Зависимости между ошибками на классах тестируемых реализаций

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

Аннотация. Статья посвящена проблеме зависимости между ошибками, определяемыми спецификацией, и связанной с ней проблеме оптимизации тестов. Между ошибками имеется зависимость, если существует такое строгое подмножество ошибок, что любая неконформная реализация (то есть реализация, в которой есть какая-нибудь ошибка) содержит ошибку из этого подмножества. Соответственно, достаточно, чтобы тесты обнаруживали ошибки только из этого подмножества. Предлагается формальная модель тестового взаимодействия самого общего вида и конформность типа редукции, для которых зависимость между ошибками практически отсутствует. Показывается, что многие известные конформности в различных семантиках взаимодействия являются частными случаями этой общей модели. В этой общей модели зависимость между ошибками может возникать, когда в качестве класса тестируемых реализаций выбирается то или иное строгое подмножество класса всех реализаций. Частные семантики взаимодействия и/или различные гипотезы о реализации (в частности, гипотезы о безопасности) как раз и предполагают, что тестируемая реализация не любая, а относится к некоторому подклассу (безопасных) реализаций.

Ключевые слова: Семантика взаимодействия, трассы, LTS, конформность, зависимость между ошибками, генерация тестов, дивергенция, разрушение, безопасное тестирование.

1. Введение

Правильность исследуемой системы в самом широком смысле понимается как её соответствие заданным требованиям. Для верификации (проверки) этого соответствия с помощью формальных методов, объекты и отношения реального мира отображаются в модельные, математические объекты и отношения. Модель исследуемой системы называется реализацией, модель требований – спецификацией, а соответствие требованиям отображается в модельную конформность. Последняя понимается как обычное математическое соответствие, то есть подмножество декартового произведения множеств реализаций и спецификаций.

Спецификация и конформность считаются заданными. Что касается реализации как модели исследуемой системы, то *тестовая гипотеза*

предполагает, что такая модель существует для каждой исследуемой системы [[8]].

Если реализация, как модель исследуемой системы, известна, то возможна статическая (аналитическая) верификация, которая сводится к проверке того, что пара формальных моделей <реализация, спецификация> принадлежит допустимому множеству таких пар, определяемому отношением конформности.

Что делать, если реализация неизвестна (или её слишком сложно построить по исследуемой системе)? В этом случае возникает необходимость в тестировании, которое понимается как динамическая верификация конформности, то есть её проверка в процессе экспериментов. Конечно, для того, чтобы тестирование было возможным, сама конформность должна быть выражена в терминах взаимодействия реализации с окружающей средой. Тест взаимодействует с системой, подменяя собой окружение.

В данной работе мы будем рассматривать не любое тестирование, а только такое, которое основано на трёх предположениях.

<u>Первое предположение</u>. Мы будем рассматривать только *дискретное* тестовое взаимодействие, которое сводится к последовательности дискретных событий двух видов: тестовых воздействий на реализацию и наблюдений над поведением реализации. Эту последовательность мы будем называть *трассой*. Отметим, что в общем случае не всякое поведение реализации может наблюдаться в тестовом эксперименте, то есть в реализации могут происходить события, которые не наблюдаемы и, тем самым, не различимы между собой и не входят в трассу. Такое ненаблюдаемое поведение реализации традиционно обозначается символом τ и называется τ-активностью.

Дискретное взаимодействие моделируется с помощью, так называемой, машины тестирования, внутри которой находится реализация. Тестовому воздействию соответствует нажатие той или иной кнопки на клавиатуре машины, а наблюдению – появление его символа на экране дисплея. Таким образом, трасса – это последовательность кнопок и наблюдений. Тест понимается как инструкция оператору машины, в которой указывается, что оператор должен делать после той или иной трассы: нажимать кнопки (и какие кнопки) и/или ждать наблюдений.

Всё, что мы можем узнать о реализации с помощью тестирования, – это множество её трасс. Наблюдение в эксперименте некоторой трассы предполагает, что все её префиксы наблюдались в этом же эксперименте в более ранние моменты времени. Поэтому множество трасс реализации префикс-замкнуто. Тестовый эксперимент может быть пустым: ни одна кнопка не нажимается, и наблюдения не ожидаются. В этом случае естественно считать, что наблюдается пустая трасса. Тем самым, в каждой реализации есть пустая трасса, то есть множество трасс реализации не пусто.

Поскольку тестовое воздействие (нажатие кнопки) не зависит от самой реализации (оператор в любой момент времени может нажать любую кнопку), продолжение трассы реализации кнопкой также даёт трассу реализации. Это означает, что множество трасс реализации вместе с каждой трассой σ содержит и все трассы вида σρ, где ρ – последовательность кнопок.

Спецификацию теперь можно понимать как описание того, какие множества трасс реализаций правильные (конформные), а какие нет. В общем случае, если реализация имеет множество трасс f, то конформна или неконформна не каждая отдельная трасса σ∈f, а всё множество трасс f целиком.

Тест (как инструкция оператору) также задаётся непустым префикс-замкнутым множеством трасс. Тестовый эксперимент – это прогон теста, который заканчивается, когда получается максимальная в тесте трасса или «ответвление в сторону», то есть после некоторой трассы теста получается наблюдение, которым эта трасса не продолжается в тесте. Результатом прогона теста является трасса реализации σ∈f. Различные прогоны одного и того же теста могут давать разные результаты, если реализация и/или тест недетерминированы.

Тест недетерминирован, если после какой-то трассы теста недетерминировано поведение оператора машины тестирования: он может как ждать наблюдения, так и нажимать кнопки, или он может только нажимать кнопки, но таких кнопок несколько. Иными словами, тест недетерминирован, если некоторая его немаксимальная трасса продолжается в тесте и наблюдениями и кнопками, или несколькими кнопками. Недетерминированный тест эквивалентен набору детерминированных тестов в том смысле, что они дают возможность наблюдения одного и того же множества трасс реализации. Поэтому, как правило, рассматривают только детерминированные тесты.

Что касается реализации, то предполагается, что её недетерминизм – это результат абстрагирования от некоторых неучитываемых внешних факторов – погодных условий, которые определяют выбор того или иного поведения детерминировано. *Гипотеза о глобальном тестировании* предполагает, что любые погодные условия могут быть воспроизведены в тестовом эксперименте. Для этого даже детерминированный тест должен прогоняться несколько раз, чтобы наблюдать все возможные для этого теста трассы реализации.

При тестировании выполняется прогон некоторых тестов из некоторого набора тестов при некоторых погодных условиях. Результатом тестирования является множество ` трасс, наблюдаемых во всех этих тестовых экспериментах. Выносится вердикт ***pass*** (проходит) или ***fail*** (ошибка). Набор тестов *значимый*, если каждая конформная реализация его проходит, *исчерпывающий*, если каждая неконформная реализация его не проходит (обнаруживается ошибка), и *полный*, если он значимый и исчерпывающий. Заметим, что ` – это не обязательно множество всех трасс реализации. Если

спецификация утверждает, что любая реализация с большим множеством трасс f⊇` неконформна, то значимый набор тестов может (хотя и не обязан), а полный набор тестов должен выносить вердикт ***fail*** при наблюдении множества трасс ` (или любого его надмножества). Если спецификация утверждает, что любая реализация с большим множеством трасс f⊇`, наоборот, конформна, то исчерпывающий (и полный) набор тестов может (хотя и не обязан), а полный набор тестов должен выносить вердикт ***pass*** при наблюдении множества трасс ` (или любого его надмножества).

<u>Второе предположение</u>. В настоящей работе мы ограничимся только теми конформностями, которые отвечают *принципу независимости трасс*: любая трасса реализации конформна или неконформна независимо от других её трасс. Конформности такого типа называются *редукциями*. Не является редукцией, например, конформность, которая разрешает реализации иметь как трассу σ_1, так и трассу σ_2, но не обе одновременно. Принцип независимости исключает из рассмотрения конформности типа симуляций, которые основаны на соответствии состояний реализации и спецификации, а также конформности, которые требуют обязательного наличия в реализации тех или иных наблюдений после тех или иных трасс.

Для редукции можно считать, что спецификация s (прямо или косвенно) определяет множество *разрешаемых* трасс p. Если реализация имеет множество трасс f, то конформность означает вложенность f⊆p и является частичным (нестрогим) порядком (рефлексивное, симметричное и транзитивное отношение). Трасса σ∉p называется *ошибкой*.

При тестировании редукции обнаружение любой ошибки σ∈f\p означает, что реализация неконформна. Поэтому значимый набор тестов (тест) может (хотя и не обязан) выносить вердикт ***fail*** сразу, как только наблюдается такая трасса σ. Набор тестов исчерпывающий, если для каждой неконформной реализации хотя бы одна имеющаяся в ней ошибка σ∈f\p может быть обнаружена некоторым тестом из набора, то есть является трассой этого теста. Это означает, что неконформность реализации обнаруживается всегда за конечное время, тогда как вывод о конформности реализации может быть сделан, вообще говоря, только после всех прогонов при всех возможных погодных условиях всех тестов полного набора (число таких прогонов может быть бесконечно).

<u>Третье предположение</u>. На практике, естественно, используются только конечные тесты, точнее, тесты, которые заканчиваются за конечное время. При дискретном взаимодействии это означает, что при задании спецификации и генерации тестов используются только конечные трассы. Заметим, что для спецификации, основанной на конечных трассах, бесконечные тестовые эксперименты ничего не добавляют. Однако в общем случае это не так. Например, рассмотрим две реализации, в которых возможны только два наблюдения: x и y. Кнопки не используются. В одной реализации есть

бесконечная цепочка x. В другой реализации такой бесконечной цепочки нет, но есть бесконечный «веер» конечных цепочек x. В обеих реализациях нет перехода по y. При конечных тестовых экспериментах эти две реализации неразличимы. Для спецификации, в которой наблюдение y считается ошибкой после любого числа x, эти реализации обе конформны. В то же время бесконечный тестовый эксперимент позволяет эти реализации различить: в первой реализации есть бесконечная трасса x, а во второй нет. Если допускаются бесконечные трассы, то спецификация может трактовать бесконечную цепочку x как ошибку. Разумеется, такая ошибка не может быть найдена за конечное время.

По определению любая реализация, содержащая ошибку, неконформна. В то же время, кроме ошибок, определяемых спецификацией, то есть трасс, не принадлежащих множеству разрешаемых трасс p, могут быть другие трассы (принадлежащие p), которые, тем не менее, не встречаются в конформных реализациях. Такие трассы будем называть неконформными. После этого под ошибкой мы будем понимать любую неконформную трассу, а ошибки, определяемые спецификацией (трассы не из p), будем называть ошибками 1-го рода. Ошибка 2-го рода – это неконформная трасса, не являющаяся ошибкой 1-го рода, то есть принадлежащая p.

В данной статье рассматривается проблема зависимости между ошибками и тесно связанная с ней проблема оптимизации полного набора тестов. Будем говорить, что из множества ошибок A следует множество ошибок B и обозначать $A\rightarrow B$, если любая реализация, в которой есть ошибка из A, содержит ошибку из B. Если $A\rightarrow B$, то вместо тестов, которые ловят ошибки из A можно использовать тесты, которые ловят ошибки из B. Если A – это множество всех ошибок, то $B\subset A$ и, очевидно, $B\rightarrow A$. Если также $A\rightarrow B$, то множества A и B эквивалентны (обозначается $A\sim B$). Одним из таких подмножеств ошибок, эквивалентных множеству всех ошибок, является, конечно, множество ошибок 1-го рода. Однако могут существовать и другие множества ошибок, эквивалентные множеству ошибок 1-го рода, в том числе его строгие подмножества. Бывает и так, что множество ошибок 1-го рода бесконечно, но существует эквивалентное ему конечное множество ошибок. Это даёт возможность существенной оптимизации тестов.

