И.Б.Бурдонов, А.С.Косачев.
Системы с приоритетами: конформность, тестирование, композиция.
Труды Института системного программирования РАН, N 14.1, 2008, стр.23-54.
32 стр.

Бурдонов И.Б., Косачев А.С.

Системы с приоритетами: конформность, тестирование, композиция

Аннотация. В статье представлен подход к моделированию компонентов распределенных систем, взаимодействие которых построено на обработке событий с учетом их приоритетов. Несмотря на широкое использование на практике приоритетной обработки запросов или сообщений, математические модели взаимодействия таких программ чаще всего абстрагируются от приоритетов, вводя излишний недетерминизм в описание их поведения. Предложенный подход пытается избежать этого недостатка за счет определения параллельной композиции, моделирующей взаимодействие такого рода. Основное содержание статьи – построение теории формального тестирования компонентов, использующих приоритеты. В рамках этой теории вводятся понятие безопасного выполнения модели и отношение конформности между моделями, а также рассматриваются вопросы построения наборов тестов, проверяющих конформность.

> «Если воняет, то это химия, когда ничего не работает – физика, а если понять нельзя ни слова – математика».
>
> *Из «Законов Мэрфи», цит. по публичной лекции В.И.Арнольда «Сложность конечных последовательностей нулей и единиц и геометрия конечных функциональных пространств».*

Введение

В существующих теориях тестирования конформности (conformance testing) подразумевается отсутствие приоритетов между действиями, которые тестируемая система может выполнять в данной ситуации [23]. Это называется правилом недетерминированного выбора на выполнение одного из таких действий. В то же время для реальных программных и аппаратных систем это правило не всегда адекватно отражает требуемое поведение системы. Рассмотрим несколько примеров.

Выход из дивергенции. Под дивергенцией понимается бесконечная внутренняя активность («зацикливание») системы. Запрос, поступающий извне, может бесконечно долго игнорироваться системой, если он имеет тот же приоритет, что внутренняя активность. Заметим, что внутренняя активность может быть инициирована предыдущим запросом. Если речь идёт о составной системе, собранной из нескольких компонентов, то такая внутренняя активность 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.

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.

Приоритет выдачи над приёмом в неограниченных очередях. Этот обратный пример характерен для неограниченной очереди, используемой в качестве буфера между взаимодействующими системами, в частности, при асинхронном тестировании (тестировании в контексте). 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 асинхронном тестировании 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.

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».

Приоритетная обработка входных воздействий. Если в систему поступает одновременно несколько запросов, то часто требуется их обработка в соответствии с некоторыми приоритетами между ними. Это часто 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.

Отсутствие приоритетов в моделях систем не даёт возможности проверять при тестировании выполнение тех требований к системе, которые могут быть выражены только в форме приоритетов. В данной статье предлагается способ введения приоритетов в теорию конформности: семантику взаимодействия и

модель системы, отношение конформности, методы генерации тестов и оператор композиции (сборки составной системы из взаимодействующих между собой компонентов). Теория конформности без приоритетов кратко описана в нашей статье [13], подробное изложение с доказательствами утверждений содержится в диссертации одного из авторов данной статьи [16], теория конформности для класса, так называемых βγδ-семантик излагается в книге [15]. Здесь мы сначала повторим основные положения этой теории, а затем модифицируем их для случая приоритетов.

1. Теория конформности без приоритетов

1.1. Семантика взаимодействия и безопасное тестирование

Верификация конформности понимается как проверка соответствия исследуемой системы заданным требованиям. В модельном мире система отображается в реализационную модель (реализацию), требования – в спецификационную модель (спецификацию), а их соответствие – в бинарное отношение конформности. Если требования выражены в терминах взаимодействия системы с окружающим миром, возможно тестирование как проверка конформности в процессе тестовых экспериментов, когда тест подменяет собой окружение системы. Само отношение конформности и его тестирование основаны на той или иной модели взаимодействия.

Мы будем рассматривать такие семантики взаимодействия, которые основаны только на внешнем, наблюдаемом поведении системы и не учитывают её внутреннее устройство, отображаемое на уровне модели в понятии *состояния*. В этом случае говорят о тестировании методом «чёрного ящика» или функциональном тестировании. Мы можем наблюдать только такое поведение реализации, которое, во-первых, «спровоцировано» тестом (управление) и, во-вторых, наблюдаемо во внешнем взаимодействии. Такое взаимодействие может моделироваться с помощью, так называемой, машины тестирования [13,15,16,22,23,33]. Она представляет собой «чёрный ящик», внутри которого находится реализация (Рис.1). Управление сводится к тому, что оператор машины, выполняя тест (понимаемый как инструкция оператору), нажимает какие-то кнопки на клавиатуре машины, «приказывая» или «разрешая» реализации выполнять те или иные действия, которые могут им наблюдаться. Наблюдения (на «дисплее» машины) бывают двух типов: наблюдение некоторого *действия*, разрешённого оператором и выполняемого реализацией, и наблюдение *отказа* как отсутствия каких бы то ни было действий из числа тех, что разрешены нажатыми кнопками.

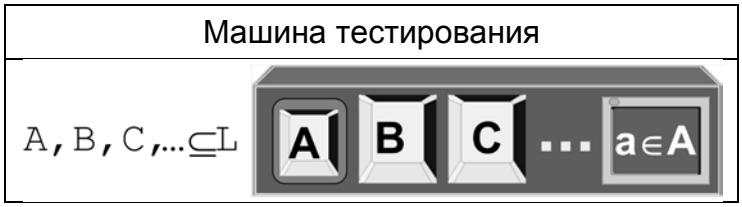


Рис.1.

