Безопасность, верификация и теория конформности

И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин

1. Введение и постановка проблемы

Кризис в индустрии производства ПО в 2000 году не только отрезвил участников рынка, развеяв множество грандиозных, хотя и несбыточных планов, но и вскрыл возникший перекос в технологиях разработки программ. Они позволяли достаточно быстро и с небольшими затратами создавать программные комплексы такой сложности, которая и не снилась даже большим коллективам разработчиков еще в середине 80-х годов. Однако для проверки их безопасности и корректности работы использовались те же методы, которые были изобретены еще в 60-х годах. Если до кризиса клиента еще можно было убедить отдать деньги за сырой продукт, распаляя его надежды на потрясающие результаты внедрения: снижение издержек, повышение эффективности и пр., то после ситуация резко изменилась. Теперь руководители проектов должны были, скрепя сердце, тратить драгоценное время и не менее дорогие усилия разработчиков не только на то, чтобы создать нечто новое и невиданное, но и продемонстрировать заказчику, что оно действительно решает нужные ему задачи и делает это правильно. Неожиданно высокая трудоемкость этой второй части работ, на которую раньше и внимания-то особо не обращали, вынудила заняться поисками технологий, позволяющих сделать это более эффективно.

В новых обстоятельствах внезапно пригодилась «нерыночная» увлеченность программистов первой волны, уже ставших классиками Computer Science, корректностью работы программ. Стали востребованными созданные на основе их идей техники разработки ПО, изначально работающего правильно, и методы проверки правильности функционирования произвольно взятой программы. Даже компании, чьи продукты 10 лет назад славились далеко не идеальной стабильностью и надежностью работы, провозглашают курс на повышение надежности и безопасности работы ПО и проводят для своих разработчиков и партнеров тренинги по техникам написания корректных и безопасных программ.

Прямо на наших глазах складывается новая дисциплина, стоящая на границе между теоретической информатикой и такими инженерными науками, как инженерия ПО и системная инженерия, — *тестирование на основе моделей*. Во многом она опирается на работы различных исследователей, выполненные в 80-х и в 90-х годах прошлого века и посвященные проблемам тестирования телекоммуникационных протоколов. Самые первые же статьи, которые с полным правом можно отнести с этой области, появились еще в конце 60-х–начале 70-х годов. Тем не менее, серьезный практический интерес к ее результатам возник только в начале XXI века, в связи с востребованностью эффективных методов контроля и обеспечения качества сложных программных и программно-аппаратных систем в индустрии.

1.1. Безопасность, конформность и тестирование

В первом приближении будем считать, что система безопасна (secure), если при любом взаимодействии с ней она не может себя вести недекларированным образом. Декларированность поведения определяется требованиями к системе. Конформность понимается как соответствие требованиям, то есть отсутствие недекларированного поведения. Верификацией системы называют проверку её соответствия требованиям.

Поведение целевой (исследуемой) системы описывается в терминах её «внешнего», то есть наблюдаемого взаимодействия с окружением. У системы может быть и внутренняя, не проявляющаяся во внешних взаимодействиях, ненаблюдаемая активность.

Рассматривается широко распространенный частный случай взаимодействия, при котором система и окружение обмениваются между собой дискретными порциями информации (сообщениями). Такие системы называют системами ввода-вывода [23,27,30]. Сообщение, передаваемое из окружения в систему, называется стимулом (input), а сообщение, передаваемое из системы в окружение, – реакцией (output). Следуя нотации алгебры процессов CCS (Calculus of Communicating Systems [25, 26]), обозначим стимулы как `?m`, а реакции – как `!m`, где `m` – символ сообщения.

Чтобы формально определить соответствие системы требованиям, нужно, прежде всего, формализовать и то и другое. Для этого объекты реального мира отображаются в модельные (математические) объекты. Модель требований называется спецификацией, а модель целевой системы – реализацией. Между объектами-моделями задается отношение конформности как математическое соответствие, то есть подмножество декартового произведения множеств реализаций и спецификаций. Если реализация как модель системы конформна спецификации как модели требований, то считается, что система конформна требованиям. Верификация проверяет отношение реализации к спецификации, но само моделирование, то есть соответствие системы и требований их моделям, предполагается «правильным» и не проверяется. Точно также предполагается, что отображение реальной конформности в модельное отношение также «правильное», то есть модельное отношение является формализацией интуитивного понимания требований [17].

Спецификация считается известной и заданной. Что касается реализации, то, так называемая, «тестовая гипотеза» предполагает лишь, что модель системы существует, но она может быть как известна, так и неизвестна [1]. Что делать в последнем случае? Если конформность выражается в терминах взаимодействия, становится возможным тестирование как проверка конформности в эксперименте (динамическая верификация). При этом по спецификации генерируется набор модельных тестов и определяется формальное отношение «реализация прошла (passes) тест». Набор тестов называется полным, если реализация конформна спецификации тогда и только тогда, когда реализация проходит каждый тест из набора. После этого модельные тесты транслируются в реализационное представление и пропускаются на целевой системе. Если все они проходят, то конформность системы требованиям считается доказанной. Разумеется, здесь предполагается, что трансляция тестов осуществлена «правильно».

1.2. Композиционная система и запрещённые действия

Целевая система может быть (и, как правило, является) не монолитной, а состоящей из компонентов. В этом случае всякое взаимодействие системы с окружением включает взаимодействие компонентов между собой. Последнее становится внутренней, ненаблюдаемой извне активностью. Возникает вопрос: можно ли построить безопасную систему из опасных компонентов? Можно, если учитывать, что окружение каждого компонента не произвольно, а ограничено, частично или полностью, другими компонентами, поведение которых определяется требованиями к ним.

В этом случае под опасным компонентом следует понимать не любой компонент, а такой, поведение которого определено, но частично. Для этого спецификация должна описывать не только декларированное поведение, но и случаи, когда поведение «имеет право» быть любым, включая заведомо не разрешенное [9,24]. Такое поведение будем называть «запрещенным» и обозначать в спецификации символом γ. Компонент, конформный такой спецификации, будем называть условно-безопасным. Если условно-безопасный компонент конформен своей спецификации, то он может иметь опасное поведение, но нам известно, при каком взаимодействии этого компонента с его окружением это возможно, а при каком нет. Композиционная система, построенная из условно-безопасных компонентов, безопасна, если запрещенное поведение в любом

компоненте окажется недостижимо при любом взаимодействии композиционной системы с её окружением.

Приведём примеры запрещенного поведения. 1) В системах динамического распределения памяти поведение процедуры освобождения памяти не декларировано для освобождения незахваченной ранее памяти. 2) «Кнопка запуска ракеты» в некоторых случаях явно не разрешается.

1.3. Системы размеченных переходов

В качестве моделей (реализаций и спецификаций) выбирем достаточно традиционные системы размеченных переходов – LTS (Labelled Transition System). Одна из причин такого выбора заключается в том, что для таких систем определены операции композиции, заимствуемые из алгебр процессов.

LTS – это ориентированный граф, в котором вершины называются состояниями, выделено одно начальное состояние, дуги помечены символами действий и называются переходами. Действием может быть передача стимула или реакции из заданного вместе с LTS алфавита L внешних действий, а также внутреннее, ненаблюдаемое действие, традиционно обозначаемое символом τ. Дополнительно введем запрещённое действие γ. Переход из состояния s в состояние $s`$ по действию $z$ обозначается $s{-\!\!-}z{\to}s`$. Для дальнейшего нам понадобится понятие стабильного состояния – состояния, в котором нет τ- и γ-переходов. Теперь безопасная система определяется как система, в LTS-модели которой нет γ-переходов, достижимых из начального состояния.