Один вид такой зависимости между ошибками присущ любой конформности типа редукции для любого дискретного взаимодействия. Во-первых, любая реализация, содержащая трассу σ, содержит и трассу $\sigma\rho$, где ρ – последовательность кнопок. Поэтому, если $\sigma\rho$ ошибка, то σ тоже ошибка. Поэтому, если $\sigma\rho\in A$, то $A \rightarrow A\cup\{\sigma\}$. Во-вторых, множество трасс реализации префикс-замкнуто. Поэтому если ошибка μ является префиксом трассы σ (будем обозначать это $\mu\leq\sigma$), то σ тоже ошибка. Поэтому, если $\mu\in A$, то $A \rightarrow A\backslash\{\sigma\}$. Это даёт возможность следующей оптимизации тестов: для полноты тестирования достаточно обнаруживать только такие ошибки, которые минимальны по префиксности во множестве всех ошибок (а не только ошибок 1-го рода), такие ошибки не заканчиваются кнопками. Множество таких ошибок эквивалентно множеству ошибок 1-го рода и, тем самым, множеству всех ошибок.

В то же время существует много различных конформностей типа редукции, для которых между ошибками имеются и другие зависимости. Нахождение таких зависимостей и связанная с этим оптимизация тестов иногда представляют собой трудную задачу [например, [6],[7]].

Цель данной статьи – определить общую природу зависимостей между ошибками. Для этого мы формально определим общую модель дискретного взаимодействия и общую конформность типа редукции. Мы покажем следующее. Во-первых, для такой общей редукции не существует зависимости между ошибками, кроме указанной выше. Во-вторых, другие конформности типа редукции являются частным случаем общей редукции, то есть сводятся к ней. При этом сужается класс рассматриваемых спецификаций и тестируемых реализаций. В-третьих, сужение класса спецификаций не влияет на зависимость между ошибками, тогда как сужение класса тестируемых реализаций влечет появление дополнительных зависимостей между ошибками [например, [6],[7]]. Важно отметить, что каждая такая частная редукция определяет некий естественный для неё класс тестируемых реализаций. Однако на практике часто используются дополнительные ограничения на тестируемые реализации, что, в свою очередь, также приводит к появлению дополнительных зависимостей между ошибками [например, [14],[15]]. Иными словами, мы сведём проблему зависимостей между ошибками на классе тестируемых реализаций, естественном для той или иной частной редукции, к общей проблеме зависимостей между ошибками, возникающей как результат сужения класса тестируемых реализаций.

2. Общая модель

В этом разделе мы формально определим общую модель дискретного взаимодействия и общую конформность типа редукции.

2.1. Семантика взаимодействия

Для многих семантик взаимодействия частного вида между кнопками и наблюдениями существует та или иная предустановленная связь. Некоторые из таких семантик мы рассмотрим ниже. В общем же случае никакой предустановленной связи между кнопками и наблюдениями мы предполагать не будем. Другое дело, что в конкретной реализации такая связь может быть.

Будем считать, что заданы два непересекающихся универсума символов: **B** – тестовых воздействий (кнопок – **b**uttons) и **O** – наблюдений (**o**bservations). Такую семантику будем называть ***B*/*O*-*семантикой***. Трасса – это последовательность в алфавите **B**∪**O**. Семантику будем называть *конечной*, если суммарное число кнопок и наблюдений конечны.

2.2. Машина тестирования

B/**O**-семантика моделируется машиной тестирования, представляющей собой «чёрный ящик», внутри которого находится реализация. Машина снабжена клавиатурой для управления и дисплеем для наблюдения.

Клавиатура представляет собой множество кнопок **B**. Тестовое воздействие осуществляется нажатием той или иной кнопки на клавиатуре. Когда нажимается кнопка, машина тестирования передаёт в реализацию однократный сигнал о соответствующем тестовом воздействии и дожидается ответного сигнала о том, что реализация «приняла к сведению» это тестовое воздействие, после чего машина может передавать в реализацию сигнал о следующем тестовом воздействии. Кнопка не фиксируется (автоматически отжимается), что даёт возможность оператору машины нажимать следующую (другую или ту же самую) кнопку. Одновременно можно нажимать только одну кнопку, соответствующую ровно одному тестовому воздействию.

На дисплее машины последовательно высвечиваются символы кнопок и наблюдений, то есть символы из **B**∪**O**. Для того чтобы оператор машины мог различать идущие подряд одинаковые символы, между ними экран кратковременно гаснет. Последовательность символов, появляющихся на экране, как раз и является трассой, наблюдаемой в процессе тестового эксперимента. Символ кнопки появляется на экране в тот момент, когда оператор нажимает соответствующую кнопку. Наблюдение появляется на экране дисплея тогда, когда в реализации происходит соответствующее наблюдаемое событие. Ненаблюдаемая τ-активность реализации никак не отражается на экране дисплея.

Для того чтобы можно было выполнять несколько тестовых экспериментов, машина может быть снабжена кнопкой *рестарта*. Эта кнопка сбрасывает реализацию в начальное состояние и гасит экран. Каждый новый рестарт машины может вызывать изменение погодных условий, от которых зависит поведение реализации. Гипотеза о глобальном тестировании предполагает, что в последовательности рестартов воспроизводятся все возможные погодные условия.

В то же время рестарт позволяет выполнить не более чем счётное число тестовых экспериментов, тем самым, для не более чем счётного числа погодных условий. Для того чтобы обойти это ограничение, машина тестирования может быть снабжена не кнопкой рестарта, а кнопкой *репликации*. Однократное нажатие такой кнопки создаёт множество копий машины тестирования произвольной мощности. Тестирование происходит с каждой копией машины независимым образом, то есть на каждой копии выполняется свой тестовый эксперимент. Для каждой копии фиксируется свой вариант погодных условий. Гипотеза о глобальном тестировании

предполагает, что для каждого варианта погодных условий при репликации создается, по крайней мере, одна копия машины.

Важно отметить, что для конформностей типа редукции репликацию достаточно делать один раз перед началом тестирования, а не много раз после получения тех или иных трасс. Многократная репликация (после каждого шага тестирования, то есть после каждого наблюдения и после нажатия каждой кнопки) требуется для конформностей типа симуляции.

2.3. Реализация

Для конформности типа редукции реализация, фактически, сводится к множеству её трасс. Такая *трассовая модель реализации* формально определяется как множество $I \subseteq (\mathbf{B} \cup \mathbf{O})^*$, которое: 1) не пусто, 2) префикс-замкнуто, 3) вместе с каждой трассой σ содержит и все трассы вида σρ, где ρ – последовательность кнопок.

Для компактного задания множества трасс, в частности, для задания бесконечного множества трасс конечным образом, используется модель LTS (Labelled Transition System). Она представляет собой ориентированный граф, вершины которого называются состояниями, одно состояние выделено в качестве начального, дуги помечены символами из **B**∪**O** и называются переходами. Ненаблюдаемая τ-активность понимается как цепочка элементарных τ-событий, каждое из которых изображается переходом, помеченным символом τ. LTS-реализацию будем называть *конечной*, если конечно число её состояний, достижимых из начального состояния.

Поскольку тестовое воздействие (нажатие кнопки машины тестирования) на реализацию выполняется извне её и не зависит от неё, переход по кнопке означает лишь тот факт, что реализация «узнала» о выполненном тестовом воздействии. В результате такого перехода реализация меняет своё состояние, что впоследствии может привести к изменению её поведения, то есть к появлению других наблюдений. Если в некотором состоянии реализация игнорирует тестовое воздействие, то это эквивалентно тому, что в этом состоянии есть переход-петля по этой кнопке. Поэтому отсутствие перехода по кнопке в состоянии реализации трактуется как наличие перехода-петли по этой кнопке в этом состоянии.

Маршрутом называется последовательность смежных переходов, когда начало любого перехода, кроме первого, совпадает с концом предыдущего перехода. Трасса реализации – это последовательность пометок переходов маршрута, начинающегося в начальном состоянии, с пропуском символа τ.

Множество трасс LTS-реализации является трассовой моделью реализации и, наоборот, для любой трассовой модели реализации существует LTS с таким же множеством трасс.

Очевидно, что для каждой трассы σ существует наименьшая по множеству трасс реализация, содержащая эту трассу σ, – это множество трасс {μρ | μ≤σ & ρ∈B*}. Соответствующая LTS-реализация изображена на 0.

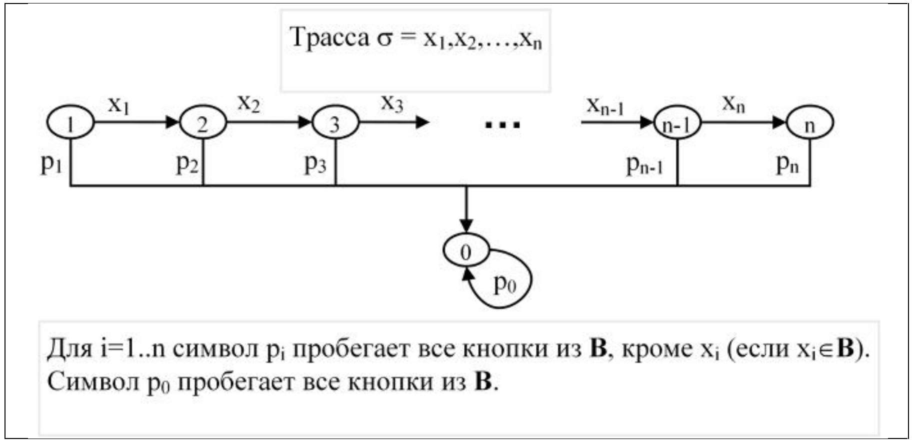

Рис. 1.

2.4. Взаимодействие с реализацией

При тестировании в любой наблюдаемой трассе подпоследовательность кнопок содержит ровно те кнопки, которые оператор машины нажимал и ровно в том порядке, в котором он их нажимал. С внешней точки зрения тестовое воздействие влияет лишь на те наблюдения, которые появляются в трассе. Иными словами, нажатие кнопки лишь регулирует поток наблюдений над поведением реализации. Это достигается с помощью переходов по кнопкам, которые меняют состояние реализации при нажатии кнопки и, тем самым, дальнейший поток наблюдений.

Что касается ненаблюдаемого поведения реализации, то мы исходим из *основного допущения о ²-активности*: между любыми двумя наблюдениями (и до первого в трассе наблюдения) в реализации может быть любая конечная τ-активность [[10]]. В терминах LTS это означает, что между двумя переходами по наблюдению (и перед первым таким переходом) реализация может выполнить любое конечное число τ-переходов. Мы распространяем это допущение также на тестовые воздействия: реализация может выполнить любое конечное число τ-переходов между любыми двумя переходами по наблюдениям или кнопкам (и перед первым таким переходом).

Особенностью нашей модели взаимодействия является *приоритет тестового воздействия над поведением реализации*, как наблюдаемым, так и ненаблюдаемым. Если после получения в тестовом эксперименте трассы σ

оператор не нажимает кнопку, а ждёт наблюдений, то может быть получено любое наблюдение, которое в реализации есть после трассы σ, то есть может быть получена любая имеющаяся в реализации трасса σu, где u – наблюдение. Кроме того, если после трассы σ в реализации есть бесконечная τ-активность (*дивергенция* как бесконечная цепочка τ-переходов), то никаких наблюдений может и не быть. Если же сразу после наблюдения трассы σ оператор нажимает кнопку p, то реализация обязана выполнить переход по кнопке p, будет получена трасса σp. Однако по основному допущению о τ-активности между переходом по последнему символу (наблюдению или кнопке) трассы σ и переходом по следующему наблюдению u в трассе σu или следующей кнопке p в трассе σp реализация может выполнить любое конечное число τ-переходов. Итак, реализация обязана «принять к сведению» тестовое воздействие через конечное время после нажатия соответствующей 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 этой кнопке (отсутствие перехода по кнопке интерпретируется как наличие такой петли).