Следует подчеркнуть, что при управлении оператор разрешает реализации выполнять именно множество действий, а не обязательно одно действие. Например, при тестировании реактивных систем, основанных на обмене стимулами и реакциями, посылка одного стимула из теста в реализацию может интерпретироваться как разрешение реализации выполнять только одно действие – приём этого стимула. Однако приём тестом ответной реакции должен означать разрешение реализации выдавать любую реакцию как раз для того, чтобы проверить, правильна эта реакция или нет. Мы будем считать, что оператор нажимает одну кнопку, но на кнопке «написано», вообще говоря, не одно действие, а множество разрешаемых действий. Когда происходит наблюдение (действие или отказ) кнопка автоматически отжимается, и все внешние действия считаются запрещёнными. Далее оператор может нажимать другую (или ту же самую) кнопку.

В то же время «кнопочное» множество – это, вообще говоря, не любое подмножество множества всех действий. В вопросе о том, какие множества действий могут разрешаться тестом, а какие нет, среди исследователей существует большое разнообразие точек зрения. Например, для реактивных систем обычно считается, что нельзя (или бессмысленно) смешивать посылку стимулов с приёмом реакций (Ян Тритманс). Но существует и прямо противоположный подход: нельзя «тормозить» выдачу реакций реализацией, поэтому, даже посылая стимул, тест должен быть готов к приёму любой реакции (А.Ф. Петренко в [36]).

Также следует подчеркнуть, что наблюдение отказа возможно не при любой кнопке. И здесь разные исследователи опираются на разные предположения. Для тех же реактивных систем долгое время считалось, что тест может наблюдать отсутствие реакций (*quiescence*, стационарность), например, по тайм-ауту, но не видит, принимает реализация посланный ей стимул или нет (*input refusal*, блокировка стимула). С другой стороны, в последние годы появляется всё больше и больше работ, в которых такие блокировки стимулов допускаются или допускаются частично [10-15,26,27,31]. Также и реакции, если они принимаются тестом по разным «выходным каналам», можно принимать не все, а лишь те, которые относятся к одному или нескольким выбранным каналам [26,27].

Итак, семантика взаимодействия определяется тем, какие (в принципе) существуют наблюдаемые действия – алфавит внешних действий L, какие множества действий может разрешать тест – набор кнопок машины тестирования, и для каких из этих кнопок наблюдаемы соответствующие отказы – семейство $\mathfrak{R}\subseteq\mathcal{P}(\mathsf{L})$, а для каких нет – семейство $\mathfrak{Q}\subseteq\mathcal{P}(\mathsf{L})$. Предполагается, что $\mathfrak{R}\cap\mathfrak{Q}=\varnothing$ и $\cup\mathfrak{R}\cup\cup\mathfrak{Q}=\mathsf{L}$. Такую семантику мы называем $\mathfrak{R}/\mathfrak{Q}$-семантикой.

Кроме внешних, наблюдаемых действий реализация может совершать внутренние, ненаблюдаемые (и, следовательно, неразличимые оператором) действия, которые обозначаются символом τ. Выполнение таких действий не регулируется оператором – они всегда разрешены. Предполагается, что любая конечная последовательность любых действий совершается за конечное время, а бесконечная последовательность – за бесконечное время. Бесконечная последовательность τ-действий («зацикливание») называется *дивергенцией* и обозначается символом Δ. Кроме этого мы вводим специальное, не регулируемое кнопками, действие, которое называем *разрушением* и обозначаем символом γ. Оно моделирует любое запрещённое или недекларированное поведение реализации. Например, в терминах пред- и постусловий, поведение программы определено (постусловием) только в том случае, когда выполнено предусловие обращения к ней. Если же предусловие нарушено, поведение программы считается полностью неопределённым. Семантика разрушения предполагает, в частности, что в результате такого поведения система может быть разрушена.

При тестировании мы должны избегать возникновения ненаблюдаемых отказов ($\mathfrak{Q}$-отказов), попыток выхода из дивергенции и разрушения. Такое тестирование называется безопасным. Опасность разрушения подразумевается его семантикой. Поясним остальные случаи. В целом их опасность означает, что после нажатия кнопки оператор может не получить никакого наблюдения, и, не зная об этом, не может ни продолжать тестирование, ни закончить его. Если после нажатия кнопки P возникает $\mathfrak{Q}$-отказ P, оператор не знает, нужно ли ему продолжать ждать наблюдения внешнего действия, разрешённого нажатой кнопкой, или это бессмысленно, поскольку машина стоит. Однако, узнать об остановке машины оператор не может, поскольку это как раз означало бы наблюдение отказа P. Само по себе возникновение дивергенции не опасно, однако, нажимая после этого любую кнопку, оператор, не наблюдая

внешнего действия или $\mathfrak{R}$-отказа (если нажимается $\mathfrak{R}$-кнопка), не знает, случится ли такое наблюдение в будущем, или реализация так и будет бесконечно долго выполнять свои внутренние действия.

Можно также отметить, что из-за внутренних действий нажатие пустой $\mathfrak{R}$-кнопки (кнопки с пустым множеством разрешаемых действий) не эквивалентно отсутствию нажатой кнопки. В обоих случаях все внешние действия запрещены, однако наблюдение отказа означает, что оператор узнаёт об остановке машины, когда она не может выполнять также и внутренние действия. Пустая $\mathfrak{R}$-кнопка не может вызвать разрушение после действия (никакого действия быть не может), но она опасна, если есть дивергенция, как и любая другая кнопка. Пустую $\mathfrak{Q}$-кнопку вообще никогда нельзя нажимать, поскольку никакого наблюдения быть не может: все внешние действия запрещены, а отказ не наблюдаем. Поэтому можно считать, что $\varnothing \notin \mathfrak{Q}$.