1.4. Композиция LTS

Определим композицию LTS, которая строит LTS-модель композиционной системы по LTS-моделям её компонентов. Такую композицию будем называть *прямой*. Она соответствует оператору композиции в алгебре процессов CCS. Результат композиции LTS $\mathbf{s_1}$ (в алфавите A) и $\mathbf{s_2}$ (в алфавите B) – это третья LTS, обозначаемая $\mathbf{s_1}\|\mathbf{s_2}$. Её состояния – это пары состояний LTS-операндов, начальное состояние – это пара начальных состояний. Стимул $?m$ и реакция $!m$ называются *противоположными* действиями; определяется операция «подчёркивание»: $\underline{?m}=!m$ и $\underline{!m}=?m$. В алфавит композиции попадают те внешние действия из алфавита каждого LTS-операнда, для которых в алфавите другого LTS-операнда нет противоположных действий: $A\|B=(A\setminus\underline{B})\cup(B\setminus\underline{A})$. Композиционные переходы делятся на асинхронные и синхронные. Асинхронному переходу соответствует переход в одном из LTS-операндов, помеченный внешним действием композиционного алфавита, символом τ или γ; измениться может состояние только одного LTS-операнда. Синхронному переходу соответствует пара переходов в LTS-операндах по противоположным действиям, которая в композиции становится τ-переходом; измениться могут состояния обоих LTS-операндов. Формально композиция переходов определяется следующими правилами вывода:

$$z\in(A\setminus\underline{B})\cup\{\tau,\gamma\}\ \&\ a{-\!\!-}z{\to}a` \ \vdash\ ab{-\!\!-}z{\to}a`b;$$

$$z\in(B\setminus\underline{A})\cup\{\tau,\gamma\}\ \&\ b{-\!\!-}z{\to}b` \ \vdash\ ab{-\!\!-}z{\to}ab`;$$

$$z\in A\cap\underline{B}\ \&\ a{-\!\!-}z{\to}a`\ \&\ b{-\!\!-}\underline{z}{\to}b` \ \vdash\ ab{-\!\!-}\tau{\to}a`b`.$$

1.5. Проблема монотонности композиции и предлагаемое решение

Будем предполагать, что отношение конформности является предпорядком (рефлексивно и транзитивно). Пусть нам заданы спецификации компонентов. Определим свойства, которыми должна обладать «полная» спецификация композиционной системы:

1. Условие монотонности: композиция любых реализаций компонентов, конформных своим спецификациям, конформна спецификации системы.
2. Условие максимальности требований: спецификация системы предъявляет к системе самые сильные требования среди всех спецификаций, которые удовлетворяют условию монотонности, то есть конформна каждой из них.

Спецификацию, удовлетворяющую условию монотонности, будем называть *корректной* спецификацией. Заметим, что спецификация, разрешающая «всё», заведомо корректна. Если же корректная спецификация удовлетворяет условию максимальности требований, то будем называть её *косой* композицией спецификаций компонентов. Такую косую композицию будем обозначать $\mathbf{S_1} /\!/ \mathbf{S_2}$. Проблема заключается в том, что прямая композиция спецификаций компонентов, вообще говоря, не является не только косой композицией, но и вообще корректной спецификацией системы: она может не удовлетворять условию монотонности [18,10,11]. В частности, возможно нарушение безопасности: прямая композиция спецификаций безопасна, а композиция конформных реализаций опасна.

Основная идея предлагаемого решения проблемы заключается в том, чтобы косую композицию выполнять как прямую композицию преобразованных спецификаций. Преобразование $\mathcal{T}$: LTS→LTS, удовлетворяющее условию $\mathcal{T}(\mathbf{S_1}) \| \mathcal{T}(\mathbf{S_2}) = \mathbf{S_1} /\!/ \mathbf{S_2}$, будем называть *монотонным преобразованием*. Преобразованная спецификация должна быть эквивалентна исходной спецификации по отношению конформности: множество конформных реализаций сохраняется при преобразовании.

2. Отношение конформности $ioco_{\beta\gamma\delta}$

Рассматривается отношение конформности, основанное на минимальных тестовых возможностях по управлению реализацией (тестовые воздействия) и наблюдению её поведения. Тест может посылать в реализацию отдельный стимул и наблюдать либо его приём реализацией, либо отказ в приёме стимула. Также тест может принимать любую реакцию от реализации, наблюдая либо выдаваемую реализацией реакцию, либо отсутствие выдаваемых реакций. Отношение конформности формулируется в терминах трасс (последовательностей) наблюдений и складывается из двух условий: 1) гипотеза о безопасности, которая является предусловием тестирования и определяет класс реализаций, при тестировании которых не возникает запрещённых действий, и 2) условие, проверяемое при тестировании реализаций, удовлетворяющих гипотезе о безопасности.

2.1. Трассы наблюдений

Для того чтобы можно было проводить как статическую, так и динамическую верификацию, будем формализовать поведение системы как множество трасс (последовательностей) наблюдений, которые могут быть получены в тестовых экспериментах с системой. Спецификацию также будем понимать как множество трасс, а конформность как отношение между двумя множествами трасс. Поскольку состояния тестируемой системы, как правило, не наблюдаемы при тестировании, не будем рассматривать отношения конформности, основанные на состояниях, такие как изоморфизм и всевозможные симуляции: сильная (strong), слабая (weak), и т.п. В том случае, когда состояния реализации всё же открыты, будем считать, что имеется специальная реакция «сообщение о состоянии» (status message). Предполагается, что эта реакция «достоверна», то есть она выдаётся в том состоянии, о котором она сообщает, и сама не меняет этого состояния. Наблюдения таких специальных реакций, если они есть, также включаются в трассы наблюдений. Можно отметить, что тестирование реализаций с открытыми состояниями существенно упрощается, сводясь, фактически, к обходу графа переходов LTS [24,4, 5, 6, 20, 7].

Будем рассчитывать на минимальные тестовые возможности управления реализацией, которые сводятся к тестовым воздействиям двух типов: приём реакций от реализации и посылка стимула в реализацию. При этом будем считать, что тестер не может «выбирать», какую реакцию от реализации ему принять, а какую нет: либо принимается любая реакция, либо никакая не принимается. Стимул, напротив, всегда посылается один: нет совмещения посылки стимула с приёмом реакций или посылкой другого стимула.

Также будем рассчитывать на минимальные тестовые возможности наблюдения, которые сводятся к двум типам: наблюдение внешнего действия (выдача реализацией реакции или приём реализацией стимула) и отказ, понимаемый как отсутствие таких наблюдений. Последнее возможно в том случае, когда во взаимодействии возникает тупик: реализация ничего не может делать и *останавливается* (до следующего тестового воздействия, которое может этот тупик разрешить). Поскольку τ- и γ-действия всегда могут выполняться, тупик возникает только в стабильных состояниях. Для указанных выше типов тестовых воздействий у нас могут быть отказы двух типов:

a) *Стационарность* как отсутствие реакций (quiescent), когда тестер принимает реакции; обозначается символом δ [30,31]. Соответствующее состояние реализации называется стационарным.
b) *Блокировка стимула* (input refusal) как отказ в приёме данного стимула `?x`, посылаемого в реализацию тестером; обозначается как `{?x}` [28,29,22,2,8,16,14].

Стационарность может наблюдаться, если, например, время ожидания следующей реакции ограничено сверху известным тайм-аутом. Истечение тайм-аута означает, что реакций больше не будет до нового тестового воздействия (посылки стимула). Другой вариант: окружение получает специальное достоверное (безошибочное и, значит, не требующее тестирования) «извещение о прекращении передачи реакций». Блокировка стимула наблюдаема, если посылка стимула в реализацию предусматривает достоверное «уведомление о вручении», которое должно придти за время, ограниченное сверху известным тайм-аутом. Другой вариант: окружение достоверно получает либо «уведомление о вручении», либо «уведомление о невручении».