В результате мы получаем следующий протокол взаимодействия. Если нет тестового воздействия, то есть никакая кнопка не нажата, реализация может выполнить, в зависимости от погодных условий, любую цепочку переходов по наблюдениям и τ-переходов, начинающуюся в её текущем состоянии. Если осуществляется тестовое воздействие, то есть оператор нажал кнопку p, то реализация может выполнить, в зависимости от погодных условий, любую конечную цепочку τ-переходов, после чего должна выполнить любой переход по кнопке p. Выполняя p-переход, реализация как бы «сообщает» машине тестирования о том, что тестовое воздействие ею воспринято. После этого реализация готова к восприятию следующего тестового воздействия (нажатию той или иной кнопки).

Заметим, что при таком протоколе взаимодействия появление на экране дисплея символа кнопки в момент нажатия кнопки, фактически, эквивалентно его появлению в момент совершения реализацией перехода по этой кнопке. Именно поэтому наблюдаемая на экране дисплея трасса совпадает с трассой маршрута, который реализация за это время проходит.

При наличии в состоянии нескольких переходов, которые реализация может выполнять, выбирается один из них недетерминированным образом. Также при нажатой кнопке выбор числа τ-переходов, которые выполняются до перехода по кнопке, происходит недетерминировано. Оба этих выбора понимаются как выборы в зависимости от погодных условий. Гипотеза о глобальном тестировании гарантирует возможность перебора всех возможных погодных условий.

В любом сеансе взаимодействия с реализацией через машину тестирования на экране дисплея наблюдается некоторая трасса реализации, а также (в более ранние моменты времени) все ее префиксы.

2.5. Оператор машины тестирования

Оператор машины тестирования моделирует работу тестовой системы. Мы предполагаем, что при тестировании оператор выполняет тест, понимаемый как инструкция оператору. В этой инструкции указывается, что может делать оператор после получения той или иной трассы: ждать наблюдений и/или нажимать кнопки и какие именно кнопки. Естественно, что если тест определяет поведение оператора после трассы μ, то для того, чтобы можно было получить эту трассу μ, тест должен определять поведение оператора и после любого префикса трассы μ.

Если тест разрешает оператору нажимать кнопку p после трассы μ, то предполагается, что оператор может нажать эту кнопку через любое время после получения трассы μ. Тем самым, оператору *не запрещено* выдерживать любые паузы после получения тех или иных трасс, в том числе перед нажатием следующей кнопки: в любой момент времени он может устроить себе «перерыв на чай». Это означает, что когда оператор нажимает кнопку p спустя какое-то время после трассы μ, а после нажатия кнопки ждёт ещё какое-то время, то реально будет получена не трасса μp, а трасса $\mu\pi_1 p\pi_2$, где π_1 и π_2 – последовательности наблюдений.

В то же время для полноты тестирования необходимо, чтобы любая интересующая нас трасса реализации могла наблюдаться при взаимодействии с реализацией через машину тестирования. А для этого оператор должен иметь *возможность* достаточно быстро нажимать кнопки после полученных трасс. Тогда, нажимая кнопку p сразу после трассы μ, оператор будет наблюдать именно трассу μp. Разумеется, если после этой трассы оператор какое-то время не выключает машину и не нажимает кнопок, то может быть получено продолжение этой трассы последовательностью наблюдений, то есть трасса $\mu p\pi_2$.

2.6. Спецификация и общая редукция

Как было сказано во введении, спецификация явно или неявно определяет множество разрешённых трасс и одновременно его дополнение – множество ошибок 1-го рода. В данной работе под спецификацией будет пониматься произвольное множество конечных трасс, понимаемое просто как множество ошибок 1-го рода. Реализация конформна, если в ней нет ошибок 1-го рода, то есть трасс спецификации. Такую конформность будем называть далее *общей редукцией*.

Спецификацию как множество ошибок 1-го рода можно задавать с помощью порождающего графа, то есть LTS с выделенными конечными вершинами.

LTS-спецификацию будем называть *конечной*, если конечно число её состояний, достижимых из начального состояния. Для некоторых бесконечных множеств ошибок 1-го рода LTS-спецификация может быть конечной. Как известно, для любого порождающего графа существует процедура детерминизации, строящая детерминированный граф, порождающий то же множество последовательностей. Поэтому для любой спецификации существует детерминированная LTS-спецификация, определяющая то же множество трасс. Детерминированность здесь означает, что в каждом достижимом состоянии нет τ-переходов и для каждого символа $x\in\mathbf{B}\cup\mathbf{O}$ определено не более одного перехода по x из этого состояния. Если LTS-спецификация конечная, то после детерминизации она тоже остаётся конечной.

Спецификация s задаёт класс конформных реализаций, который будем обозначать как C_s. Если спецификация содержит пустую трассу, то есть пустая трасса считается ошибкой 1-го рода, то все реализации неконформны, поскольку любая реализация содержит пустую трассу. Если спецификация – это пустое множество, то все реализации конформны.

2.7. Тест

Тестом будем называть множество t конечных трасс. Получение при тестировании любой из этих трасс приводит к вынесению вердикта ***fail***, получение любой другой трассы – к получению вердикта ***pass***. Тест понимается как инструкция оператору машины тестирования. Задача теста – проверить, имеется ли в реализации хотя бы одна из трасс теста: если это так, то тестирование заканчивается и выносится общий вердикт ***fail***.

<u>Нажатие кнопки</u>. В процессе тестирования после получения трассы μ оператор может нажать кнопку p, если трасса μp является префиксом некоторой трассы σ∈t. Предполагается, что если трасса μp, где p кнопка, является префиксом некоторой трассы σ∈t, то хотя бы в одном сеансе тестирования после получения трассы μ оператор нажимает кнопку p, причем достаточно быстро после получения трассы μ. Также предполагается, что если трасса μu, где u наблюдение, является префиксом некоторой трассы σ∈t, то хотя бы в одном сеансе тестирования после получения трассы μ оператор ожидает наблюдения. Если эти предположения выполнены, то гипотеза о глобальном тестировании гарантирует, что если некоторая трасса σ∈t встречается в реализации, то она будет получена хотя бы в одном сеансе тестирования.

<u>Выключение машины</u>. Обозначим префикс-замыкание множества t последовательностей: ***pre***(t) = {μ | ∃σ∈t μ≤σ}. В процессе тестирования после получения трассы μ возможны три случая:

1) μ∈***pre***(t)\t, то есть μ является строгим префиксом какой-либо трассы из t;
2) μ∈t;

3) $\mu \notin$ ***pre***(t), то есть μ не является префиксом какой-либо трассы из t.

В случае 1 оператор должен продолжать сеанс тестирования, то есть не должен выключать машину тестирования. В случае 2 и 3 оператор должен закончить сеанс тестирования, то есть должен выключить машину тестирования. При этом выносится вердикт: в случае 2 – ***fail***, в случае 3 – ***pass***.

Трасса, которая может быть получена в некотором сеансе тестирования с данным тестом (не только в конце сеанса), имеет вид либо $\mu\pi$, где μ является префиксом некоторой трассы $\sigma \in t$, а π последовательность наблюдений, либо $\mu\pi_1 p \pi_2$, где μp является префиксом некоторой трассы $\sigma \in t$, p – кнопка, а π_1 и π_2 последовательности наблюдений. Множество таких трасс будем называть расширением теста и обозначать:

$$\boldsymbol{exp}(t) = \{\mu\pi \mid \mu \in \boldsymbol{pre}(t) \ \& \ \pi \in \mathbf{O}^*\} \cup \{\mu\pi_1 p \pi_2 \mid \mu p \in \boldsymbol{pre}(t) \ \& \ p \in \mathbf{B} \ \& \ \pi_1 \in \mathbf{O}^* \ \& \ \pi_1 \in \mathbf{O}^*\}.$$

Множество трасс, которые могут быть получены в конце сеанса тестирования, равно (***exp***(t) \ ***pre***(t)) ∪ t = ***exp***(t) \ (***pre***(t) \ t).

Реализация *проходит* тест, если при любом сеансе тестирования (при любых погодных условиях) выносится вердикт ***pass***. Реализация *проходит* набор тестов, если она проходит каждый тест из набора. Для заданного класса реализаций (в частности, для класса всех реализаций) набор тестов (тест) значимый, если каждая конформная реализация из этого класса его проходит, исчерпывающий, если каждая неконформная реализация из этого класса его не проходит, и полный, если он значимый и исчерпывающий.

Тест *детерминированный*, если он однозначно определяет поведение оператора. Это значит, что любая трасса из префикс-замыкания теста в этом префикс-замыкании либо продолжается одной кнопкой и не продолжается наблюдениями, либо не продолжается кнопками:

$$\forall \mu \in \boldsymbol{pre}(t) \ (\ |\{p \in \mathbf{B} \mid \mu p \in \boldsymbol{pre}(t)\}| = 1 \ \& \ \{u \in \mathbf{O} \mid \mu u \in \boldsymbol{pre}(t)\} = \varnothing \ ) \vee \{p \in \mathbf{B} \mid \mu p \in \boldsymbol{pre}(t)\} = \varnothing.$$

Тест *примитивный*, если он содержит только одну трассу. Очевидно, что примитивный тест детерминированный. Любой тест t эквивалентен объединению множества примитивных тестов в том смысле, что они выносят вердикт ***fail*** для одних и тех же реализаций: $t = \cup \{ \{\sigma\} \mid \sigma \in t \}$. Также очевидно, что спецификация (как множество ошибок 1-го рода) является полным тестом на классе всех реализаций. Отсюда следует, что набор примитивных тестов, построенных по всем ошибкам 1-го рода, то есть по всем трассам спецификации, является полным на классе всех реализаций.

2.8. Нормализация спецификации и оптимизация тестов

Как уже было отмечено во введении, кроме ошибок 1-го рода, то есть трасс спецификации, могут быть и другие ошибки – неконформные трассы, то есть трассы, не встречающиеся в конформных реализациях. Ошибки, не являющиеся ошибками 1-го рода, называются ошибками 2-го рода. Для полноты тестирования достаточно обнаруживать только такие ошибки, которые минимальны по префиксности во множестве всех ошибок (а не только ошибок 1-го рода). Такие ошибки будем называть *первичными* ошибками, *вторичная* ошибка – это ошибка, у которой есть строгий префикс, являющийся ошибкой. Первичные ошибки не заканчиваются кнопками. Множество первичных ошибок эквивалентно множеству ошибок 1-го рода и, тем самым, множеству всех ошибок. Оно, очевидно, является наименьшим по вложенности подмножеством ошибок, эквивалентным множеству всех ошибок. Его можно рассматривать как спецификацию, которую будем называть *нормализованной* спецификацией.

Для каждой спецификации s множество всех ошибок строится с помощью систематического применения следующих операций:

1) Если p – кнопка и $\sigma p \in s$, то добавляем в s трассу σ.
2) Если $\mu \in s$ и $\mu < \sigma$, то добавляем в s трассу σ.

Если s* – множество всех ошибок для спецификации s, то процедура нормализации сводится к удалению неминимальных по префиксности ошибок: если $\mu \in s$, $\sigma \in s$ и $\mu < \sigma$, то удаляем из s трассу σ.

Нормализацию можно выполнить и непосредственно по исходной спецификации s как систематическое применение следующих операций:

3) Если p – кнопка и $\sigma p \in s$, то добавляем в s трассу σ.
4) Если $\mu \in s$, $\sigma \in s$ и $\mu < \sigma$, то удаляем из s трассу σ.

Нормализованные спецификации взаимно-однозначно соответствуют своим классам конформных реализаций: $a = b \Leftrightarrow C_a = C_b$.