1.2. LTS-модель и трассовая модель

В качестве модели реализации и спецификации мы используем *систему помеченных переходов* (LTS – Labelled Transition System). LTS – это ориентированный граф с выделенной начальной вершиной, дуги которого помечены некоторыми символами. Формально, LTS – это совокупность $S=LTS(V_S, L, E_S, s_0)$, где V_S – непустое множество состояний (вершин графа), L – алфавит внешних действий, τ – символ внутреннего действия, γ – символ разрешения, $E_S \subseteq V_S \times (L \cup \{\tau,\gamma\}) \times V_S$ – множество переходов (помеченных дуг графа), $s_0 \in V_S$ – начальное состояние (начальная вершина графа). Переход из состояния s в состояние s' по действию z обозначается $s \text{—}z\to s'$. Обозначим $s \text{—}z\not\to =_{\text{def}} \nexists s'\ s \text{—}z\to s'$. Выполнение LTS, помещённой в «чёрный ящик» машины тестирования, сводится к выполнению того или иного перехода, определённого в текущем состоянии и разрешаемого нажатой кнопкой (τ- и γ-переходы разрешены при нажатии любой кнопки и при отсутствии нажатой кнопки). Состояние называется *стабильным*, если в нём не определены τ- и γ-переходы, и *дивергентным*, если в нём начинается бесконечная цепочка τ-переходов (в частности, τ-цикл, в том числе, τ-петля). Отказ P порождается стабильным состоянием, в котором нет переходов по действиям из P.

Для получения трасс LTS достаточно добавить в каждом стабильном состоянии виртуальные петли, помеченные порождаемыми состоянием отказами, а также

добавить Δ-переходы во всех дивергентных состояниях. После этого рассматриваются все конечные маршруты LTS, начинающиеся в начальном состоянии и не продолжающиеся после Δ- или γ-перехода. Трассой маршрута считается последовательность пометок его переходов с пропуском τ-переходов. Такие трассы мы называем *полными* или ***F*-*трассами***, а множество ***F***-трасс LTS $\mathbf{S}$ – *полной трассовой моделью* или ***F*-*моделью***, и обозначать $\boldsymbol{F}(\mathbf{S})$. ***F***-трасса, все отказы которой принадлежат семейству $\mathfrak{R}$, называется $\mathfrak{R}$-трассой. Это те трассы, которые могут наблюдаться на машине тестирования в $\mathfrak{R}/\mathfrak{Q}$-семантике. Множество всех $\mathfrak{R}$-трасс LTS, то есть проекция её ***F***-модели на алфавит, состоящий из всех внешних действий, $\mathfrak{R}$-отказов, символов Δ и γ, называется $\mathfrak{R}$-моделью, соответствующей «взгляду» на реализацию в $\mathfrak{R}/\mathfrak{Q}$-семантике.

1.3. Гипотеза о безопасности и безопасная конформность

Безопасное тестирование, прежде всего, предполагает формальное определение на уровне модели отношения безопасности «кнопка безопасна в модели после $\mathfrak{R}$-трассы». При безопасном тестировании будут нажиматься только безопасные кнопки. Это отношение различно для реализационной и спецификационной моделей. В LTS-реализации $\mathbf{I}$ отношение безопасности означает, что нажатие кнопки P после $\mathfrak{R}$-трассы σ не может означать попытку выхода из дивергенции (после трассы нет дивергенции), не может вызывать разрушение (после действия, разрешаемого кнопкой), и не может привести к ненаблюдаемому отказу (если это $\mathfrak{Q}$-кнопка):

P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{I}$ ***after*** σ $=_{\text{def}}$ $\forall \mathrm{u}\in\mathrm{P}\ \sigma\cdot\langle \mathrm{u},\gamma\rangle\notin\boldsymbol{F}(\mathbf{I})$ & $\sigma\cdot\langle\Delta\rangle\notin\boldsymbol{F}(\mathbf{I})$.

P ***safe in*** $\mathbf{I}$ ***after*** σ $=_{\text{def}}$ P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{I}$ ***after*** σ & $(\mathrm{P}\in\mathfrak{Q} \Rightarrow \sigma\cdot\langle\mathrm{P}\rangle\notin\boldsymbol{F}(\mathbf{I}))$.

В LTS-спецификации $\mathbf{S}$ отношение безопасности отличается только для $\mathfrak{Q}$-кнопок: мы не требуем, чтобы трасса σ не продолжалась безопасным $\mathfrak{Q}$-отказом Q, но требуем, чтобы она продолжалась хотя бы одним действием $\mathrm{z}\in\mathrm{Q}$. Кроме того, если действие разрешается хотя бы одной неразрушающей $\mathfrak{Q}$-кнопкой, то оно должно разрешаться какой-нибудь безопасной кнопкой. Если это неразрушающая $\mathfrak{R}$-кнопка, то она же и безопасна. Но если все неразрушающие кнопки, разрешающие действие, являются $\mathfrak{Q}$-кнопками, то хотя бы одна из них должна быть объявлена безопасной. Такое отношение

безопасности всегда существует: достаточно объявить безопасной каждую неразрушающую кнопку, разрешающую действие, продолжающее трассу. Однако, в целом указанные требования неоднозначно определяют отношение ***safe by***, и при задании спецификации указывается конкретное отношение. Требования к отношению ***safe by*** записываются так: $\forall R{\in}\mathfrak{R}\ \forall z{\in}L\ \forall Q{\in}\mathfrak{Q}$

1) R ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Leftrightarrow R$ ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** σ,

2) $\exists T{\in}\mathfrak{Q}\ T$ ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** $\sigma\ \&\ z{\in}T\ \&\ \sigma{\cdot}\langle z\rangle{\in}\boldsymbol{F}(\mathbf{S})$
$\Rightarrow \exists P{\in}\mathfrak{R}{\cup}\mathfrak{Q}\ z{\in}P\ \&\ P$ ***safe by*** $\mathbf{S}$ ***after*** σ,

3) Q ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow Q$ ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** $\sigma\ \&\ \exists v{\in}Q\ \sigma{\cdot}\langle v\rangle{\in}\boldsymbol{F}(\mathbf{S})$.