Наблюдение отказов возможно только, если в LTS нет дивергенции – бесконечной последовательности τ-действий, поскольку в этом случае система не выполняет никаких внешних действий и не останавливается. На каждое тестовое воздействие (наблюдение внешнего действия или отказа) желательно получение ответа, поэтому дивергенцию будем считать запрещенным действием и рассматривать как один из видов γ-действия.

γ-действие будем считать условно-наблюдаемым действием, и включать его в трассу. Как событие γ-действие может возникать при взаимодействии, но в дальнейшем ограничимся только таким тестированием, при котором γ-действия не будут выполняться. Тем самым, предполагается, что окружение должно взаимодействовать с реализацией правильно в том смысле, что такое взаимодействие не должно приводить к γ-действиям реализации. Если, например, интерфейс реализации содержит операции с предусловиями, то окружение не должно вызывать эти операции с нарушением предусловий: поведение реализации в этом случае считается нежелательным (неопределённым) и моделируется γ-действием. Иными словами, поскольку мы хотим убедиться в правильности реализации, а не окружения, нас не интересует поведение реализации при вызове её операций с нарушением предусловий, и такое поведение не регламентируется спецификацией. Семантика γ-действия предполагает, что после него любые наблюдения недостоверны, поэтому после γ-действия никакие наблюдения невозможны. Напомним, что наша цель – уметь строить безопасные системы из условно-безопасных компонентов, когда при взаимодействии компонентов γ-действия в них не выполняются.

Для получения всех трасс LTS достаточно в каждом стабильном состоянии добавить петлю, помеченную отказом, который в этом состоянии может возникать. Кроме того, каждый γ-переход перенаправим в добавленное терминальное γ-состояние, и для

моделирования дивергенции γ-действием из каждого дивергентного состояния (в котором начинается дивергенция) добавим γ-переход в γ-состояние. После этого нужно взять последовательности символов, кроме символа τ, которыми помечены все конечные цепочки переходов LTS, начинающиеся в начальном состоянии. Множество трасс LTS **S** будем обозначать как ***Traces***(**S**) и называть *трассовой моделью*. Заметим, что такие трассы, в которые, кроме внешних действий, входят отказы, являются разновидностью трасс с отказами (failure traces) [12,13]. Отличие в том, что мы разрешаем только два типа отказов: стационарность и блокировки стимулов.

2.2. Безопасные трассы и гипотеза о безопасности

Будем говорить, что тестирование безопасно, если оно не приводит к γ-действиям (включая дивергенцию). Определим это формально для LTS **S**. Заметим, что все определения безопасности существенно зависят только от трассовой модели ***Traces***(**S**), то есть LTS с одним множеством трасс эквивалентны с точки зрения безопасности.

Стимул ?x и его блокировка {?x} безопасны после трассы σ, если $\sigma ?x \gamma \notin$ ***Traces***(**S**). Все реакции и стационарность безопасны после трассы σ, если ни для какой реакции !y не существует трассы $\sigma !y \gamma \notin$ ***Traces***(**S**). Множество символов, безопасных в LTS **S** после трассы σ, обозначается как ***SafeSymbols***(**S** ***after*** σ). Заметим, что, если трасса $\sigma \notin$ ***Traces***(**S**), все символы безопасны после неё, а если $\sigma \in$ ***Traces***(**S**), то это не обязательно так. Кроме того, продолжение трассы $\sigma \in$ ***Traces***(**S**) безопасным символом может принадлежать или не принадлежать множеству трасс LTS. Трасса σ безопасна, если $\gamma \notin$ ***Traces***(**S**), $\sigma \in$ ***Traces***(**S**) и любой префикс трассы σ продолжается в ней символом, безопасным после этого префикса. Подмножество безопасных трасс LTS **S** будем обозначать как ***SafeTraces***(**S**). LTS безопасна, если все её трассы безопасны: **S** ***secure*** $=_{def}$ ***SafeTraces***(**S**)=***Traces***(**S**). Тестирование безопасно, если наблюдаются только безопасные трассы.

Поскольку тестируется неизвестная реализация, не может быть абсолютной уверенности в безопасности тестирования. Поэтому необходимо выдвинуть гипотезу о том, что реализация принадлежит некоторому классу *безопасно-тестируемых* реализаций. Такую гипотезу будем называть *гипотезой о безопасности*. Ограничимся такими отношениями конформности, которые проверяемы при безопасном тестировании, и поэтому желательно, чтобы класс конформных реализаций являлся подклассом класса безопасно-тестируемых реализаций. В этом случае имеется потенциальная возможность удостовериться, является ли любая реализация конформной или нет: если она не удовлетворяет гипотезе о безопасности, она неконформна, а если удовлетворяет, то проводим тестирование. Здесь отношение конформности складывается из двух условий: 1) гипотеза о безопасности, которая является предусловием тестирования, и 2) условие, проверяемое при тестировании реализаций, удовлетворяющих гипотезе о безопасности.

Безотносительно к спецификации реализация безопасно-тестируема только тогда, когда в ней вообще нет γ-действий. Хотя в этом случае любое тестирование безопасно, ни откуда не следует, что такой класс содержит все реализации, конформные данной спецификации. Выход в том, чтобы проводить тестирование не «вообще», а для проверки конформности заданной, известной спецификации. Это уже будет не любое тестирование. Например, нам не нужно сравнивать поведение реализации и спецификации после трассы, которой, согласно спецификации, не должно быть в конформной реализации. Фактически, здесь идёт речь о том, что тестирование проводится, вообще говоря, «до первой ошибки». Таким образом, класс безопасно-тестируемых реализаций зависит от спецификации.

Сформулируем предлагаемую гипотезу о безопасности для выбранных нами тестовых возможностей: реализация **I** безопасно-тестируема для заданной спецификации **S**, обозначается **I** ***safe for*** **S** тогда и только тогда, когда выполняются два условия:

1. $\gamma \notin \mathit{Traces}(\mathtt{S}) \Rightarrow \gamma \notin \mathit{Traces}(\mathtt{I})$.
2. $\forall \sigma \in \mathit{SafeTraces}(\mathtt{S})\ \forall u \in \mathit{SafeSymbols}(\mathtt{S}\ \mathit{after}\ \sigma)\ u \in \mathit{SafeSymbols}(\mathtt{I}\ \mathit{after}\ \sigma)$.

Короче это можно сформулировать, используя тестовые трассы. Трасса называется тестовой, если она безопасна или является продолжением безопасной трассы символом, безопасным после неё. Обозначая множество тестовых трасс LTS **S** как $\mathit{TestTraces}(\mathtt{S})$, имеем: $\mathtt{I}\ \mathit{safe\ for}\ \mathtt{S} \Leftrightarrow \mathit{TestTraces}(\mathtt{S}) \cap \mathit{Traces}(\mathtt{I}) \subseteq \mathit{TestTraces}(\mathtt{I})$. Тестовые трассы спецификации – это как раз те трассы, которые нужны для проверки конформности любой безопасно-тестируемой реализации.

2.3. Отношение конформности

Правила предлагаемого отношения конформности $\mathit{ioco}_{\beta\gamma\delta}$ применяются к безопасно-тестируемым реализациям и имеют вид "*Реализация может ... только тогда, когда для спецификации это возможно в такой же ситуации, то есть, после той же самой трассы, безопасной в спецификации*", где многоточие означает *символ, оставляющий трассу безопасной в спецификации*, то есть одно из следующих:

Правило реакции: "выдать данную реакцию".
Правило стационарности: "не выдавать никаких реакций".
Правило стимула: "принять данный стимул".
Правило блокировки: "блокировать данный стимул".