Пусть спецификация s нормализованная. Как было отмечено выше, для каждой трассы σ существует наименьшая по множеству трасс реализация, содержащая эту трассу σ, – это множество трасс $\{\mu\rho \mid \mu \le \sigma \ \& \ \rho \in B^*\}$. Отсюда следует, что набор T тестов значимый тогда и только тогда, когда каждая трасса каждого теста из набора имеет в качестве префикса ошибку из s, то есть s коинициально ĆT. Набор T тестов исчерпывающий тогда и только тогда, когда каждая трасса из s имеет префикс, являющийся трассой некоторого теста из набора, то есть ĆT коинициально s.

Набор T тестов полный тогда и только тогда, когда s и ĆT взаимно коинициальны. Поскольку s нормализована, условие ĆT коинициально s можно заменить на условие вложенности $s \subseteq$ ĆT. Действительно, в противном случае найдётся трасса $\mu\in s\backslash\acute{C}T$, а тогда, поскольку $\acute{C}T$ коинициально s, найдётся трасса $\mu_1<\mu$ такая, что $\mu_1\in\acute{C}T$, а тогда, поскольку s коинициально $\acute{C}T$, найдётся трасса $\mu_2\leq\mu_1$ такая, что $\mu_2\in s$, следовательно, в s имеются две ошибки $\mu_2<\mu$, что противоречит нормализованности s.

Иными словами, набор T тестов полный тогда и только тогда, когда все трассы всех его тестов – это все первичные ошибки (трассы из нормализованной спецификации s) и некоторые их продолжения. Очевидная оптимизация – это удаление таких продолжений, что даёт в итоге набор T‘ тестов, множество всех трасс всех тестов которого – это нормализованная спецификация: $s = \acute{C}\{\{\sigma\} \mid \sigma\in s\} = \acute{C}T'$. Тем самым, оптимизированный полный набор T‘ тестов – это покрытие s, а набор примитивных тестов $\{\{\sigma\}|\sigma\in s\}$ является одним из возможных разбиений s.

3. Класс реализаций

По разным причинам в качестве тестируемых реализаций рассматриваются не любые реализации, а принадлежащие тому или иному классу реализаций I. Это приводит к появлению дополнительных зависимостей между ошибками (в том числе, первичными) и, соответственно, даёт возможность дополнительной оптимизации тестов.

Первое определение эквивалентности спецификаций: Будем говорить, что две спецификации a и b эквивалентны на классе реализаций I, если на этом классе спецификации определяют одну и ту же конформность реализаций: $I\cap C_a=I\cap C_b$. Если I – класс всех реализаций, то $C_a\subseteq I$ и $C_b\subseteq I$, поэтому эквивалентность нормализованных спецификаций ($C_a=C_b$) совпадает с равенством (a=b). На других подклассах реализаций это, вообще говоря, не верно.

Трассу, встречающуюся в реализациях класса I, будем называть *актуальной* на классе I. Если I – множество трассовых реализаций, то множество актуальных трасс равно $\cup I$. На классе всех реализаций все трассы актуальны. На других классах реализаций могут быть как актуальные, так и неактуальные трассы. Трасса, которая встречается в конформных реализациях из класса I, называется конформной на классе I. Это конформная трасса, которая актуальна на классе I. Ошибки (в том числе ошибки 1-го рода и 2-го рода) делятся на актуальные и неактуальные. При тестировании реализаций из класса I, очевидно, достаточно обнаруживать только актуальные на этом классе ошибки. Множество трасс, конформных на классе I, для спецификации s равно $\cup(I\cap C_s)$. Соответственно, множество ошибок, актуальных на классе I, равно $\cup I \setminus \cup(I\cap C_s)$.

Это дает нам второе определение эквивалентности спецификаций: Будем говорить, что две спецификации a и b эквивалентны на классе реализаций I, если они определяют одно и то же множество актуальных ошибок: $\cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b)$.

На самом деле, два определения эквивалентности спецификаций равносильны:

$$I\cap C_a=I\cap C_b \Leftrightarrow \cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b).$$

Докажем это. Сначала покажем, что $\cup(I\cap C_a) = \cup(I\cap C_b) \Leftrightarrow \cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b)$.

Действительно, если $\cup(I\cap C_a) = \cup(I\cap C_b)$, то, очевидно, $\cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b)$.

Покажем, что если $\cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b)$, то $\cup(I\cap C_a) = \cup(I\cap C_b)$. Пусть это не верно, например, трасса $\sigma\in\cup(I\cap C_a) \setminus \cup(I\cap C_b)$. Тогда эта трасса принадлежит некоторой реализации из I, которая конформна для a. Но тогда эта реализация не принадлежит C_b, то есть в ней есть ошибка из b. Эта ошибка принадлежит $\cup(I\cap C_a)$, следовательно, не принадлежит $\cup I \setminus \cup(I\cap C_a)$. Также эта ошибка принадлежит $\cup I$, но не принадлежит $\cup(I\cap C_b)$, следовательно, принадлежит $\cup I \setminus \cup(I\cap C_b)$, что противоречит равенству $\cup I \setminus \cup(I\cap C_a) = \cup I \setminus \cup(I\cap C_b)$. Теперь покажем, что $I\cap C_a=I\cap C_b \Leftrightarrow \cup(I\cap C_a) = \cup(I\cap C_b)$. Если $I\cap C_a=I\cap C_b$, то, очевидно, $\cup(I\cap C_a) = \cup(I\cap C_b)$. Покажем, что если $\cup(I\cap C_a) = \cup(I\cap C_b)$, то $I\cap C_a=I\cap C_b$. Пусть это не верно, тогда есть некоторая реализация, принадлежащая, например, $I\cap C_a \setminus I\cap C_b$. Тогда эта реализация принадлежит I и не принадлежит C_b, следовательно, в ней есть некоторая ошибка из b. Тем самым, эта ошибка принадлежит $\cup(I\cap C_a)$. Но эта ошибка не может принадлежать $\cup(I\cap C_b)$, что противоречит равенству $\cup(I\cap C_a) = \cup(I\cap C_b)$.

Теперь пусть I – это отображение, задающее для каждой спецификации s класс тестируемых реализаций I_s. Будем говорить, что для отображения I спецификация b *может быть использована вместо спецификации* a, если 1) $I_a \subseteq I_b$, 2) $I_a\cap C_a=I_a\cap C_b$. Первое условие говорит о том, что любая реализация, которую мы могли тестировать для проверки конформности спецификации a, можно тестировать для проверки конформности спецификации b. Второе условие (эквивалентность спецификаций на классе I_a) говорит о том, что спецификации a и b определяют одинаковую конформность реализаций на классе тестируемых реализаций для спецификации a.

Ошибка обнаруживается набором тестов, если она является трассой одного из тестов набора. Любой набор тестов, который полон на классе тестируемых реализаций I, очевидно, задаёт множество обнаруживаемых ошибок (множество всех трасс всех его тестов), эквивалентное на классе I множеству всех ошибок.

4. Гипотеза о безопасности

В наших работах [[1],[2],[5],[6]] мы ввели понятие безопасного тестирования. Это такое тестирование, при котором не проходятся трассы реализации, которые считаются опасными. Гипотеза о безопасности определяет класс реализаций, которые можно безопасно тестировать для проверки конформности данной спецификации.

4.1. Общий вид гипотезы о безопасности

Будем говорить, что задана *гипотеза о безопасности*, если для каждой реализации i определено префикс-замкнутое подмножество ***SafeTraces***(i) ее трасс, которые называются безопасными трассами. Тестирование данной реализации называется безопасным, если в процессе него могут получаться только безопасные трассы этой реализации. Реализация i безопасна для теста t, если тестирование с помощью этого теста безопасно для этой реализации: ***exp***(t) ⊆ ***SafeTraces***(i). Каждый тест t определяет класс безопасных реализаций ***SafeImpl***(t) = { i | ***exp***(t) ⊆ ***SafeTraces***(i) }. Набор тестов T определяет класс реализаций, безопасных для каждого теста из набора: ***SafeImpl***(T) = { i | ∀t∈T ***exp***(t) ⊆ ***SafeTraces***(i) } = ∩ { ***SafeImpl***(t) | t∈T }. Спецификация s определяет класс безопасных реализаций как класс реализаций, безопасных для полного теста s или, что то же самое, для набора примитивных тестов, построенных по ошибкам спецификации: ***SafeImpl***(s) = ***SafeImpl***({{σ}|σ∈s}). В общем случае, если предполагается безопасное тестирование реализаций из заданного класса I, тестироваться будут безопасные реализации из класса I, то есть реализации из класса I ∩ ***SafeImpl***(s).

4.2. Гипотеза о конечном времени ожидания наблюдения

Для того чтобы каждый сеанс тестирования был конечным по времени, нужно, чтобы были конечными времена ожидания кнопок и наблюдений на экране дисплея: 1) кнопка должна появляться на экране дисплея через конечное время после ее нажатия, 2) если оператор ждет наблюдений, то какое-нибудь наблюдение должно появиться на экране дисплея через конечное время.

Первое условие гарантированно выполнено в данной модели взаимодействия с реализацией. А второе условие может и не выполняться. Сформулируем требование к реализации, чтобы выполнялось это второе условие для данного теста.

<u>Гипотеза о наблюдениях – ª-гипотеза</u>[1]: если трасса реализации является префиксом трассы теста и продолжается в префикс-замыкании теста наблюдением, то в реализации она также должна продолжаться каким-нибудь (не обязательно тем же самым) наблюдением при любом поведении реализации. Это означает: 1) в реализации в каждом стабильном состоянии (состоянии, в котором не начинаются τ-переходы) после этой трассы имеется

[1] Символом λ принято обозначать ситуацию, когда возникает deadlock или дивергенция [[10]]

переход по какому-нибудь наблюдению, 2) после трассы нет дивергенции. λ-гипотеза является частным случаем гипотезы о безопасности.

Определим формально множество ***SafeTraces***ª(i) безопасных трасс реализации i для λ-гипотезы. λ-трассой реализации будем называть трассу реализации, которая заканчивается в стабильном состоянии, где нет переходов по наблюдениям, или в дивергентном состоянии. Трассу реализации будем называть безопасной, если любой ее строгий префикс, за которым в трассе следует наблюдение, не является λ-трассой.

λ-гипотеза не меняет актуальность трасс: все трассы актуальны. λ-гипотеза меняет конформность трасс: на классе безопасных реализаций, определяемом этой гипотезой, трасса μ неконформна, если для каждого наблюдения u трасса μu является ошибкой. Для определения первичных ошибок спецификации в случае λ-гипотезы применяется следующая **<u>процедура ª-нормализации спецификации</u>**: систематически применяем три действия:

1) Если для каждого наблюдения u трасса σu∈s, то добавляем в s трассу σ.
2) Если p – кнопка и σp∈s, то добавляем в s трассу σ.
3) Если μ∈s, σ∈s и μ<σ, то удаляем из s трассу σ.

Полученное множество трасс – это множество первичных ошибок в случае λ-гипотезы.

4.3. Гипотеза о разрушении

Другой разновидностью гипотезы о безопасности является, так называемая, гипотеза о разрушении. Под разрушением понимается любое поведение реализации, которое нежелательно во время тестирования [[1],[2],[5]]. Причины нежелательности того или иного поведения могут быть самыми разными, мы не налагаем здесь никаких ограничений. Для изображения разрушения в LTS-модели реализации некоторые её переходы по наблюдениям или τ-переходы (ненаблюдаемое поведение) заменяются γ-переходами, то есть переходами, помеченными специальным символом разрушения – γ.

Теперь под моделью реализации мы будем понимать LTS в алфавите с добавленным символом γ. Поскольку нас не интересует поведение реализации после разрушения, под трассой будем понимать последовательность кнопок и наблюдений, быть может, заканчивающуюся разрушением.

<u>Гипотеза о разрушении – ¥-гипотеза</u>: для спецификации s любая трасса из ***exp***(s) не продолжается в реализации разрушением. γ-гипотеза является частным случаем гипотезы о безопасности.