Безопасность кнопок определяет безопасность действий и $\mathfrak{R}$-отказов после $\mathfrak{R}$-трассы. $\mathfrak{R}$-отказ R безопасен, если после трассы безопасна кнопка R. Действие безопасно, если оно разрешается некоторой кнопкой, безопасной после трассы. Теперь мы можем определить *безопасные трассы*. $\mathfrak{R}$-трасса безопасна, если 1) модель не разрушается с самого начала (сразу после включения машины ещё до нажатия первой кнопки), то есть в ней нет трассы $\langle\gamma\rangle$, 2) каждый символ трассы безопасен после непосредственно предшествующего ему префикса трассы. Множества безопасных трасс реализации $\mathbf{I}$ и спецификации $\mathbf{S}$ обозначим $\boldsymbol{SafeIn}(\mathbf{I})$ и $\boldsymbol{SafeBy}(\mathbf{S})$, соответственно.

Требование безопасности тестирования выделяет класс *безопасных* реализаций, то есть таких, которые могут быть безопасно протестированы для проверки их конформности или неконформности заданной спецификации. Этот класс определяется следующей *гипотезой о безопасности*: реализация $\mathbf{I}$ *безопасна* для спецификации $\mathbf{S}$, если 1) в реализации нет разрушения с самого начала, если этого нет в спецификации, 2) после общей безопасной трассы реализации и спецификации любая кнопка, безопасная в спецификации, безопасна после этой трассы в реализации:

$$\mathbf{I}\ \textbf{\textit{safe for}}\ \mathbf{S} =_{\text{def}} \quad (\langle\gamma\rangle{\notin}\boldsymbol{F}(\mathbf{S}) \Rightarrow \langle\gamma\rangle{\notin}\boldsymbol{F}(\mathbf{I}))$$
$$\&\ \forall\sigma{\in}\boldsymbol{SafeBy}(\mathbf{S}){\cap}\boldsymbol{SafeIn}(\mathbf{I})\ \forall P{\in}\mathfrak{R}{\cup}\mathfrak{Q}$$
$$(P\ \textbf{\textit{safe by}}\ \mathbf{S}\ \textbf{\textit{after}}\ \sigma \Rightarrow P\ \textbf{\textit{safe in}}\ \mathbf{I}\ \textbf{\textit{after}}\ \sigma).$$

Следует отметить, что гипотеза о безопасности не проверяема при тестировании и является его предусловием. После этого можно определить отношение (безопасной) *конформности*: реализация $\mathbf{I}$ *безопасно конформна*

(или просто *конформна*) спецификации **S**, если она безопасна и выполнено *тестируемое условие*: любое наблюдение, возможное в реализации в ответ на нажатие безопасной (в спецификации) кнопки, разрешается спецификацией:

$$\mathbf{I}\ \textit{saco}\ \mathbf{S} =_{\text{def}} \mathbf{I}\ \textit{safe for}\ \mathbf{S} \ \&\ \forall\sigma\in\textit{SafeBy}(\mathbf{S})\cap\textit{SafeIn}(\mathbf{I})\ \forall \mathtt{P}\ \textit{safe by}\ \mathbf{S}\ \textit{after}\ \sigma\ \textit{obs}(\sigma,\mathtt{P},\mathbf{I})\subseteq\textit{obs}(\sigma,\mathtt{P},\mathbf{S}),$$

где $\textit{obs}(\sigma,\mathtt{P},\mathbf{T}) =_{\text{def}} \{\mathtt{u}\mid\sigma\cdot\langle\mathtt{u}\rangle\in\boldsymbol{F}(\mathbf{T})\ \&\ (\mathtt{u}\in\mathtt{P}\ \vee\ \mathtt{u}=\mathtt{P}\ \&\ \mathtt{P}\in\mathfrak{R})\}$ – множество наблюдений, которые можно получить над моделью **T** при нажатии кнопки P после трассы σ.

1.4. Параллельная композиция и генерация тестов

Взаимодействие двух систем моделируется в LTS-теории оператором параллельной композиции. Мы используем оператор композиции, аналогичный тому, который определяется в алгебре процессов CCS (Calculus of Communicating Systems) [32,34]. Будем считать, что для каждого внешнего действия z определено противоположное действие $\underline{z}$ так, что $\underline{\underline{z}}=z$. Например, посылке стимула из теста соответствует приём теста в реализации, а выдаче реакции реализацией соответствует приём этой реакции в тесте. Параллельное выполнение двух LTS в алфавитах A и B понимается так, что переходы по противоположным действиям z и $\underline{z}$, где $z\in A$ и $\underline{z}\in B$, выполняются синхронно, то есть, в обеих LTS одновременно, причём в композиции это становится τ-переходом. Такие действия называются синхронными. Остальные внешние действия $z\in A\setminus\underline{B}$ и $z\in B\setminus\underline{A}$, а также символы τ и γ называются асинхронными. Переход по такому символу выполняются в одной из LTS при сохранении состояния другой LTS. Результатом композиции двух LTS **I** и **T** становится LTS $\mathbf{I}\between\mathbf{T}$ в алфавите $A\between B =_{\text{def}} (A\setminus\underline{B})\cup(B\setminus\underline{A})$. Её состояния – это пары состояний it LTS-операндов, начальное состояние – это пара начальных состояний, а переходы порождаются следующими правилами вывода:

(1) $z\in\{\tau,\gamma\}\cup A\setminus\underline{B}$ & $i\text{—}z\to i'$ $\vdash$ $it\text{—}z\to i't$,

(2) $z\in\{\tau,\gamma\}\cup B\setminus\underline{A}$ & $t\text{—}z\to t'$ $\vdash$ $it\text{—}z\to it'$,

(3) $z\in A\cap\underline{B}$ & $i\text{—}z\to i'$ & $t\text{—}\underline{z}\to t'$ $\vdash$ $it\text{—}\tau\to i't'$.