Теперь дадим формальное определение (заметим, что оно зависит только от трассовых моделей реализации и спецификации):

$\mathtt{I}\ \mathit{ioco}_{\beta\gamma\delta}\ \mathtt{S} =_{\mathtt{def}} \mathtt{I}\ \mathit{safe\ for}\ \mathtt{S}\ \&$
$\forall \sigma \in \mathit{SafeTraces}(\mathtt{S})\ \forall u \in \mathit{SafeSymbols}(\mathtt{S}\ \mathit{after}\ \sigma)\ \sigma u \in \mathit{Traces}(\mathtt{I}) \Rightarrow \sigma u \in \mathit{Traces}(\mathtt{S})$.

В терминах тестовых трасс проверяемое условие записывается короче:

$\mathit{TestTraces}(\mathtt{S}) \cap \mathit{Traces}(\mathtt{I}) \subseteq \mathit{Traces}(\mathtt{S})$. Соответственно:

$\mathtt{I}\ \mathit{ioco}_{\beta\gamma\delta}\ \mathtt{S} \Leftrightarrow \mathit{TestTraces}(\mathtt{S}) \cap \mathit{Traces}(\mathtt{I}) \subseteq \mathit{Traces}(\mathtt{S}) \cap \mathit{TestTraces}(\mathtt{I})$.

На Рис 1 приведены примеры спецификации и трёх реализаций. Первая реализация конформна спецификации, вторая не удовлетворяет гипотезе о безопасности, третья безопасно-тестируема, но не конформна.

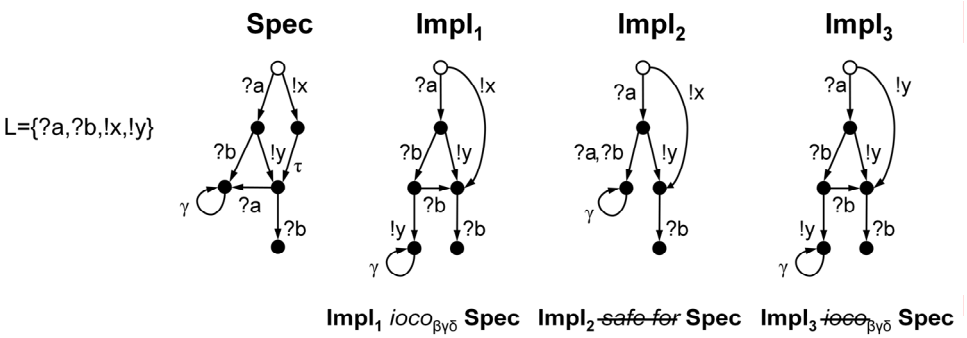

Рис 1. Конформные и неконформные реализации.

Если спецификация безопасна, конформность сводится к вложенности трасс. На Рис 2 приведены примеры безопасной реализации, которая конформна как безопасной, так и опасной спецификациям. Последняя, в силу рефлексивности, конформна сама себе.

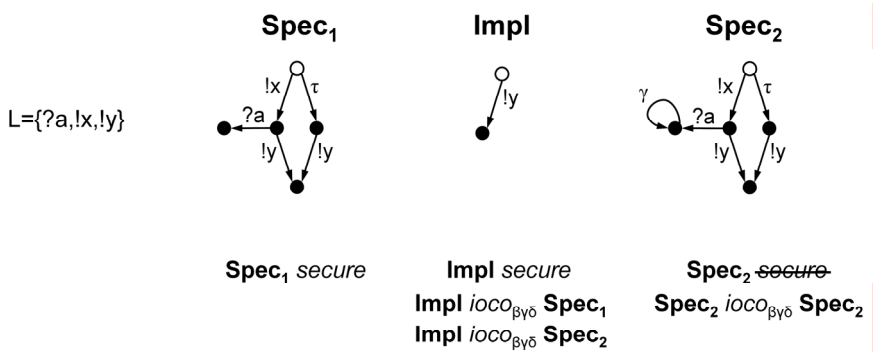

Рис 2. Безопасные и опасные реализации и спецификации.

3. Монотонное преобразование спецификации

Напомним, что целью монотонного преобразования является получение спецификации, эквивалентной исходной спецификации по множеству конформных реализаций, но удовлетворяющей условию монотонности: композиция любых реализаций компонентов, конформных своим спецификациям, конформна композиции преобразованных спецификаций.

3.1. Возможность построения безопасной системы из опасных компонентов

Для проверки безопасности системы, построенной из компонентов, конформных своим спецификациям, нам нужно: 1) построить косую композицию спецификаций компонентов, 2) проверить её безопасность (недостижимость γ-действий). Прежде всего, продемонстрируем возможность этого на простом примере (**Ошибка! Источник ссылки не найден.**). Здесь косая композиция опасных спецификаций совпадает с их прямой композицией и сама безопасна.

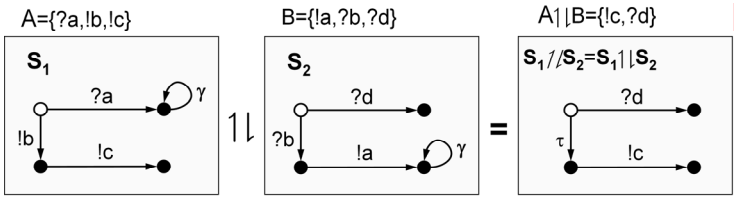

Рис 3. Безопасная система, построенная из опасных компонентов

3.2. Существование монотонного преобразования

Монотонных преобразований LTS-спецификаций может быть много. Сначала покажем, что такое преобразование существует. Определим объединение множества LTS как новую LTS, которая строится добавлением нового начального состояния и проведением из него τ-переходов во все начальные состояния объединяемых LTS (Рис 4). Трассовая модель объединения LTS совпадает с объединением трассовых моделей объединяемых LTS.

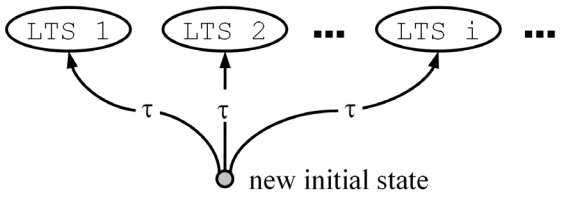

Рис 4. Объединение LTS.

Нетрудно показать, что объединение реализаций, конформных заданной спецификации, является её монотонным преобразованием. Конечно, такое определение преобразования неконструктивно и, как правило, неалгоритмизуемо. В частности,

конформных реализаций может быть несчётное множество. Кроме того, объединение конформных реализаций содержит много «лишнего», без чего можно обойтись в монотонно преобразованной спецификации. Поэтому далее дадим конструктивное определение другого монотонного преобразования, которое при некоторых ограничениях на LTS-спецификацию может быть алгоритмизуемо.

3.3. Причины несохранения конформности и нарушения безопасности

Сначала рассмотрим примеры несохранения конформности и нарушения безопасности при композиции. Выделим четыре причины этого и четыре аспекта предлагаемого монотонного преобразования спецификаций.

3.3.1. Полное стабильное состояние

Полное стабильное состояние – это стабильное состояние, в котором определены переходы по всем стимулам и, по крайней мере, по одной реакции, то есть в нём нет блокировок стимулов и стационарности. В примере на Рис 5 у первого компонента реализация **I** конформна спецификации **S**. Они различаются тем, что в спецификации начальное состояние нестабильное, а в реализации полное стабильное. У второго компонента спецификация и реализация совпадают – это LTS **T**, которая (по рефлексивности отношения) конформна сама себе. У прямой композиции спецификаций **S**⫴**T** начальное состояние нестабильно, а у прямой композиции реализаций **I**⫴**T** – неполное стабильное. Из-за этого в композиции реализаций есть трасса `{?a}!x`, которой нет в композиции спецификаций. Поскольку γ-действие в композиции спецификаций отсутствует, конформность эквивалентна вложенности трасс. Но в этом примере конформность не сохраняется при прямой композиции спецификаций.