Определим формально множество ***SafeTraces***¥(i) безопасных трасс реализации i для γ-гипотезы: трассу реализации будем называть безопасной, если любой ее префикс не продолжается в реализации разрушением.

γ-гипотеза меняет актуальность трасс: трасса актуальная, если она не имеет вида μγ, где μ∈***exp***(s). γ-гипотеза не меняет конформность актуальных трасс: на классе безопасных реализаций, определяемом этой гипотезой, неконформна та и только та актуальная трасса, префикс которой является ошибкой. Поскольку γ-гипотеза не меняет конформность актуальных трасс, процедура нормализации не требуется: все ошибки спецификации являются первичными.

λ- и γ-гипотезы в совокупности определяют класс безопасных реализаций ***SafeImpl***ª¥(s) = ***SafeImpl***ª(s) ∩ ***SafeImpl***¥(s).

5. Моделирование других семантик

В этом разделе мы покажем, каким образом некоторые известные конформности типа редукции сводятся к общей редукции в **B**/**O**-семантике, описанной выше.

5.1. R/Q-семантика

В наших работах [[1],[5],[6]] рассматривалась **R**/**Q**-семантика как обобщение многих семантик взаимодействия, в частности, семантики популярного отношение ***ioco*** [[12],[13]]. **R**/**Q**-семантика задаётся двумя непересекающимися множествами кнопок: **R** и **Q**. Фиксируется жёсткая связь между кнопками и наблюдениями таким образом, что каждой кнопке p однозначно соответствует подмножество ***obs***(p) наблюдений, «разрешаемых» этой кнопкой. Наблюдения делятся на действия из фиксированного алфавита **L** и отказы – подмножества **L**, причём отказ r⊆**L** понимается как отсутствие действий из множества r. **R**-кнопке соответствует как множество r⊆**L** разрешаемых ею действий, так и отказ r, то есть такая кнопка задаётся множеством разрешаемых ею наблюдений r∪{r}[2]. **Q**-кнопке соответствует только множество q⊆**L** разрешаемых ею действий (говорят, что соответствующий отказ q ненаблюдаем). После нажатия кнопки допускается только одно наблюдение, разрешаемое этой кнопкой; чтобы получить следующее наблюдение, нужно нажать ту же или другую кнопку. Это правило не распространяется на τ-активность и разрушение. В LTS-модели реализации изображаются только переходы по действиям, τ- и γ-переходы. Переходы по отказам – виртуальные: отказ r, где r∪{r}∈**R**, наблюдается в стабильном

[2] В наших предыдущих работах кнопка задавалась только множеством r разрешаемых ею действий с дополнительным указанием, что r∈**R**.

состоянии, в котором нет переходов по действиям из r; можно считать, что в этом состоянии имеется виртуальный переход-петля по отказу r.

В **R**/**Q**-семантике реализация задавалась LTS в алфавите **L**∪{γ}. Определялось отношение ***safe in*** безопасности кнопок после трасс реализации. Кнопка безопасна после трассы, если она *неразрушающая*: в реализации трасса не заканчивается в дивергентном состоянии или в состоянии, где есть переход по действию, разрешаемому этой кнопкой, ведущий в состояние, где есть γ-переход. Для **R**-кнопки этого достаточно для её безопасности по ***safe in***, а для **Q**-кнопки дополнительно требуется, чтобы трасса не заканчивалась в стабильном состоянии, где нет переходов по действиям из этой кнопки.

Спецификация задавалась LTS в алфавите **L**∪{γ} и, кроме того, отношением ***safe by***, которое определяло кнопки, безопасные после трасс спецификации. Это отношение должно соблюдать три правила. 1) Безопасные по ***safe by*** кнопки неразрушающие. 2) Если после трассы некоторое действие разрешается неразрушающей кнопкой, то оно разрешается и некоторой (не обязательно той же самой) безопасной по ***safe by*** кнопкой. 3) Если **Q**-кнопка безопасна по ***safe by*** после трассы, то трасса продолжается в спецификации каким-нибудь действием, разрешаемым этой кнопкой. Безопасность кнопок определяет безопасные трассы спецификации как трассы, в которых каждое наблюдение разрешается некоторой кнопкой, безопасной после предшествующего этому наблюдению префиксу трассы.

Гипотеза о безопасности в **R**/**Q**-семантике требовала: 1) разрушение возможно в реализации с самого начала (до нажатия кнопок), если это имеет место в спецификации; 2) после общей безопасной трассы спецификации и реализации кнопка, безопасная по ***safe by*** в спецификации, должна быть безопасна по ***safe in*** в реализации. Класс безопасных реализаций, определяемый этой гипотезой о безопасности, обозначим ***SafeImpl***$_{\mathbf{R/Q}}$(s).

Отношение конформности ***saco*** определялось для реализаций, удовлетворяющих такой гипотезе о безопасности, и требовало: после общей безопасной трассы спецификации и реализации любое наблюдение в реализации, которое разрешается кнопкой, безопасной после этой трассы по ***safe by*** в спецификации, должно быть после этой трассы и в спецификации. Класс безопасных и конформных реализаций, определяемый отношением ***saco***, обозначим ***ConfImpl***$_{\mathbf{R/Q}}$(s).

Для генерации стандартного полного набора тестов используются, так называемые, тестовые трассы спецификации. Тестовая трасса – это безопасная трасса спецификации, продолженная наблюдением, которое разрешается какой-нибудь кнопкой, безопасной в спецификации после этой трассы. Эти кнопки как раз и вставляются в трассу для получения теста: перед каждым **R**-отказом r вставляется кнопка r∪{r}, а перед каждым действием z – какая-нибудь безопасная кнопка p, разрешающая z. Если тестовая трасса отсутствует в спецификации, назначается вердикт ***fail*** (в спецификации нет последнего наблюдения трассы).

Чтобы представить **R**/**Q**-семантику как частный случай **B**/**O**-семантики и конформность ***saco*** как частный случай общей редукции, прежде всего, будем предполагать, что все её кнопки и наблюдения являются кнопками и наблюдениями общей машины тестирования: $\mathbf{R}\cup\mathbf{Q} \subseteq \mathbf{B}$ и $\mathbf{L}\cup\mathbf{R} \subseteq \mathbf{O}$.

Сначала выполним следующее преобразование LTS-реализаций:

1) Добавляем виртуальные петли по отказам (в стабильных состояниях).
2) Для каждой кнопки $p\in\mathbf{R}\cup\mathbf{Q}$ и каждого состояния a добавляем новое состояние a_p и новый переход $a \xrightarrow{p} a_p$.
3) В каждом новом состоянии a_p проводим переход $a_p \xrightarrow{x} b$ тогда и только тогда, когда $x\in$***obs***(p) и имеется переход $a \xrightarrow{x} b$.
4) В каждом новом состоянии a_p проводим переход $a_p \xrightarrow{\tau} b_p$ тогда и только тогда, когда имеется переход $a \xrightarrow{\tau} b$, и проводим переход $a_p \xrightarrow{\gamma} b_p$ тогда и только тогда, когда имеется переход $a \xrightarrow{\gamma} b$.
5) После этого удаляем все переходы по наблюдениям из старых состояний, оставляя τ- и γ-переходы.

У нас получится LTS, состояния которой делятся на «старые» и «новые», переходы по кнопкам ведут из старых состояний в новые (из a в a_p), переходы по наблюдениям – из новых состояний в старые (из a_p в b), причем только по тем наблюдениям, которые разрешаются кнопкой, которой помечен переход в это новое состояние. Переход по отказу r ведёт из состояния a_r в состояние a. τ- и γ-переходы ведут как из старых состояний в старые (из a в b), так и из соответствующих новых в соответствующие новые (из a_p в b_p).

Преобразованная **R**/**Q**-спецификация – это множество ошибок, состоящее из всех трасс тестов стандартного полного набора тестов, которым назначен вердикт ***fail***.

После этих преобразований конформность рассматривается не на классе всех реализаций, а на классе преобразованных **R**/**Q**-реализаций. Из-за этого возникают множественное следование ошибок и эквивалентные множества ошибок, не совпадающие со спецификацией, то есть возникают различные эквивалентные спецификации, которые остаются различными даже после нормализации. Все такие спецификации получаются в результате аналогичного преобразования из любого полного набора **R**/**Q**-тестов, если взять все ***fail***-трассы тестов набора как ошибки. Или, иными словами, существуют полные наборы тестов, различие между которыми не устраняется тривиальной оптимизацией, аналогичной нормализации спецификаций. В этом и состоит проблема оптимизации тестов для **R**/**Q**-семантики.

Тем не менее, эта проблема сводится к проблеме эквивалентности спецификаций на том или ином классе реализаций. Любую новую спецификацию с ª¥-гипотезой можно использовать вместо старой спецификации на классе реализаций, определяемом **R**/**Q**-гипотезой о безопасности. Если результаты преобразований реализации i и спецификации s обозначить i^* и s^*, соответственно, то:

(***SafeImpl***$_{\mathbf{R/Q}}$(s))* ⊆ ***SafeImpl***ª¥(s^*) и

(***ConfImpl***$_{\mathbf{R/Q}}$(s))* = (***SafeImpl***$_{\mathbf{R/Q}}$(s))$^*\cap C_{s^*}$.

Любую безопасную для данной спецификации трассу, которую мы можем получить на **R**/**Q**-машине с данной реализацией i, мы можем получить на **B**/**O**-машине с преобразованной реализацией i^*. Нажатие кнопки точно такое же, потом ждём наблюдения, а потом снова нажимаем кнопку и т.д. Заметим, что нажатие «лишней» кнопки из $\mathbf{B} \setminus (\mathbf{R}\cup\mathbf{Q})$ преобразованной реализацией просто игнорируется, поскольку в каждом её состоянии нет перехода по такой кнопке, что трактуется как наличие перехода-петли по кнопке. «Лишние» наблюдения из $\mathbf{O} \setminus (\mathbf{L}\cup\mathbf{R})$ отсутствуют в преобразованной реализации и поэтому не будут появляться на экране дисплея.

5.2. R/Q-семантика с приоритетами

В **R**/**Q**-семантике τ- и γ-действия всегда разрешены, а нажатие кнопки p разрешает реализации, кроме того, выполнять любое действие из множества действий p. Если реализация может выполнить любое действие, разрешаемое нажатой кнопкой и определённое в её текущем состоянии, то говорят, что система не имеет приоритетов. В этом случае для того, чтобы реализация могла выполнить действие z, определённое в её текущем состоянии, можно нажимать любую кнопку p такую, что $z\in p\cup\{\tau,\gamma\}$ (если $z=\tau$ или $z=\gamma$, то такое действие может выполняться и в том случае, когда никакая кнопка не нажата). Система с приоритетами – это такая система, в которой выполнимость действия, определённого в текущем состоянии реализации, зависит от множества разрешённых внешних действий (если никакая кнопка не нажата, это множество пусто). При этом внешнее действие должно принадлежать «кнопочному» множеству p, а τ- и γ-действия разрешены при нажатии любой кнопки, а также когда никакая кнопка не нажата.

Системы с приоритетами для **R**/**Q**-семантики были введены в наших работах [[3],[4]]. В LTS-модели такой системы переход помечается не только действием z из $\mathbf{L}\cup\{\tau,\gamma\}$, но и предикатом π от множества разрешённых внешних действий, то есть парой (z,π). Если разрешено множество внешних действий p (нажата **R**-кнопка $p\cup\{p\}$, **Q**-кнопка p или $p=\varnothing$ и никакая кнопка не нажата), то такой переход может выполняться только в том случае, когда $z\in p\cup\{\tau,\gamma\}$ и π(p)=***true***. Такой предикат можно понимать как булевскую функцию от булевских переменных $z_1,z_2,\ldots$, взаимно-однозначно соответствующих внешним действиям из алфавита **L**: переменная z_i принимает значение ***true***, если $z_i\in p$. Эта булевская функция может быть представлена в виде совершенной дизъюнктивной нормальной формы, дизъюнкты которой соответствуют множествам разрешённых внешних действий $p_1,p_2,\ldots$ Поэтому переход по паре (z,π) эквивалентен множеству кратных переходов вида (z,p_j); переход (z,p_j) выполняется, если $z\in p_j\cup\{\tau,\gamma\}$ и разрешено множество внешних действий p_j.