Тестирование понимается как замкнутая композиция LTS-реализации **I** в алфавите A и LTS-теста **T** в противоположном алфавите $B=\underline{A}$. Для обнаружения отказов в тесте (но не в реализации!) допускаются специальные θ-

переходы, которые срабатывают тогда и только тогда, когда никакие другие переходы не могут выполняться:

(4) $t \text{—}\theta\rightarrow t` \;\&\; \boldsymbol{Deadlock}(i,t) \;\vdash it \text{—}\tau\rightarrow it`,$

где $\boldsymbol{Deadlock}(i,t) = i \text{—}\tau\not\rightarrow \;\&\; i \text{—}\gamma\not\rightarrow \;\&\; t \text{—}\tau\not\rightarrow \;\&\; t \text{—}\gamma\not\rightarrow$

$\&\; (\forall z \in A \cap \underline{B}\; i \text{—} z\not\rightarrow \vee\; t \text{—} \underline{z}\not\rightarrow).$

Мы будем предполагать, что в тесте также нет разрушения (требование $t \text{—}\gamma\not\rightarrow$ всегда выполнено).

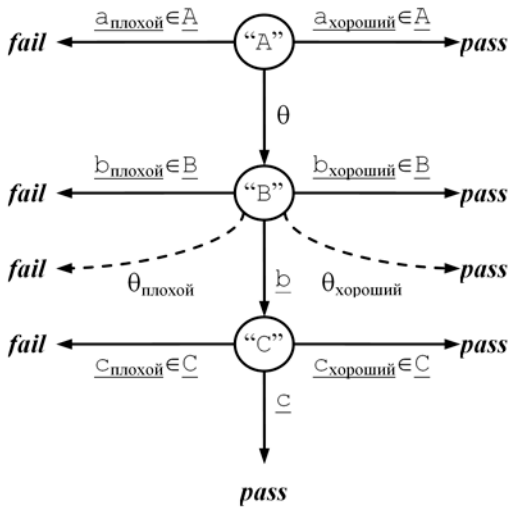
Поскольку алфавиты реализации и теста противоположны, композиционный алфавит пуст и в композиционной LTS есть только τ- и γ-переходы. При безопасном тестировании γ-переходы недостижимы. Выполнению теста соответствует прохождение τ-маршрута, начинающегося в начальном состоянии композиции $\mathbf{I}|\lfloor\mathbf{T}$. Тест заканчивается, когда достигается терминальное состояние теста. Каждому такому терминальному состоянию назначается вердикт ***pass*** или ***fail***. Реализация *проходит* тест, если состояния теста с вердиктом ***fail*** недостижимы. Реализация проходит набор тестов, если она проходит каждый тест из набора. Набор тестов *значимый*, если каждая конформная реализация его проходит; *исчерпывающий*, если каждая неконформная реализация его не проходит; *полный*, если он значимый и исчерпывающий. Задача заключается в генерации полного набора тестов по спецификации.

Обычно ограничиваются, так называемыми, *управляемыми* тестами, то есть тестами без лишнего недетерминизма. Для этого множество внешних действий, для которых определены переходы в данном состоянии теста, должно быть одним из «кнопочных» множеств $\mathfrak{R}/\mathfrak{Q}$-семантики (точнее, множеством противоположных действий, поскольку при композиции CCS тест определяется в противоположном алфавите). Оператор, исполняя тест, однозначно определяет, какую кнопку ему нужно нажимать в данном состоянии теста. Если это $\mathfrak{R}$-кнопка, то в состоянии должен быть также определён θ-переход.

Полным набором всегда является набор всех *примитивных* тестов. Примитивный тест строится по одной выделенной безопасной $\mathfrak{R}$-трассе спецификации. Для этого сначала в трассу перед каждым $\mathfrak{R}$-отказом R вставляется кнопка R, а перед каждым действием z – какая-нибудь безопасная (после префикса трассы) кнопка P, разрешающая действие z. Для различения кнопок и отказов (и то и другое – подмножества внешних действий)

мы будем кнопки заключать в кавычки и писать “P”, а не просто P. Безопасность трассы гарантирует безопасность кнопки R и наличие такой безопасной кнопки P. Выбор кнопки P может быть неоднозначным, то есть по одной безопасной трассе спецификации можно сгенерировать, вообще говоря, множество разных примитивных тестов. После расстановки кнопок получается последовательность, которая во втором разделе статьи называется $\mathfrak{R}/\mathfrak{Q}$-историей. По ней и строится LTS-тест (Рис.2). Его состояниями становятся расставленные кнопки, начальное состояние – это первая в трассе кнопка, символы переходов из состояния-кнопки – это действия, противоположные тем, которые могут наблюдаться после нажатия этой кнопки, или символ θ, если это $\mathfrak{R}$-кнопка. Если это не последняя кнопка, то один переход ведёт в состояние, соответствующее следующей кнопке. Остальные переходы ведут в терминальные состояния. Вердикт ***pass*** назначается тогда, когда соответствующая $\mathfrak{R}$-трасса есть в спецификации, а вердикт ***fail*** – когда нет. Такой вердикт соответствует *строгим* тестам, которые, во-первых, значимые (не ловят ложных ошибок) и, во-вторых, не пропускают обнаруженных ошибок. Любой строгий тест можно заменить на объединение примитивных тестов, которое обнаруживает те же самые ошибки.