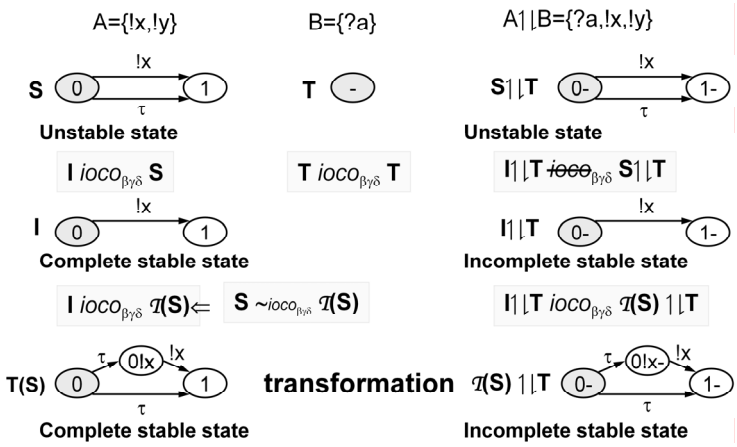

Рис 5. Полное стабильное состояние.

Как сохранить конформность? Рассмотрим преобразование, создающее в спецификации **S** полное стабильное состояние. Преобразованная спецификация $\mathcal{T}(\mathbf{S})$ эквивалентна исходной спецификации **S**, следовательно, **I** $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\mathcal{T}(\mathbf{S})$. Но теперь в композиции спецификаций $\mathcal{T}(\mathbf{S})$⫴**T** появилось неполное стабильное состояние, как в реализации **I**⫴**T**, и конформность сохраняется.

3.3.2. Сингулярность

Сингулярным состоянием называем стабильное состояние, из которого выходит не более одного перехода, помеченного реакцией. В примере на Рис 6 у первого компонента реализация **I** конформна спецификации **S**. Они различаются тем, что в спецификации начальное состояние несингулярно, а в реализации – сингулярно. У второго компонента спецификация и реализация совпадают – это LTS **T**. Как и в предыдущем примере у

прямой композиции спецификаций **S⫴T** начальное состояние нестабильно, а у прямой композиции реализаций **I⫴T** – неполное стабильное. Из-за этого в композиции реализаций есть трасса {?a}!x, которой нет в композиции спецификаций. Поскольку γ-действия в композиции спецификаций отсутствуют, в этом примере также конформность не сохраняется при прямой композиции спецификаций.

Для сохранения конформности нужно преобразовать спецификацию, «расщепив» несингулярное состояние на несколько (по числу выходящих реакций) сингулярных состояний. В данном примере несингулярное состояние 0 расщепляется на два сингулярных состояния 0!x и 0!y, из которых ведутся переходы только по одной реакции !x и !y, соответственно. Преобразованная спецификация $\mathcal{T}$(**S**) эквивалентна исходной **S**, но теперь в композиции спецификаций $\mathcal{T}$(**S**)⫴**T** появилось неполное стабильное состояние, как в реализации **I⫴T**, и конформность сохраняется.

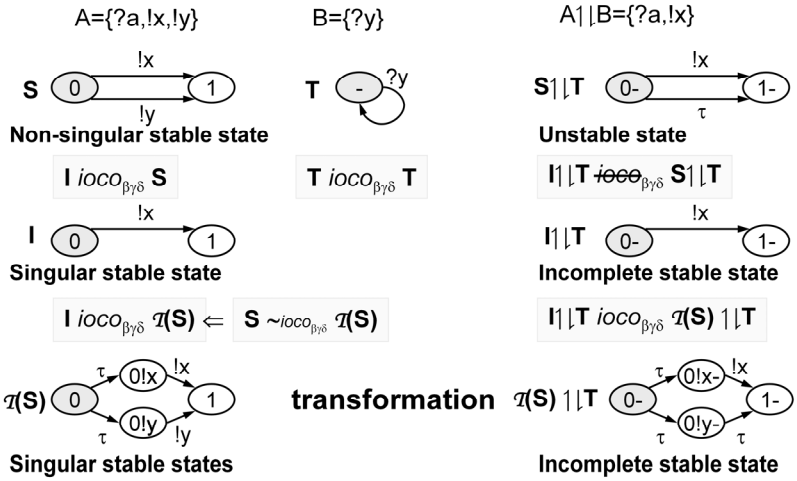

Рис 6. Сингулярность.

3.3.3. Экспонентная LTS

Еще одним аспектом преобразования является преобразование, которое называется *экспоненцированием*. Оно похоже на известное преобразование детерминизации конечного автомата [15] с тем, однако, отличием, что у нас сохраняются τ-переходы. Результат такого преобразования будем называть *экспонентной* LTS. Состояния экспонентной LTS (*экспонентные состояния*) соответствуют трассам исходной LTS **S** и являются множествами состояний после этих трасс. Для трассы σ∈***Traces***(**S**) соответствующее экспонентное состояние будем обозначать [σ]. Начальное экспонентное состояние [ϵ] соответствует пустой трассе. В экспонентной LTS проводится переход [σ]—z→[σz], если z≠τ и σz∈***Traces***(**S**). Здесь [σz] совпадает с множество концов b всех переходов вида a—z→b, где a пробегает множество A. Внутренний переход [σ]—τ→[σρ] проводится, если трасса ρ состоит только из отказов и σρ∈***Traces***(**S**). Здесь [σρ] состоит из тех состояний s∈[σ], которые стабильны и порождают все отказы из ρ. Нетрудно показать, что экспонентная LTS эквивалентна исходной по отношению конформности.

На Рис 7 приведён пример частичного экспоненцирования и последующей сингуляризации. Добавляются экспонентные состояния только пустой трассы ϵ и всех трасс отказов. Кроме того, из экспонентного состояния [ϵ] проведены только τ-переходы. В этом примере этого достаточно для монотонности преобразования.

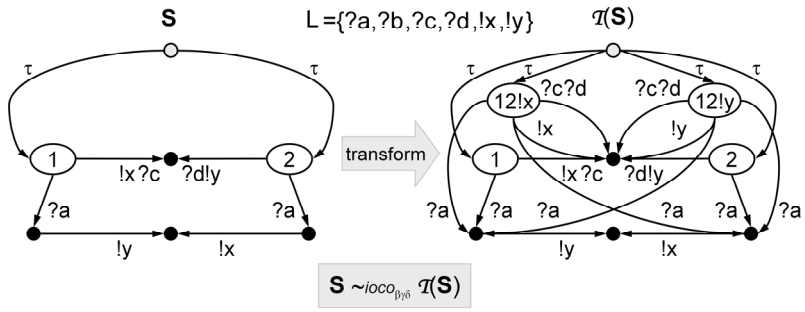

Рис 7. Экспоненцирование.

Покажем необходимость всех экспонентных состояний: `[{?d}]={1}`, `[{?c}]={2}` и `[{?b}]={12}`, расщепляемого на два сингулярных состояния `{12!x}` и `{12!y}`.

Необходимость сохранения состояния `{1}` показано на Рис 8. Поскольку **S** ***ioco***$_{\beta\gamma\delta}$ $\mathcal{T}$(**S**) и **T** ***ioco***$_{\beta\gamma\delta}$ **T**, должно быть **S**⫛**T** ***ioco***$_{\beta\gamma\delta}$ $\mathcal{T}$(**S**)⫛**T**. Поскольку в спецификации γ-действий нет, должна быть вложенность трасс ***Traces***(**S**⫛**T**)⊆***Traces***($\mathcal{T}$(**S**)⫛**T**). Однако в композиции **S**⫛**T** имеется трасса `{?b}?a!y`, которой не было бы в композиции $\mathcal{T}$(**S**)⫛**T**, если бы там не было состояния `{1}`. Аналогично показывается необходимость сохранения состояния `{2}`.