При наличии приоритетов меняются понятия отказа и дивергенции. Отказ p возникает при нажатии кнопки p в том случае, когда в текущем состоянии нет переходов, помеченных парой вида (z,p), где $z \in p \cup \{\tau,\gamma\}$. Дивергенция 1) при нажатой кнопке p, или 2) когда никакая кнопка не нажата, возникает, когда в текущем состоянии начинается бесконечная цепочка переходов, помеченных, в первом случае, парой (τ,p) или, во втором случае, парой $(\tau,\varnothing)$. Соответственно, будем говорить о p-дивергенции и о p-дивергентных и p-конвергентных состояниях.

Рассмотрим несколько характерных примеров использования приоритетов.

Выход из дивергенции. Запрос, поступающий извне, может бесконечно долго игнорироваться системой, если он имеет тот же приоритет, что бесконечная внутренняя активность, то есть дивергенция. Заметим, что внутренняя активность может быть инициирована предыдущим запросом. Если речь идёт о составной системе, собранной из нескольких компонентов, то дивергенция может быть естественным результатом взаимодействия компонентов между собой. И в этом случае для обработки запроса, поступающего в систему (в один из её компонентов) извне, он должен иметь больший приоритет, чем внутреннее взаимодействие. Моделирование в **R**/**Q**-семантике. Переход по внешнему действию имеет тождественно истинный предикат, а τ-переход имеет предикат π, истинный только на пустом подмножестве алфавита внешних действий: $\pi(U)=(U=\varnothing)$.

Выход из осцилляции (приоритет приёма над выдачей). Под осцилляцией понимается бесконечная цепочка выдачи сообщений системой. Для того чтобы такую цепочку можно было прервать, заставив систему обрабатывать поступающий извне запрос, последний должен иметь больший приоритет, чем выдача сообщений. Моделирование в **R**/**Q**-семантике. Переход по запросу имеет тождественно истинный предикат, а переход по выдаче сообщения имеет предикат π, истинный на любом подмножестве действий, не содержащем запросов: $\pi(U)=(\forall ?x\ ?x \notin U)$, где префиксный знак «?» означает запрос. Обычно также подразумевается, что внутренняя активность менее приоритетна, чем приём запроса, то есть τ-переход имеет такой же предикат, как переход по выдаче сообщения.

Приоритет выдачи над приёмом в неограниченных очередях. Этот обратный пример характерен для неограниченной очереди, используемой в качестве буфера между взаимодействующими системами, в частности, при тестировании в контексте [[11]]. Здесь нужно, чтобы выборка из очереди была приоритетней постановки в очередь. В противном случае очередь имеет право только принимать сообщения и никогда их не выдавать. При тестировании в контексте для входной очереди это означает, что все входные сообщения, посылаемые тестом, не доходят до реализации, бесконечно накапливаясь в очереди. Соответственно, для выходной очереди это означает, что тест может не получать никаких ответных сообщений от реализации, хотя она их выдаёт, поскольку они «оседают» в очереди. Моделирование в **R**/**Q**-семантике.

Переход по выдаче имеет тождественно истинный предикат, а переход по приёму имеет предикат π, истинный на любом подмножестве действий, не содержащем выдачу: $\pi(U)=(\forall !y\ !y \notin U)$, где префиксный знак «!» означает выдачу. Обычно также подразумевается, что внутренняя активность менее приоритетна, чем выдача, то есть τ-переход имеет такой же предикат, как переход по приёму.

Прерывание цепочки действий. Команда «отменить» (cancel) должна останавливать выполнение последовательности действий, инициированной предыдущим запросом, и вызывать цепочку завершающих действий. При отсутствии приоритетов такая команда, даже если она выдана сразу после выдачи запроса, имеет право быть выполнена только после того, как вся обработка закончится, то есть, фактически, ничего «не отменяет». Моделирование в **R**/**Q**-семантике. Переход по команде «отменить» (cancel) имеет тождественно истинный предикат, а все остальные переходы имеют предикат π, истинный на любом подмножестве действий, не содержащем “cancel”: $\pi(U)=(cancel \notin U)$.

Приоритетная обработка запросов. Если в систему поступает одновременно несколько запросов, то часто требуется их обработка в соответствии с некоторыми приоритетами между ними. Это реализуется в виде очереди запросов с приоритетами или в виде нескольких очередей запросов с приоритетами между очередями. К этому типу приоритетов относится и обработка аппаратных прерываний в операционной системе. Моделирование в **R**/**Q**-семантике. Множество запросов разбивается на непересекающиеся подмножества $X_1,X_2,\ldots$ так, что запросы из подмножества с большим индексом имеют больший приоритет. Предикат π_i на переходе по запросу из X_i истинен на любом подмножестве действий, не содержащем запросы из подмножества с большим номером: $\pi_i(U)=(\forall j>i\ U \cap X_j=\varnothing)$.

Как было сказано выше, в **B**/**O**-семантике имеется приоритет тестового воздействия над поведением реализации, как наблюдаемым, так и ненаблюдаемым. Это даёт возможность представить **R**/**Q**-семантику с приоритетами как частный случай **B**/**O**-семантики и конформность ***saco*** с приоритетами как частный случай общей редукции. Как и для **R**/**Q**-семантики без приоритетов будем предполагать, что все её кнопки и наблюдения являются кнопками и наблюдениями общей машины тестирования: $\mathbf{R} \cup \mathbf{Q} \subseteq \mathbf{B}$ и $\mathbf{L} \cup \mathbf{R} \subseteq \mathbf{O}$.

Сначала выполним следующее преобразование LTS-реализаций:

1) Добавляем виртуальные петли по отказам: если в состоянии a нет переходов вида $a \xrightarrow{(z,p)} b$, где $z \in p \cup \{\tau,\gamma\}$, добавляем переход $a \xrightarrow{(p,p)} a$.
2) Для каждого подмножества действий $p \subseteq \mathbf{L}$ и каждого состояния a добавляем новое состояние a_p и новый переход $a \xrightarrow{p} a_p$.

3) В каждом новом состоянии a_p проводим переход a_p—x→b тогда и только тогда, когда x∈***obs***(p) и имеется переход a—(x,p)→b.
4) В каждом новом состоянии a_p проводим переход a_p—τ→b_p тогда и только тогда, когда имеется переход a—(τ,p)→b, и проводим переход a_p—γ→b_p тогда и только тогда, когда имеется переход a—(γ,p)→b.
5) В каждом новом состоянии a_p для каждого подмножества q⊆**L** проводим переход a_p—q→a_q.
6) В каждом старом состоянии a переход a—(τ,∅)→b заменяется переходом a—τ→b, а переход a—(γ,∅)→b заменяется переходом a—γ→b. Все остальные переходы из состояния a удаляются.

У нас получится LTS, состояния которой делятся на «старые» и «новые». Переход по кнопке p ведёт из старого состояния в новое состояние: a—p→a_p, а также из нового состояния в новое состояние: a_p—q→a_q. Переход по x, где x наблюдение, ведёт из нового состояния в старое: a_p—x→b, причем только в том случае, когда был переход a—(x,p)→b. Если x – это отказ, то p=x и b=a. τ- и γ-переход ведёт либо из нового состояния в новое состояние: a_p—τ/γ→b_p, если был переход a—(τ/γ,p)→b, либо из старого состояния в старое состояние: a—τ/γ→b, если был переход a—(τ/γ,∅)→b.

Преобразование спецификации, конформность, класс тестируемых реализаций и тестирование через машину тестирования определяются аналогично случаю отсутствия приоритетов.

Отметим одну особенность такого моделирования **R**/**Q**-семантики с приоритетами средствами **B**/**O**-семантики. В **R**/**Q**-семантике, содержащей пустую кнопку (∅∈**R**∪**Q**), не различаются τ- и γ-переходы при нажатии пустой кнопки и при отсутствии нажатой кнопки: в обоих случаях множество разрешённых внешних действий одно и то же – пустое множество. Из-за этого невозможно потребовать, чтобы такой переход срабатывал только в том случае, когда никакая кнопка не нажата, или, наоборот, чтобы он срабатывал при нажатии пустой кнопки, но не мог выполняться, если никакая кнопка не нажата. При моделировании в **B**/**O**-семантике τ-переходы $a_∅$—τ→$b_∅$ и a—τ→b возникают всегда одновременно (если был переход a—(τ,∅)→b). Но после такого моделирования в **B**/**O**-семантике мы можем разрешить эту проблему, оставив только один из этих двух τ-переходов.

5.3. Ready-trace семантика

Ready-trace семантика [[9]] отличается от **R**/**Q**-семантики тем, что при возникновении **R**-отказа наблюдается не **R**-отказ, а множество действий, которые реализация может выполнить в данном стабильном состоянии. Это множество называется множеством готовности (***ready set***). Понятно, что если после нажатия **R**-кнопки p наблюдается множество готовности r, то p∩r=∅. Тем самым, наблюдаться может, вообще говоря, не любое множество готовности, а только такое, которое имеют непустое пересечение хотя бы с

одной **R**-кнопкой. После наблюдения множества готовности r нажатие любой кнопки, имеющей с r пустое пересечение, вызовет отказ в том же состоянии. Поэтому имеет смысл нажимать только такие кнопки, которые имеют с r непустое пересечение. После наблюдения действия нажатие **R**-кнопки может вызвать наблюдение действия или множества готовности, а нажатие **Q**-кнопки – наблюдение действия или ненаблюдаемый deadlock.

Таким образом, множества готовности в ready-trace семантике занимают место **R**-отказов в **R**/**Q**-семантике. Трасса готовности – это последовательность действий и множеств готовности, причем после множества готовности может быть только действие, принадлежащее этому множеству. Для получения трасс готовности в каждом стабильном состоянии добавляем переход-петлю по множеству готовности, если это множество имеет пустое пересечение с какой-нибудь **R**-кнопкой. Далее, как обычно, рассматриваются маршруты и в качестве трасс берутся пометки на их переходах с пропуском символа τ.

Безопасность кнопок в реализации (***safe in***) и спецификации (***safe by***) определяется так же, как в **R**/**Q**-семантике, но только после трасс готовности. Аналогично определяются безопасные трассы реализации и спецификации, а также гипотеза о безопасности. Спецификация указывает, какие множества готовности и какие действия могут наблюдаться после тех или иных трасс готовности. Отношение конформности ***resaco*** требует: после общей безопасной трассы готовности спецификации и реализации любое наблюдение (действие или множество готовности), которое возможно в реализации после нажатия кнопки, безопасной после этой трассы по ***safe by*** в спецификации, должно быть после этой трассы и в спецификации. Аналогично определяется и генерация полного набора тестов: в трассы готовности вставляются кнопки: перед действием вставляется безопасная **R**- или **Q**-кнопка, разрешающая это действие, а перед множеством готовности – безопасная **R**-кнопка, имеющая пустое пересечение с этим множеством готовности.

Чтобы представить ready-trace семантику как частный случай **B**/**O**-семантики и конформность ***resaco*** как частный случай общей редукции, прежде всего, будем предполагать, что все её кнопки и наблюдения (действия и наблюдаемые множества готовности) являются кнопками и наблюдениями общей машины тестирования: **R**∪**Q** ⊆ **B** и **L**∪{r⊆**L**|∃p∈**R** r∩p=∅} ⊆ **O**.