Примитивный тест для безопасной $\mathfrak{R}$-трассы σ



σ=⟨A,b,c⟩⇒вставляем ,безопасные кнопки⇒

⇒⟨“A”,A,“B”,b,“C”,c⟩ такие, что:

A, B∈$\mathfrak{R}$, C∈$\mathfrak{Q}$, b∈B, c∈C.

Внешнее действие имеет индекс «хорошо», если оно есть в спецификации после префикса трассы; иначе – индекс «плохо».

Пунктиром показаны взаимоисключающие θ-переходы, ведущие из одного состояния. Индексы «хорошо» и «плохо» означают наличие или отсутствие продолжения префикса трассы соответствующим $\mathfrak{R}$-отказом.

Рис.2.

Теперь рассмотрим две проблемы, связанные с применением теории тестирования конформности на практике.

1.5. Недетерминизм и глобальное тестирование

Как уже было сказано, в каждый момент времени реализация может выполнять любое определённое в ней и разрешённое оператором внешнее действие, а также определённые и всегда разрешённые внутренние действия (мы избегаем разрушения при безопасном тестировании). Если таких действий несколько, выбирается одно из них недетерминированным образом. Здесь предполагается, что недетерминизм поведения реализации – это явление того уровня абстракции, которое определяется нашими тестовыми возможностями по наблюдению и управлению, то есть семантикой взаимодействия. Иными словами, поведение реализации недетерминировано, поскольку оно зависит от неких не учитываемых нами факторов – «погодных условий», которые определяют выбор выполняемого действия детерминировано.

Для того, чтобы тестирование могло быть полным, мы должны предположить, что любые погодные условия могут быть воспроизведены в тестовом эксперименте, причём для каждого теста. Если такая возможность есть, тестирование называется *глобальным* [33]. Мы абстрагируемся от количества вариантов погодных условий. Здесь нам важна только потенциальная возможность проверить поведение системы при любых погодных условиях и любом поведении оператора. Конечно, на практике используется только конечное число прогонов каждого теста. Без дополнительных условий мы не можем быть уверены, что провели тестовые испытания каждого теста для всех возможных погодных условий. Возможны различные решения этой проблемы.

Одно из них – специальные тестовые возможности по управлению погодой. Для этого мы должны выйти за рамки модели, которая как раз и абстрагировалась от второстепенных деталей внешних факторов, то есть от погоды. Тем самым, тестирование становится зависящим не только от спецификации, но и от реализационных деталей, от того, что можно назвать операционной обстановкой, в которой работает реализация. Для каждого варианта такой операционной обстановки мы будем вынуждены создавать свой набор тестов. Тем не менее, в некоторых частных случаях на этом пути можно получить практические выгоды.

Другое решение – специальные реализационные гипотезы. Для конечного числа прогонов теста предполагают, что, если реализация ведёт себя правильно при некоторых погодных условиях, то она будет вести себя правильно при любых погодных условиях [6].

Третье решение основано на том, что нам известно распределение вероятностей тех или иных погодных условий. В этом случае тестирование оказывается полным с той или иной вероятностью [19].

Близкое к этому четвёртое решение предполагает, что в каждой ситуации (после трассы) возможно лишь конечное число погодных условий (с точностью до эквивалентности) и существует такое число N, что после N прогонов теста гарантированно будет проверено поведение реализации при всех возможных в этой ситуации погодных условиях [21,32].

Наконец, существует и более радикальное решение – просто запретить недетерминизм реализации, то есть реализационная гипотеза ограничивает класс реализаций только детерминированными реализациями. При всей своей наивности, это достаточно распространённый практический приём [35] (он применялся и в ИСП РАН в рамках системы UniTESK). Обоснованием может служить то, что во многих случаях заранее известно, что интересующие нас реализации детерминированы.

1.6. Бесконечность полного набора тестов

На практике используются только конечные наборы конечных (по времени выполнения) тестов. Поскольку тесты конечные, полный набор, как правило, содержит бесконечное число тестов (полный набор конечен только для моделей с конечным поведением, то есть конечным числом трасс; в частности, в LTS-спецификации не должно быть циклов). Возможны различные решения этой проблемы. В конечном счёте все они сводятся к специальным реализационным гипотезам. Такие гипотезы, по сути, предполагают ровно то, что хотелось бы «доказать»: если реализация ведёт себя правильно на тестах данного конечного набора, то она будет вести себя правильно на всех тестах полного набора. Обоснованием может служить некоторое (не важно как полученное) «знание» о том, как могут быть устроены тестируемые реализации (тестируем не все возможные реализации, а только такие). На классе всех реализаций такие конечные наборы тестов будут только значимыми (не ловят ложных ошибок), а полными – только на подклассе, определяемом реализационной гипотезой.

Конечный набор тестов строится по тому или иному *критерию покрытия*, чтобы проверить все интересующие нас классы ситуаций (ошибок) [24,25,37]. Теоретически конечный набор можно получить фильтрацией по критерию покрытия перечислимого полного набора. Поэтому теория конформности должна быть развита до уровня алгоритмов перечисления полного набора. Однако на практике обычно используются более прямые методы построения нужного конечного набора. Достаточно общий подход сводится к тому, что

вместо исходной спецификационной модели используется более грубая, так называемая, тестовая модель. Тестовая модель – это результат факторизации исходной LTS-спецификации по отношению эквивалентности переходов, что обычно сводится к эквивалентности состояний и/или действий [1]. Иногда при факторизации исчезает недетерминизм, что заодно решает и эту проблему. Разумеется, чтобы такой подход был оправданным, нужны мотивированные реализационные гипотезы о том, что ошибки, возможные в реализации, обнаруживаются при тестировании по факторизованной спецификации (вообще по конечному набору, удовлетворяющему критерию покрытия).