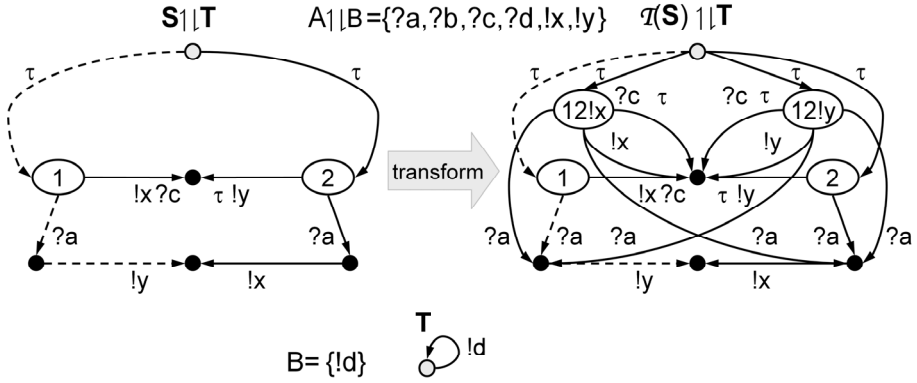

Рис 8. Экспоненцирование: необходимость сохранения состояния `{1}`.

Необходимость добавления состояния `{12!x}` показано на Рис 9. Поскольку **I** ***ioco***$_{\beta\gamma\delta}$ **S** и **T** ***ioco***$_{\beta\gamma\delta}$ **T**, должно быть **I**⫛**T** ***ioco***$_{\beta\gamma\delta}$ $\mathcal{T}$(**S**)⫛**T**. Заметим, что **I**⫛**T** ¬***ioco***$_{\beta\gamma\delta}$ **S**⫛**T**, то есть без преобразования конформность не сохраняется. Поскольку в спецификации γ-действий нет, должна быть вложенность трасс. Однако в композиции **I**⫛**T** имеется трасса `{?b}?a!x`, которой нет в композиции **S**⫛**T**. В композиции $\mathcal{T}$(**S**)⫛**T** эта трасса есть, но без состояния `{12!x}` её бы не было. Аналогично показывается необходимость сохранения состояния `{12!y}`.

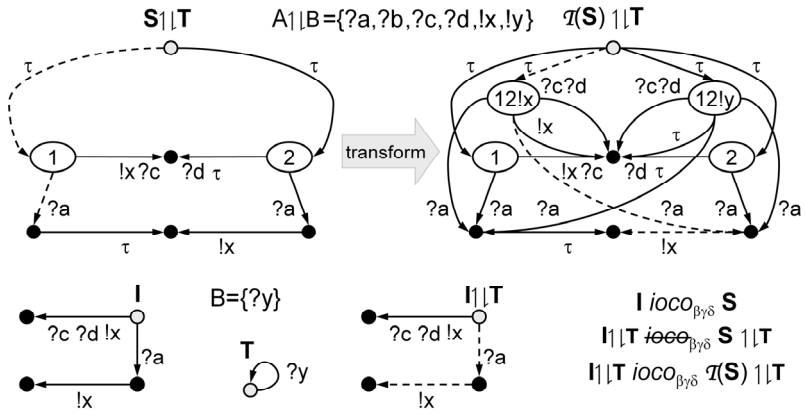

Рис 9. Экспоненцирование: необходимость добавления состояния {12!x}.

3.3.4. γ-однородность

Состояние называется γ-однородным, если после всех реакций следует γ-действие или все реакции безопасны. На Рис 10 приведён пример, когда начальное состояние спецификации **S** не γ-однородно: переход по реакции !x приводит к γ-действию, а перехода по реакции !y нет. В конформной реализации **I**, наоборот, нет перехода по реакции !x, а переход по реакции !y приводит к γ-действию. Из-за этого композиция спецификаций **S⫴T** безопасна, а композиция конформных реализаций **I⫴T** опасна, что означает и неконформность.

Чтобы сохранить конформность, нужно преобразовать спецификацию, сделав все её состояния γ-однородными. Если в состоянии есть переход по реакции, приводящий к γ-действию, добавляем переходы из этого состояния по всем реакциям, ведущие к γ-действию. Будем считать, что все переходы по опасным действиям (действиям, после которых возможно γ-действие) ведут в специальное состояние **Γ**, в котором определена только γ-петля.

Заметим, что при сочетании сингуляризации и γ-однородности из каждого стабильного состояния ведёт либо только один переход по безопасной реакции, либо переходы по всем опасным реакциям в состояние **Γ**, либо ни одного перехода реакциям.

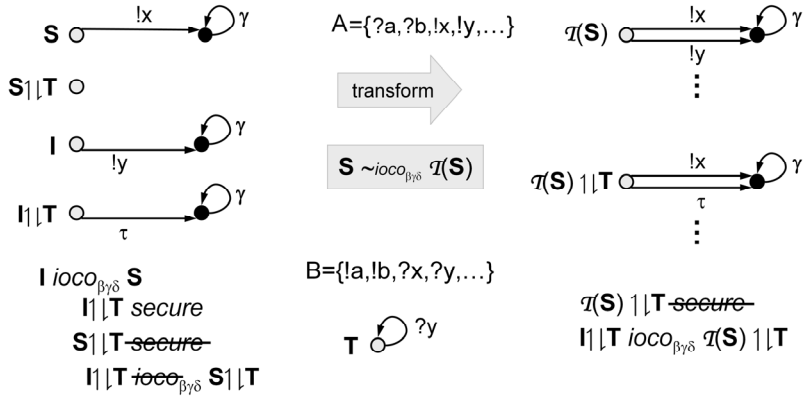

Рис 10. γ-однородность

3.4. Формальное определение монотонного преобразования

Теперь можно формально определить предлагаемое монотонное преобразование спецификации. Пусть задана LTS-спецификация **S** в алфавите L. Состояния преобразованной LTS 𝒯(**S**) строятся на основе экспонентных состояний. Начальным является экспонентное состояние [ϵ], соответствующее пустой трассе. Переходы

определяются правилами вывода в таблице, где T=*Traces*(s), S=*SafeTraces*(s), μ – трасса, не заканчивающаяся на отказ, и ρ – непустая трасса отказов.

