995.
97. D. Lee and M. Yannakakis. Principles and Methods of Testing Finite-State Machines. A survey. Proceedings of the IEEE, Vol. 84, № 8, 1996, pp. 1090–1123.
98. Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.

99. Heerink L. Ins and Outs in Refusal Testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1998.
100. Marine Tabourier, Ana Cavalli and Melania Ionescu , A GSM-MAP Protocol Experiment Using Passive Testing, Proceeding of FM'99 (World Congress on Formal methods in development of Computing Systems), Toulouse (France), 20-24 September 1999.
101. Jard C., Jéron T., Tanguy L., Viho C. Remote testing can be as powerful as local testing. In Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX' 99, Beijing, China, J. Wu, S. Chanson, Q. Gao (eds.), pp. 25-40, October 1999.
102. van der Bijl M., Rensink A., Tretmans J. Compositional testing with ioco. Formal Approaches to Software Testing: Third International Workshop, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Editors: Alexandre Petrenko, Andreas Ulrich ISBN: 3-540-20894-1. LNCS volume 2931, Springer, pp. 86-100.
103. van der Bijl M., Rensink A., Tretmans J. Component Based Testing with ioco. CTIT Technical Report TR-CTIT-03-34, University of Twente, 2003.
104. Alexandre Petrenko, Nina Yevtushenko: Testing from Partial Deterministic FSM Specifications. IEEE Trans. Computers 54(9): 1154-1165 (2005).
105. Adenilso da Silva Simão, Alexandre Petrenko, Nina Yevtushenko: Generating Reduced Tests for FSMs with Extra States. TestCom/FATES 2009: 129-145.
106. Maurice P. Herllihy, Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects //ACM Transactions on Programming Languages and Systems, Vol. 12, No. 3, July 1990, pp. 463-492.

И. Бурдонов, А. Косачев.
Развитие теории конформности: семантики, формальные модели, алгоритмы.
Труды Института системного программирования РАН, том 26-1, 2014 г., ISSN 2079-8156, с.27-72.
47 стр.