Примером практического тестирования может служить тестирование конечного автомата по спецификации, заданной также в виде конечного автомата [3,4,6,18,28,29]. Если у нас есть специальная операция, позволяющая достоверно и напрямую опросить текущее состояние реализации (*status message*), то, как известно, полное тестирование сводится к обходу графа переходов автомата и применению операции опроса в каждом проходимом состоянии [3,6-9,20,29]. Обход графа переходов используется также в случае тестирования методом «чёрного ящика», когда состояния реализации не видны. Но здесь для полноты тестирования требуются реализационные гипотезы, компенсирующие отсутствие тестовой возможности достоверного опроса состояний [3,4,6]. Предполагается, что число состояний реализации не превосходит число состояний спецификации (с точностью до эквивалентности состояний) или превосходит не более, чем на заранее известную величину [17,18,29,30]. Этот подход переносится и на общий случай LTS для реактивных систем [5]. В частности, когда используется, так называемое, стационарное тестирование, при котором стимулы подаются в реализацию только в её стационарных состояниях (в этом случае также снимается проблема торможения реакций) [2].

2. Теория конформности с приоритетами

2.1. Предикаты на переходах LTS-модели

Независимо от наличия или отсутствия приоритетов семантика взаимодействия предполагает, что выполняться может только то действие, которое определено в реализации и разрешено оператором машины тестирования. Если приоритетов нет, то выполняться может любое определённое и разрешённое действие, и выбор выполняемого действия не детерминирован. Наличие приоритетов означает, что не все определённые и разрешённые действия могут выполняться, то есть выполнимость действия зависит также от того, какие ещё действия определены и/или разрешены. Эту зависимость можно изобразить в виде предиката от множества разрешённых действий, который назначается

переходу LTS-модели. Поскольку для перехода $s—z→s`$ известно его пресостояние s, а для этого состояния s известно, какие ещё переходы в нём начинаются, предикат на переходе можно считать не зависящим от множества определённых (в состоянии s) действий. Иными словами, переходы по одному и тому же действию, ведущие из разных состояний, могут иметь разные предикаты.

LTS-модель с приоритетами – это LTS, алфавит которой – это декартовое произведение алфавита внешних действий и множества предикатов на алфавите внешних действий $П=\{\pi:\mathcal{P}(L)→\boldsymbol{Bool}\}$: $\mathbf{S}=LTS(V_s,L×П,E_s,s_0)$. Переход $s—z,\pi→s`$ может выполняться только тогда, когда оператор разрешает такое множество внешних действий $R⊆L$, что $z∈R∪\{\tau,\gamma\}$ и $\pi(R)=\boldsymbol{true}$. Если есть несколько выполнимых действий, выполняется одно из них, выбираемое, по-прежнему, недетерминированным образом.

Предикат можно понимать как булевскую функцию от булевских переменных $z_1,z_2,…$, взаимно-однозначно соответствующих внешним действиям из алфавита L. Например, для предиката $\pi=a\&¬b∨c$ переход $s—z,\pi→s`$ может выполняться только тогда, когда оператор разрешил такое множество внешних действий R, что $z∈R∪\{\tau,\gamma\}\ \&\ (a∈R\&b∉R∨c∈R)$. Это означает, что действие z – это внутреннее действие, разрушение или внешнее действие, разрешённое оператором, а также выполнено хотя бы одно из двух условий: 1) оператор разрешил действие a и не разрешил действие b, 2) оператор разрешил действие c.

Итак, в общем случае предикат – это булевская функция от множества разрешённых действий. Можно отметить важный частный случай, когда предикат зависит только от разрешённых *и определённых* внешних действий. Иными словами, предикат на переходе $s—z,\pi→s`$ не зависит от тех булевских переменных, которые соответствуют внешним действиям, по которым нет переходов из состояния s. Это означает, что выполнимость перехода зависит только от того, разрешено ли действие z, и какие ещё действия определены в состоянии s и разрешены оператором. В этом случае реализацию не интересуют (она «не видит») те действия, которые оператор разрешает, но они всё равно не могут выполняться, поскольку не определены в текущем состоянии реализации. Нажимая кнопку, оператор как бы «подсвечивает» некоторые действия реализации, определённые в её текущем

состоянии, и выполнимость перехода по каждому из них определяется соответствующим предикатом от множества «подсвеченных» действий.

Предикат π как булевская функция от булевских переменных-действий может быть представлен в виде совершенной дизъюнктивной нормальной формы (СДНФ) $\pi=\eta_1\vee\eta_2\vee\ldots$, где $\eta_i=x_{i1}\&x_{i2}\&\ldots$, $x_{ij}=z_j$ или $x_{ij}=\neg z_j$, и z_j пробегает множество всех внешних действий. Тогда переход $s\text{—}z,\pi\to s'$ можно заменить на множество кратных переходов с предикатами-дизъюнктами $s\text{—}z,\eta_i\to s'$. В свою очередь дизъюнкту η_i взаимно-однозначно соответствует множество P_i тех действий, для которых $x_{ij}=z_j$. При композиции это множество является множеством действий, разрешаемых партнёром. В случае, когда таким партнёром является тест для заданной $\mathfrak{R}/\mathfrak{Q}$-семантики, эти множества разрешаемых действий соответствуют кнопкам из $\mathfrak{R}\cup\mathfrak{Q}$. Тем самым, при тестировании мы можем считать, что на переходах написаны не произвольные предикаты, а кнопки $s\text{—}z,P_i\to s'$. Когда нажимается некоторая кнопка P_i, выполняться могут только переходы вида $s\text{—}z,P_i\to s'$, где $z\in P_i\cup\{\tau,\gamma\}$. Такой переход от LTS с предикатами к LTS с кнопками аналогичен переходу от расширенных автоматов (EFSM – Extended Finite State Machine) к обычным автоматам (FSM).