1)	$\vdash \Gamma\text{—}\gamma\rightarrow\Gamma$;
2) $\mu!x\in S$	$\vdash [\mu]\text{—}!x\rightarrow[\mu!x]$;
3) $\exists!y\ \mu!y\gamma\in T$	$\vdash [\mu]\text{—}!x\rightarrow\Gamma$;
4) $\mu?a\in S$	$\vdash [\mu]\text{—}?a\rightarrow[\mu?a]$;
5) $\mu?a\gamma\in T$	$\vdash [\mu]\text{—}?a\rightarrow\Gamma$;
6) $\forall?b\in L\ \mu?b\in T$ & $\mu!x\in S$	$\vdash [\mu]\text{—}\tau\rightarrow[\mu]!x\text{—}!x\rightarrow[\mu!x]$;
7) —"— —"— —"— —"— —"— & $\mu?a\in S$	$\vdash [\mu]!x\text{—}?a\rightarrow[\mu?a]$;
8) —"— —"— —"— —"— —"— & $\mu?a\gamma\in T$	$\vdash [\mu]!x\text{—}?a\rightarrow\Gamma$;
9) $\forall?b\in L\ \mu?b\in T$ & $\exists!y\ \mu!y\gamma\in T$	$\vdash [\mu]\text{—}\tau\rightarrow[\mu]\gamma\text{—}!x\rightarrow\Gamma$;
10) —"— —"— —"— —"— —"— & $\mu?a\in S$	$\vdash [\mu]\gamma\text{—}?a\rightarrow[\mu?a]$;
11) —"— —"— —"— —"— —"— & $\mu?a\gamma\in T$	$\vdash [\mu]\gamma\text{—}?a\rightarrow\Gamma$;
12) $\mu\rho!x\in S$	$\vdash [\mu]\text{—}\tau\rightarrow[\mu\rho]!x\text{—}!x\rightarrow[\mu\rho!x]$;
13) —"— —"— —"— —"— —"— & $\mu\rho?a\in S$	$\vdash [\mu\rho]!x\text{—}?a\rightarrow[\mu\rho?a]$;
14) —"— —"— —"— —"— —"— & $\mu\rho?a\gamma\in T$	$\vdash [\mu\rho]!x\text{—}?a\rightarrow\Gamma$;
15) $\exists!y\ \mu\rho!y\gamma\in T$	$\vdash [\mu]\text{—}\tau\rightarrow[\mu\rho]\gamma\text{—}!x\rightarrow\Gamma$;
16) —"— —"— —"— —"— —"— & $\mu\rho?a\in S$	$\vdash [\mu\rho]\gamma\text{—}?a\rightarrow[\mu\rho?a]$;
17) —"— —"— —"— —"— —"— & $\mu\rho?a\gamma\in T$	$\vdash [\mu\rho]\gamma\text{—}?a\rightarrow\Gamma$;
18) $\forall!y\in L\ \mu!y\notin T$ & $\mu\rho\in S$	$\vdash [\mu]\text{—}\tau\rightarrow[\mu\rho]\delta$;
19) —"— —"— —"— —"— —"— & $\mu\rho?a\in S$	$\vdash [\mu\rho]\delta\text{—}?a\rightarrow[\mu\rho?a]$;
20) —"— —"— —"— —"— —"— & $\mu\rho?a\gamma\in T$	$\vdash [\mu\rho]\delta\text{—}?a\rightarrow\Gamma$.

Поясним эти правила вывода.

Правило 1 описывает специальное состояние Γ, в котором определяется только γ-петля. В это состояние будут вести все переходы по опасным стимулам и реакциям, как это делалось для получения γ-однородной LTS.

Правила 2÷5 описывают переходы по стимулам и реакциям из нестабильного экспонентного состояния $[\mu]$ для безопасной трассы μ.

Правила 6÷8 описывают случай, когда реакции безопасны после безопасной трассы μ. Определяются τ-переходы во все сингулярные полные стабильные состояния и переходы из них по стимулам и реакциям. Каждое такое состояние имеет вид $[\mu]!x$, и из него определяется переход только по одной реакции $!x$.

Правила 9÷11 описывают случай, когда реакции опасны после безопасной трассы μ. Определяются τ-переходы во все γ-однородные полные стабильные состояния и переходы из них по стимулам и реакциям. Каждое такое состояние имеет вид $[\mu]\gamma$, и из него определяются переходы по всем реакциям в состояние Γ.

Правила 12÷14 описывают случай, когда безопасная трасса μ безопасно продолжается непустой трассой отказов ρ, и реакции безопасны после трассы $\mu\rho$. Определяются τ-переходы во все сингулярные неполные стабильные состояния и переходы из них по стимулам и реакциям. Каждое такое состояние имеет вид $[\mu\rho]!x$, и из него определяется переход только по одной реакции $!x$.

Правила 15÷17 описывают случай, когда безопасная трасса μ безопасно продолжается непустой трассой отказов ρ, а реакции опасны после трассы $\mu\rho$. Определяются τ-переходы во все γ-однородные неполные стабильные состояния и переходы из них по стимулам и реакциям. Каждое такое состояние имеет вид $[\mu\rho]\gamma$, и из него определяются переходы по всем реакциям в состояние Γ.

Правила 18÷20 описывают случай, когда безопасная трасса μ безопасно продолжается непустой трассой отказов ρ, а $\mu\rho$ не продолжается реакциями в спецификации. Определяются τ-переходы во все стационарные (неполные) стабильные состояния и переходы из них по стимулам. Каждое такое состояние имеет вид $[\mu\rho]\delta$.

За отсутствием места формальное доказательство монотонности этого преобразования опускается.

4. Монотонное преобразование в частных случаях

4.1. Системы без блокировок стимулов

Можно показать, что для важного частного случая спецификаций без блокировок стимулов монотонное преобразование можно существенно упростить. Для этого достаточно только γ-однородности. При отсутствии γ-действий монотонно тождественное преобразование.

При отсутствии блокировок в спецификациях компонентов, если косая композиция безопасна, то она совпадает с прямой композицией спецификаций. Отсюда следует, что для систем без блокировок косая композиция нужна только для проверки безопасности. Безопасность может быть нарушена только из-за дивергенции, которую считается нежелательным поведением и моделируется γ-действием. Такая дивергенция в композиции может порождаться бесконечной цепочкой синхронных переходов. Поэтому проверка безопасности, фактически, сводится к проверке дивергенции в композиции LTS.

Заметим, что для спецификаций без блокировок наше отношение $\boldsymbol{ioco}_{\beta\gamma\delta}$ совпадает с известным отношением ***ioco***, которое строится на трассах без блокировок и без γ-действий [30]. Отличие лишь в том, что для отношения ***ioco*** требуется отсутствие блокировок в каждом состоянии реализации, а для отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ – только в тех состояниях, которые достижимы по трассам, безопасным в спецификации.

Также заметим, что отношение ***ioco*** формально определяется для LTS-спецификаций, в которых могут быть блокировки стимулов, но эти блокировки игнорируются. Из-за этого прямая композиция LTS-спецификаций оказывается, вообще говоря, некорректной и для отношения ***ioco***. Вот почему стараются выполнить, так называемое, пополнение LTS-спецификаций [9,24], то есть такое преобразование, которое сохраняет (или почти сохраняет) множество конформных реализаций, но удаляет все блокировки. После такого пополнения (при отсутствии γ-действий) дальнейшее преобразование не требуется для монотонности, то есть само такое пополнение является монотонным преобразованием [11].

Если нет γ-действий, то может оказаться, что трасса в исходной спецификации не продолжалась некоторым стимулом. Это приводит к тому, что для отношения ***ioco*** вообще не проверяется поведение реализации по поводу этого стимула. Формально конформная реализация может этот стимул блокировать или принимать с любым дальнейшим поведением [11]. Фактически речь идёт о том, что поведение реализации по поводу этого стимула не определено [9,24]. При нашем подходе, когда допускаются γ-действия, это легко моделируется продолжением такой трассы стимулом и далее γ-действием. В результате получится пополненная спецификация с γ-действиями, которую для монотонности нужно дополнительно привести к γ-однородности.

4.2. Системы без γ-действий

Для систем без γ-действий (но с возможными блокировками) монотонное преобразование не намного проще, чем в общем случае. Отсутствует лишь требование γ-однородности. Однако класс таких LTS-систем имеет тот существенный недостаток, что он незамкнут по композиции. Дело в том, что при композиции может возникать

дивергенция, порождаемая бесконечной цепочкой синхронных переходов. Поскольку дивергенция рассматривается как нежелательное поведение и моделируется γ-действием, такие γ-действия могут появляться при композиции и в том случае, когда LTS-операнды не имели γ-действий.

5. Алгоритмические проблемы

За отсутствием места не будем рассматривать подробно вопросы алгоритмизации преобразования и последующей прямой композиции. Более точно, речь идёт о тех ограничениях, которые нужно наложить на спецификацию, чтобы преобразование и последующая прямая композиция стали алгоритмическими. Разумеется, и сама LTS-спецификация должна быть задана алгоритмическим способом, то есть алгоритмами, определяющими переходы из каждого состояния LTS. Здесь лишь вкратце укажем на ряд ключевых моментов.