Преобразование LTS-реализаций для ready-trace семантики аналогично преобразованию для **R**/**Q**-семантики, но только вместо виртуальных петель по **R**-отказам проводятся виртуальные петли для наблюдаемых множеств готовности:

1) Для каждой кнопки p∈**R**∪**Q** и каждого состояния a добавляем новое состояние a_p и новый переход a—p→a_p.
2) В каждом новом состоянии a_p проводим переход a_p—x→b тогда и только тогда, когда x∈p и имеется переход a—x→b.
3) В каждом новом состоянии a_p проводим переход a_p—r→a тогда и только тогда, когда состояние a стабильно, соответствующее ему множество готовности равно r и r∩p=∅.

4) В каждом новом состоянии a_p проводим переход a_p—τ→b_p тогда и только тогда, когда имеется переход a—τ→b, и проводим переход a_p—γ→b_p тогда и только тогда, когда имеется переход a—γ→b.
5) После этого удаляем все переходы по внешним действиям из старых состояний, оставляя τ- и γ-переходы.

У нас получится LTS, состояния которой делятся на «старые» и «новые», переходы по кнопкам ведут из старых состояний в новые (из a в a_p), переходы по действиям – из новых состояний в старые (из a_p в b), причем только по тем действиям x, которые разрешаются кнопкой p ($x\in p$), которой помечен переход в это новое состояние a_p. Переход по множеству готовности r ведёт из состояния a_r в состояние a. А τ- и γ-переходы ведут как из старых состояний в старые (из a в b), так и из соответствующих новых в соответствующие новые (из a_p в b_p).

Преобразование спецификации, конформность, класс тестируемых реализаций и тестирование через машину тестирования определяются аналогично **R/Q**-семантике.

6. Оптимизация тестов для различных классов реализаций

В предыдущих разделах мы показали, что для **B/O**-семантики и общей редукции на классе всех возможных реализаций имеются только тривиальные зависимости между ошибками, которые легко устраняются процедурой нормализации спецификации. Также мы рассмотрели две гипотезы о безопасности: λ- и γ-гипотезу, которые сужают класс тестируемых реализаций. При этом λ-гипотеза создаёт дополнительную зависимость между ошибками, которая, однако, легко устраняется дополнительной λ-нормализацией, а γ-гипотеза не создаёт дополнительной зависимости между ошибками и дополнительной нормализации не требуется. Далее мы рассмотрели примеры семантик и конформностей, которые сводятся к **B/O**-семантике и общей редукции, но рассматриваются на суженных классах реализаций. Из-за такого сужения возникают уже нетривиальные зависимости между ошибками, что требует нетривиальной оптимизации тестов [[6],[7]].

Всё это можно рассматривать как частный случай общей проблемы сужения класса тестируемых реализаций, которое создаёт зависимости между ошибками и даёт возможность оптимизации тестов. Рассмотрим несколько примеров такого сужения класса тестируемых реализаций, не связанных напрямую с выбором той или иной семантики или гипотезы о безопасности. В этих примерах мы покажем, что такое сужение позволяет использовать конечные полные наборы тестов. Во всех этих примерах предполагается конечность **B/O**-семантики и LTS-спецификации s. Будем считать, что суммарное число кнопок и наблюдений не превосходит числа m, а число состояний детерминированной LTS-спецификации не превосходит числа k.

Первый пример – класс LTS-реализаций с ограниченным числом состояний. Если число состояний реализации не превосходит n, то для полноты

тестирования достаточно ограничиться тестами длиной не более nk. Тогда этот набор тестов содержит не более $O(m^{nk})$ тестов.

Для доказательства этого утверждения достаточно построить композицию LTS реализации и спецификации по следующим правилам. Состояниями композиционной LTS являются пары состояний реализации и спецификации, начальное состояние – пара начальных состояний. Переход (a,b)—x→(a‘,b‘) определяется тогда и только тогда, когда в реализации есть переход a—x→a‘, а в спецификации есть переход b—x→b‘. Реализация неконформна тогда и только тогда, когда в ней есть ошибка 1-го рода, то есть трасса спецификации. Такая трасса в детерминированной спецификации заканчивается в одном состоянии, которое объявлено конечным. В реализации есть такая трасса тогда и только тогда, когда в композиционной LTS из начального состояния достижимо состояние вида (a,b), где b – конечное состояние спецификации. Такое состояние достижимо по простому маршруту (проходящему через каждое состояние не более одного раза), длина которого не превосходит числа достижимых состояний композиционной LTS, которое, в свою очередь, не превосходит общего числа состояний, равного nk. Таким образом, реализация неконформна тогда и только тогда, когда в ней есть ошибочная трасса длиной не более nk. Иными словами, набор всех примитивных тестов длиной не более nk, является полным. Число таких последовательностей в алфавите с m символами, очевидно, равно $O(m^{nk})$.

Второй пример – конечный (с точностью до изоморфизма) класс тестируемых реализаций. Для конечной семантики класс LTS-реализаций с ограниченным числом состояний, очевидно, конечен с точностью до изоморфизма. Поэтому первый пример является частным случаем второго примера. Если семантика и спецификация конечны, то для любого конечного класса реализаций существует конечный полный набор тестов. Для доказательства достаточно заметить, что любой конечный класс реализаций I является подклассом класса реализаций, число состояний которых ограничено числом n, где n – максимальное число состояний реализаций из класса I.

Третий пример – конечный подкласс неконформных реализаций класса тестируемых реализаций. В работах [[14],[15]] такой подкласс называется *классом неисправностей*. Для конечности полноты тестового набора достаточно не конечности класса I тестируемых реализаций, а его подкласса $I\backslash C_s$ неисправностей. Действительно, в каждой неконформной реализации из $i\in I\backslash C_s$ имеется некоторая ошибка 1-го рода, выберем одну такую ошибку σ_i. Набор ошибок $s_I=\{\sigma_i|i\in I\backslash C_s\}$ конечен и, очевидно, является полным тестом, а набор $\{\{\sigma_i\}|i\in I\backslash C_s\}$ примитивных тестов, построенный по этим ошибкам, является полным набором тестов для класса I.

Таким образом, фактически, при тестировании мы пытаемся найти не все ошибки 1-го рода, определяемые спецификацией s, а их конечное подмножество $s_I\subseteq s$. Это эквивалентно тому, что вместо спецификации s мы используем спецификацию s_I. Иными словами, на классе реализаций I спецификации s и s_I эквивалентны. Правда, выполняя тестирование по спецификации s, мы можем быстрее найти ошибку, чем при тестировании по спецификации s_I. Это объясняется тем, что неконформная реализация $i \in I \backslash C_s$ может содержать не только ошибку σ_i, но и какие-то ошибки, не вошедшие в конечный набор s_I. Например, спецификация s может определять как ошибочные некоторые наблюдения с самого начала (до нажатия кнопок): $a,b_1,b_2,b_3,\ldots$, а s_I содержит только одну такую ошибку a. При тестировании мы можем с самого начала ждать наблюдений и, опираясь на спецификацию s, выносим вердикт ***fail***, если получаем любую ошибку $a,b_1,b_2,b_3,\ldots$, но, опираясь на спецификацию s_I, вердикт ***fail*** выносится только для ошибки a.

Эти рассуждения дают четвёртый пример – конечное подмножество ошибок $s_I \subseteq s$ такое, что каждая неконформная (то есть содержащая хотя бы одну ошибку из s) реализация из класса I содержит хотя бы одну ошибку из s_I. Вместо конечного класса неисправностей достаточно использовать просто конечный поднабор набора ошибок, определяемого спецификацией.

Далее напомним, что класс реализаций I определяет ошибки 2-го рода: трассы, которые не встречаются в конформных реализациях класса I, но встречаются в некоторых его неконформных реализациях. Такая ошибка 2-го рода σ может не быть ошибкой 1-го рода, то есть $\sigma \notin s$. Поэтому четвёртый пример является частным случаем последнего, пятого примера, когда для класса тестируемых реализаций I задан конечный набор ошибок (1-го и 2-го родов) s_I такой, что каждая неконформная (то есть содержащая хотя бы одну ошибку из s) реализация из класса I содержит хотя бы одну ошибку из s_I. Этот пример последний, поскольку его условие просто эквивалентно условию существования конечного полного набора тестов. Если такой набор тестов существует, то множество трасс всех тестов набора – это и есть множество ошибок s_I.

7. Обоснование выбранной модели взаимодействия

В этом заключительном разделе статьи мы дадим обоснование выбранной нами модели взаимодействия. Мы рассмотрим шесть вопросов, которые возникают в связи с этой моделью: 1) когда кнопка попадает в трассу: при её нажатии и/или при выполнении LTS-реализацией перехода по кнопке; 2) почему нажатие кнопки блокирует наблюдения; 3) почему оператор должен уметь нажать кнопку достаточно быстро после получения трассы; 4) почему нажатие кнопки не блокирует τ-активность; 5) почему нажатие кнопки блокирует дивергенцию, то есть разрешает только конечную τ-активность; 6) почему переход по каждой кнопке определён в каждом состоянии реализации?

7.1. Когда кнопка попадает в трассу?

Нажатие кнопки выполняется оператором машины тестирования, и в этот момент времени он знает, какая трасса уже получена. Поэтому ничто не мешает оператору отметить тот факт, что он нажал данную кнопку после

наблюдения данной трассы. С другой стороны, поведение реализации, вообще говоря, зависит от той трассы, после которой оператор нажимает кнопку. Поэтому в любом случае при нажатии кнопки она попадает в трассу.

Если при выполнении LTS-реализацией перехода по кнопке p на экране дисплея также появляется кнопка p, то это аналогично тому, как при выполнении реализацией перехода по наблюдению на экране дисплея появляется это наблюдение. Это означает, что выполнение перехода по кнопке p есть, фактически, наблюдение, появление которого в трассе обозначим p‘, чтобы отличить от p, означающего нажатие кнопки p.

Такое наблюдение, в принципе, ничем не отличается от других наблюдений, поэтому режим работы с наблюдаемым переходом по кнопке можно считать частным случаем общей модели взаимодействия. Чтобы в этой модели изобразить такой режим работы, достаточно в LTS-реализации каждый переход по кнопке $a\text{—}p\rightarrow b$ заменить на два перехода с введением дополнительного промежуточного состояния: $a\text{—}p\rightarrow a'\text{—}p'\rightarrow b$.

7.2. Почему нажатие кнопки блокирует наблюдения?

Если нажатие кнопки не блокирует наблюдения, то появляется дополнительная зависимость между трассами реализации (и, следовательно, между ошибками). Поясним это на примере. Пусть при взаимодействии с реализацией может наблюдаться трасса up, где u наблюдение, а p кнопка. Тогда, поскольку наблюдается трасса up, то наблюдается и её префикс – трасса u. Если оператор нажимает кнопку p до наблюдения u, но наблюдения не блокируются этим нажатием, то реализация всё равно может выполнить переход по u. Тем самым, будет наблюдаться трасса pu. Следовательно, если при взаимодействии с реализацией может наблюдаться трасса up, то может наблюдаться и трасса pu.

В выбранной нами модели взаимодействия такой дополнительной зависимости между трассами нет. В то же время режим работы с отсутствием блокировки наблюдений при нажатии кнопки легко моделируется в нашей модели. Для этого достаточно систематически выполнить следующее преобразование реализации, пока оно возможно: если в реализации имеются переходы $a\text{—}p\rightarrow a_p$, $a\text{—}u\rightarrow b$ и $b\text{—}p\rightarrow b_p$, то добавим переход $a_p\text{—}u\rightarrow b_p$. Тем самым, если в состоянии a начиналась трасса up, заканчивающаяся в состоянии b_p, то теперь будет и трасса pu, заканчивающаяся в том же состоянии.

Таким образом, модель взаимодействия с блокировкой наблюдений при нажатии кнопки является более общей. Классу всех реализаций для модели без блокировки соответствует подкласс реализаций, получаемых описанной выше процедурой, для модели с блокировкой. Как и в общем случае, такое сужение класса реализаций приводит к появлению зависимостей между трассами реализации (в частности, между ошибками).