1) Множество состояний после безопасной трассы должно быть конечным. Это необходимо для того, чтобы алгоритмически строить экспонентные состояния и переходы из них. Для этого нужно ограничить степень ветвления LTS. В общем, достаточно потребовать конечности ветвления: в каждом состоянии (достижимом по безопасной трассе) должно быть определено конечное число переходов [21].
2) Поскольку дивергенция моделируется разрушением, нужно уметь алгоритмически распознавать дивергенцию. Такая дивергенция всегда распознаваема, если она является бесконечной цепочкой τ-переходов, порождённой τ-циклом в LTS. С учётом γ-действий достаточно наложить такие ограничения, чтобы при построении τ-замыкания состояния (множества состояний, достижимых из данного состояния по τ-переходам) гарантированно и за конечное время обнаруживался либо τ-цикл, либо γ-переход.
3) Дивергенция в композиционной LTS может быть не только унаследованной от дивергенции от одного из LTS-операндов, но и порождаться бесконечной цепочкой синхронных переходов. Нужно наложить на LTS-операнды такие ограничения, чтобы такая дивергенция также обнаруживалась за конечное время. В общем, эти ограничения сводятся к некоторой «регулярности» LTS-операндов: каждая бесконечная трасса без γ-ответвлений должна порождаться «регулярной» цепочкой переходов, то есть такой цепочкой, которая «укладывается» в конечную под-LTS (проходит через конечное число состояний).

6. Заключение

В данной работе формализуется понятие запрещённого поведения, что позволяет формально определить безопасные системы и условно-безопасные компоненты таких систем. Также вводится понятие косой композиции спецификаций, которая: 1) корректна, то есть удовлетворяет условию монотонности (конформность сохраняется при композиции), и 2) является самой «сильной» (предъявляющей максимальные требования) среди всех корректных спецификаций. При заданных спецификациях компонентов косая композиция этих спецификаций является точной спецификацией системы и может использоваться при использовании системы внешними клиентами.

В данной работе показано, что при заданных спецификациях компонентов можно получить косую композицию спецификаций как прямую композицию соответствующим образом преобразованных спецификаций.

Это позволяет решить следующие важнейшие проблемы.

1. Если заданы не только спецификации компонентов, но и спецификация системы, можно верифицировать декомпозицию требований (согласованность спецификаций системы и её компонентов). Декомпозиция корректна, если косая композиция конформна заданной спецификации системы.

2. При достаточной полноте тестирования компонентов снимается проблема тестирования системы (на конформность косой композиции и, тем более, заданной спецификации системы, если декомпозиция требований правильна).
3. Если нет уверенности в полноте тестирования компонентов, косая композиция может использоваться для генерации системных тестов.

Данная работа была выполнена в рамках развития технологии тестирования UniTESK [19,20]. Отдельные части этой работы выполнялись как части исследований, финансируемых грантами РФФИ и контрактами с Президиумом РАН. Результаты докладывались на нескольких конференциях [2,3,8].

Список литературы

[1] G. Bernot. Testing against formal specifications: A theoretical view. In S. Abramsky and T.S.E. Maibaum, editors, TAPSOFT'91, Volume 2, pp. 99-119. Lecture Notes in Computer Science 494, Springer-Verlag, 1991.

[2] Бурдонов И. Б., Косачев А. С. «Тестирование компонентов распределенной системы.» Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005, стр.63-65.

[3] Бурдонов И. Б., Косачев А. С. «Верификация композиции распределенной системы.» Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005, стр.67-69.

[4] И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. Использование конечных автоматов для тестирования программ. "Программирование". 2000. No. 2. стр.12-28.

[5] И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. «Асинхронные автоматы: классификация и тестирование.» Труды ИСП РАН, т. 4, 2003, с. 7-84.

[6] И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированнный случай. "Программирование". 2003. No. 5.

[7] И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированнный случай. "Программирование". 2004. No. 1.

[8] I. Bourdonov, A. Kossatchev, and V. Kuliamin. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proc. of MBT 2006, Vienna, Austria, March 2006.

[9] Gregor V. Bochmann , Alexandre Petrenko. Protocol testing: review of methods and relevance for software testing, Proceedings of the 1994 international symposium on Software testing and analysis, pp.109-124, August 17-19, 1994, Seattle, Washington, United States.

[10] M. van der Bijl, A. Rensink, J. Tretmans. Component Based Testing with ioco. CTIT Technical Report TR-CTIT-03-34, University of Twente, 2003.

[11] Machiel van der Bijl, Arend Rensink, Jan Tretmans. Compositional testing with ioco. In 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.

[12] van Glabbeek, R. The linear time – branching time spectrum I; the semantics of concrete, sequential processes, in: Handbook of Process Algebra (J.A. Bergstra, A. Ponse and S.A. Smolka, eds.), 2001, Chapter 1, Elsevier, pp 3-99.

[13] 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.

[14] L. Heerink. Ins and Outs in Refusal Testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1998.

[15] Д. Хопкрофт, Р. Мотвани, Д. Ульман. Введение в теорию автоматов, языков и вычислений, 2-е изд.. М., «Вильямс», 2002.
[16] L. Heerink, J. Tretmans. *Refusal Testing for Classes of Transition Systems with inputs and Outputs*. In T. Mizuno, N. Shiratori, T. Higashino, A. Togashi, eds. Formal Description Techniques and Protocol Specification, Testing and Verification. Chapman & Hill, 1997.
[17] Revised Working Draft on "Framework: Formal Methods in Conformance Testing", JTC1/SC21/WG1/Project 54/1 // ISO Interim Meeting / ITU-T on, Paris, 1995.
[18] C. Jard, T. Jéron, L. Tanguy, C. Viho. 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.
[19] Victor V. Kuliamin, Alexander S.Kossatchev, Alexander K. Petrenko, Nick V. Pakoulin, Igor B. Bourdonov. Integration of Functional and Timed Testing of Real-Time and Concurrent Systems. Perspectives of System Informatics // LNCS. No. 2890. 2003. pp. 450-461.
[20] В.В.Кулямин, А.К.Петренко, А.С.Косачев, И.Б.Бурдонов. Подход UniTesK к разработке тестов. "Программирование", 2003, №6, стр.25-43. V.V. Kuliamin, A.K. Petrenko, A.S. Kossatchev, and I.B. Burdonov. The UniTesK Approach to Designing Test Suites. Programming and Computer Software, 2003, 6, p.310.
[21] D.König. Über eine Schlusswaise aus dem Endlichen ins Unendliche. Acta Litt. Ac. Sci. Hung. Fran. Josep. No 3. 1927. pp. 121-130. См. также К.Куратовский, А.Мостовский. Теория множеств. М.«Мир», 1970.
[22] G. Lestiennes, M.-C. Gaudel. Test de systemes reactifs non receptifs. Journal Europeen des Systemes Automatises, Modelisation des Systemes Reactifs, pp. 255–270. Hermes, 2005.
[23] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Proceedings of the 6th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 137-151, August 1987.
[24] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines—A Survey, Proceedings of the IEEE 84, No. 8, 1090–1123, 1996.
[25] R. Milner. A Calculus of Communicating Systems, LNCS, vol. 92, Springer-Verlag, 1980.
[26] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.
[27] M. Phalippou. Relations d`Implantation et Hypotheses de Test sur des Automates a Entrees et Sorties. PhD thesis, L`Universite de Bordeaux I, France, 1994.
[28] Segala, R. Quiescence, fairness, testing and the notation of implementation (extended abstract). In LNCS, volume 715. Proceedings of the 4th International Conference on Concurrency Theory (CONCUR`93), Hildesheim, Germany, 1993.
[29] J. Tretmans. A Formal Approach to Conformance Testing. PhD. Thesis, University of Twente, Enschede, The Netherlands, 1992.
[30] Tretmans, J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.
[31] F. Vaandrager. 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.