Кроме этого, блокировка наблюдений является следствием приоритета тестового воздействия над наблюдениями. Такой приоритет необходим для того, чтобы можно было моделировать поведение систем с приоритетами. В частности, для прерывания цепочки внешних действий командой «отменить» (cancel).

7.3. Зачем оператору нужно быстро нажимать кнопки?

Прежде всего, отметим, что мы исходим из основного допущения о τ-активности: в реализации τ-активность может быть перед и после любого наблюдения, а также перед и после любого перехода по кнопке. Понятно, что любые ограничения на τ-активность только сузили бы класс рассматриваемых реализаций, что могло бы привести к появлению дополнительных зависимостей между ошибками. Наличие τ-активности ещё не означает, что она обязательно будет выполняться, но, конечно, предполагается, что хотя бы при некотором взаимодействии она выполняется. Естественно, что τ-активность может выполняться, когда никакая кнопка не нажата. Блокирует ли нажатие кнопки τ-активность или нет, мы рассмотрим в следующем пункте.

Также мы хотим, чтобы любой достижимый переход в LTS-реализации мог быть выполнен при том или ином взаимодействии с ней (в зависимости от поведения оператора и погодных условий, моделирующих недетерминированное поведение реализации). Если это не так, и какой-то переход не выполняется ни при каком взаимодействии, то это эквивалентно отсутствию этого перехода в реализации. А это, в свою очередь, приводит к сужению класса рассматриваемых реализаций, что также чревато появлением дополнительных зависимостей между ошибками. Только гипотезы о безопасности запрещают выполнение тех или иных «опасных» переходов в реализации, но, как мы рассмотрели выше, это также приводит к сужению класса реализаций и, возможно, появлению дополнительных зависимостей между ошибками.

Почему мы требуем, чтобы оператор мог нажимать кнопку достаточно быстро после получения трассы, хотя и не обязан это делать всегда? Если оператор не может нажать кнопку достаточно быстро после трассы, то реализация может успеть выполнить после этой трассы один или несколько τ-переходов. Тем самым, переход по кнопке, начинающийся в состоянии до этих τ-переходов, никогда не будет выполнен.

7.4. Почему нажатие кнопки не блокирует ²-активность?

Здесь мы снова опираемся на требование выполнимости каждого достижимого перехода. Если нажатие кнопки блокирует τ-активность, то для того, чтобы реализация могла выполнить некоторую цепочку τ-переходов (а после неё переход по кнопке), оператор не должен нажимать кнопку до тех пор, пока эта цепочка не будет выполнена, и должен нажать кнопку сразу же

после выполнения этой цепочки. Поскольку τ-активность ненаблюдаема, оператор должен просто выждать некоторый интервал времени, прежде чем нажать кнопку. Тем самым, к оператору предъявляются весьма нетривиальные требования по скорости его работы: после получения трассы он должен выдерживать паузу, прежде чем нажимать кнопку, причём длительность этой паузы должна быть, вообще говоря, произвольной в разных сеансах тестирования.

Вместо этого мы выбрали вариант, когда нажатие кнопки не блокирует τ-активность. Тогда к оператору предъявляется только одно требование, рассмотренное в предыдущем пункте: он должен уметь нажимать кнопку достаточно быстро после получения трассы, хотя и не обязан это делать всегда. У реализации появляется выбор: выполнять τ-переход или переход по нажатой кнопке. Как обычно, этот выбор недетерминирован и определяется погодными условиями.

7.5. Почему нажатие кнопки блокирует дивергенцию?

Хотя нажатие кнопки не блокирует τ-активность, но разрешает только конечную τ-активность, то есть разрешает выполнять только конечное число τ-переходов. Тем самым, нажатие кнопки блокирует дивергенцию. Это необходимо для того, чтобы реализовать «выход из дивергенции», то есть приоритет тестового воздействия над дивергенцией.

7.6. Почему переход по каждой кнопке определён в каждом состоянии реализации?

До сих пор мы предполагали, что переход по кнопке определён в каждом состоянии LTS-реализации (по умолчанию отсутствие такого перехода в состоянии трактуется как наличие перехода-петли). На самом деле, это требование не столь принципиально, если его опустить, то это влияет лишь на условие выполнения τ-активности при нажатой кнопке. Новое условие такое: реализация может не выполнить никакого перехода по нажатой кнопке p только в том случае, если она после конечного числа τ-переходов будет бесконечно двигаться по бесконечному τ-маршруту, который проходит только через такие состояния, где нет переходов по кнопке p. В противном случае реализация выполняет конечное число τ-переходов и затем переход по кнопке p.

LTS-реализацию, в которой переходы по кнопкам определены не во всех состояниях, можно промоделировать с помощью LTS, в которой такие переходы есть во всех состояниях. Для этого в исходную LTS-реализацию вносятся следующие изменения.

1. Если переход по кнопке p отсутствует в стабильном состоянии a, то возникает deadlock: реализация не может выполнить переход по p или τ-переход, поскольку их нет, и не может выполнить переход по

наблюдению, поскольку такие переходы блокируются нажатой кнопкой p, а разблокированы могут быть только после перехода по p. Внешне (для оператора машины тестирования) такой deadlock выглядит как отсутствие наблюдений. Из такого deadlock‘а можно выйти, нажав другую кнопку, по которой в стабильном состоянии a есть переход. В нашей модели это реализуется добавлением перехода a—p→a‘, ведущего в новое состояние a‘, в котором определяются переходы-петли a‘—q→a‘ по всем кнопкам q, по которым из состояния a нет переходов, а также переходы по кнопкам, по которым есть переходы из состояния a, ведущие туда же, куда они ведут из состояния a: переход a_p—r→b проводится, когда есть переход a—r→b.

2. Если переход по кнопке p отсутствует в нестабильном состоянии a, в котором не начинается бесконечный τ-маршрут, проходящий только конечное число раз через состояния, где есть переход по кнопке p, то через конечное число τ-переходов будет выполнен какой-нибудь переход по p. Нам достаточно добавить любой переход a—p→b‘, если из состояния a по τ-переходам достижимо состояние b и имеется (или добавлен в п.1) переход b—p→b‘.
3. Если переход по кнопке p отсутствует в дивергентном, состоянии a, в котором начинается бесконечный τ-маршрут, проходящий только конечное число раз через состояния, где есть переход по кнопке p, то возможно, что реализация будет проходить именно этот бесконечный τ-маршрут. В этом случае может не выполниться никакой переход по p, и переходы по наблюдениям останутся заблокированы. Внешне (для оператора машины тестирования) такая дивергенция выглядит как отсутствие наблюдений. Из неё можно выйти, нажав другую кнопку, для которой условие этого пункта не будет выполняться. Это полностью аналогично п.1, при моделировании в нашей модели выполняются те же изменения в реализации, которые описаны в п.1.

Литература

[1] Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, №5.
[2] Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.
[3] Бурдонов И.Б., Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. Труды Института системного программирования РАН, № 14.1, 2008, стр.23-54
[4] Бурдонов И.Б., Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. "Программирование", 2009, №4.
[5] Игорь Бурдонов. Теория конформности (функциональное тестирование программных систем на основе формальных моделей). LAP LAMBERT Academic Publishing, Saarbrucken, Germany, 2011, ISBN 978-3-8454-1747-9.
[6] Бурдонов И.Б., Косачев А.С. Удаление из спецификации неконформных трасс. Препринт Института Системного Программирования РАН, 2011, №23.

[7] Бурдонов И.Б., Косачев А.С. Пополнение спецификации для ***ioco***. "Программирование", 2011, №1.
[8] Bernot G. 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.
[9] 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.
[10] 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.
[11] Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1 // ISO Interim Meeting / ITU-T on, Paris, 1995.
[12] Tretmans J. Conformance testing with labelled transition systems: implementation relations and test generation. Computer Networks and ISDN Systems, v.29 n.1, p.49-79, Dec. 1996.
[13] Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.
[14] Adenilso da Silva Simão, Alexandre Petrenko, Nina Yevtushenko: Generating Reduced Tests for FSMs with Extra States. TestCom/FATES 2009: 129-145.
[15] Alexandre Petrenko, Nina Yevtushenko: Testing from Partial Deterministic FSM Specifications. IEEE Trans. Computers 54(9): 1154-1165 (2005).

Error dependencies on classes of implementations under testing.

Igor Bourdonov <igor@ispras.ru>, Alexander Kossatchev kos@ispras.ru
ISP RAS, Moscow, Russia

Abstract. The paper discusses the problem of dependency between errors defined by specification and the related problem of test optimization. There is a dependency between errors if a strict subset of errors exists such that any nonconforming implementation (i.e. an implementation containing an error) contains an error from this subset. Accordingly, it is sufficient for the tests to detect errors only from this subset. The most general formal model of test interaction and the reduction type of conformance are suggested, for which dependency between errors is almost absent. Most of the known conformances in various interaction semantics are demonstrated to be special cases of this general model. In this general model, the dependency between errors may occur when any strict subset of the class of all implementations is chosen as a class of implementations under testing. Particular interaction semantics and/or various hypotheses on implementations (specifically, the safety hypothesis), in fact, assume that the implementation under testing should belong to some subclass of (safe) implementations.

Keywords: interaction semantics, traces, LTS, conformance, error dependency, test generation, divergence, destruction, safe testing

References

[1]. Bourdonov I., Kossatchev A., Kuliamin V. Formalization of Test Experiments. Programming and Computer Software, Vol. 33, No. 5, 2007, pp. 239-260.

[2]. Bourdonov I., Kossatchev A., Kuliamin V. Teoriya sootvetstviya dlya system s blokirovkami i razrusheniem [Conformance theory of the systems with Refused Inputs and Forbidden Actions]. Moscow, «Nauka», 2008. (in Russian)

[3]. Bourdonov I., Kossatchev A. Sistemy s prioritetami: konformnost', testirovanie, kompozitsiya [Systems with priority: conformance, testing, composition]. Trudy ISP RAN [The proceeding of ISP RAS], Vol. 14.1, 2008. (in Russian)

[4]. Bourdonov I., Kossatchev A. Sistemy s prioritetami: konformnost', testirovanie, kompozitsiya [Systems with priority: conformance, testing, composition]. Programming and Computer Software, Vol. 35, No. 6, 2009,pp. 301-313.

[5]. Bourdonov I. Teoriya konformnosti (funkcional'noe testirovanie prorammny'kh system na osnove formal'ny'kh modelej [Conformance theory (functional testing on formal model base)]. LAP LAMBERT Academic Publishing, Saarbrucken, Germany, 2011, ISBN 978-3-8454-1747-9, 428 pp. http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf) (in Russian

[6]. Bourdonov I., Kossatchev A. Udalenie iz spetsifikatsii nekonformnykh trass [Nonconforming traces elimination from specification]. Preprint № 23, ISP RAN [Preprints of the Institute for System Programming of RAS, Preprint 23], 2011. (in Russian).

[7]. Bourdonov I., Kossatchev A. Specification Completion for IOCO. Programming and Computer Software, Vol. 37, No. 1, 2011, pp. 1-14.

[8]. Bernot G. 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.

[9]. 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.

[10]. 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.

[11]. Revised Working Draft on "Framework: Formal Methods in Conformance Testing". JTC1/SC21/WG1/Project 54/1 // ISO Interim Meeting / ITU-T on, Paris, 1995.

[12]. Tretmans J. Conformance testing with labelled transition systems: implementation relations and test generation. Computer Networks and ISDN Systems, v.29 n.1, p.49-79, Dec. 1996.

[13]. Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.

[14]. Adenilso da Silva Simão, Alexandre Petrenko, Nina Yevtushenko: Generating Reduced Tests for FSMs with Extra States. TestCom/FATES 2009: 129-145.

[15]. Alexandre Petrenko, Nina Yevtushenko: Testing from Partial Deterministic FSM Specifications. IEEE Trans. Computers 54(9): 1154-1165 (2005).