2.2. Остановка и дивергенция

Машина без приоритетов останавливается в стабильном состоянии (состоянии без τ- и γ-переходов), в котором нет переходов по разрешённым внешним действиям. Если есть приоритеты, то меняется, прежде всего, само понятие стабильности. Оно становится условным: состояние стабильно, если для всех переходов из этого состояния их предикаты от пустого множества разрешённых внешних действий ложны $\pi(\varnothing)=\textit{false}$. Соответственно меняется условие остановки машины: машина стоит, если при разрешённом множестве внешних действий P для всех переходов из этого состояния их предикаты ложны $\pi(P)=\textit{false}$.

Здесь мы должны уточнить, что происходит, когда кнопка отжимается. Для машины без приоритетов кнопка автоматически отжимается при любом наблюдении действия или отказа. После действия машина может выполнять любые τ-переходы (а также γ-переход), но после отказа машина стоит, поскольку отказ происходит в стабильном состоянии, в котором нет τ- и γ-переходов. Однако для машины с приоритетами отжатие кнопки меняет

множество разрешённых действий (если только не была нажатой пустая $\mathfrak{R}$-кнопка, дающее единственное наблюдение пустого отказа). После наблюдения реализация начинает выполнять τ-переходы с приоритетом $\pi(\varnothing)=\mathit{true}$. Заметим, что таким наблюдением может быть не только действие, но и отказ. Причина этого в том, что отказ P означал невозможность выполнения разрешённых внешних действий $z\in P$, а также τ- и γ-переходов, поскольку их предикаты стали ложны $\pi(P)=\mathit{false}$. После отжатия кнопки P множество разрешённых внешних действий пусто и теперь могут выполняться τ- и γ-переходы с предикатами $\pi(\varnothing)=\mathit{true}$. Далее оператор может снова нажать ту же кнопку (но без гарантии повторного наблюдения того же отказа, если реализация сменила состояние по τ-переходам) или другую кнопку.

Если допускается переключение кнопок, то есть нажатие второй кнопки, не дожидаясь наблюдения по первой кнопке, то это интерпретируется как отжатие первой кнопки, а потом нажатие второй кнопки. Мы будем считать, что «в промежутке» между двумя кнопками создаётся ситуация, когда ни одна кнопка не нажата, и реализация может выполнять τ- и γ-переходы с предикатами $\pi(\varnothing)=\mathit{true}$. Общая парадигма здесь заключается в том, что ситуация отсутствия тестового воздействия возникает всегда при включении машины (до нажатия первой кнопки), после любого наблюдения и между двумя тестовыми воздействиями. Более подробно переключение кнопок рассматривается в следующем подразделе.

Мы уже говорили, что даже для машины без приоритетов проблема дивергенции не в ней самой по себе, а в выходе из неё. При наличии приоритетов, если внешнее воздействие имеет больший приоритет, чем внутренняя активность, дивергенция прекращается. Теперь выполнимость τ-действий зависит от нажатой кнопки, и мы можем косвенно управлять ими и, следовательно, дивергенцией. Тогда можно говорить о *выполнимой* дивергенции: при одной нажатой кнопке (или когда нет нажатой кнопки) все τ-действия бесконечной цепочки выполнимы, а при другой – нет и, следовательно, нет «зацикливания». Выйти из дивергенции, которая начинает выполняться после кнопки A, можно с помощью кнопки B, при которой дивергенция не выполнима. Заметим, что для этого требуется переключение кнопок, то есть нажатие кнопки без наблюдения (которого может не быть). Единственный случай, когда из дивергенции нельзя гарантированно выйти, – это когда дивергенция выполнима при нажатии любой кнопки.

2.3. Переключение кнопок

В машине без приоритетов кнопку можно нажимать либо после включения машины, либо после того, как произошло наблюдение по предыдущей кнопке. Иными словами, запрещается переключать кнопки без наблюдения, отжимая одну кнопку и нажимая другую. Этот запрет объясняется тем, что, если приоритетов нет, возможность переключения кнопок не увеличивает мощность тестирования. Действительно, если была нажата кнопка `P`, а потом без наблюдения нажата другая кнопка `Q` (а кнопка `P` отжата), то в этом интервале времени реализация могла выполнять только τ-действия. Но τ-действия всегда разрешены, поэтому реализация могла бы выполнять их и в том случае, когда вместо кнопки `P` сразу нажималась кнопка `Q` (а второй раз, естественно, не нажималась). Тем самым, любое поведение, которое можно наблюдать в первом случае, можно было бы наблюдать и во втором случае.

При наличии приоритетов переключение без наблюдения необходимо для полноты тестирования, поскольку различные множества разрешённых действий по-разному влияют на выполнение τ-действий (τ-переходы тоже могут иметь предикаты), что приводит к внешне различимым поведениям. Например, если в реактивной системе приём стимула приоритетнее выдачи реакций и τ-действий, последние выполняются только тогда, когда реализация не может принять стимул, посылаемый ей тестом. На Рис.3 показан пример, где для получения тестом реакции `!y` после стимула `?a` нельзя сразу посылать этот стимул (тогда после него будет реакция `!x`), а нужно сначала послать стимул `?b`, а потом переключить кнопку `{?b}` на кнопку `{?a}`, послав тем самым стимул `?a`. Если реализация принимает стимул `?b`, то переключение нужно успеть сделать до приёма стимула `?b`. Если же реализация блокирует стимул `?b` (нет пунктирного перехода), то можно «не торопиться»; если блокировка `{?b}` наблюдаема, можно сначала её дождаться, а потом послать стимул `?a`.

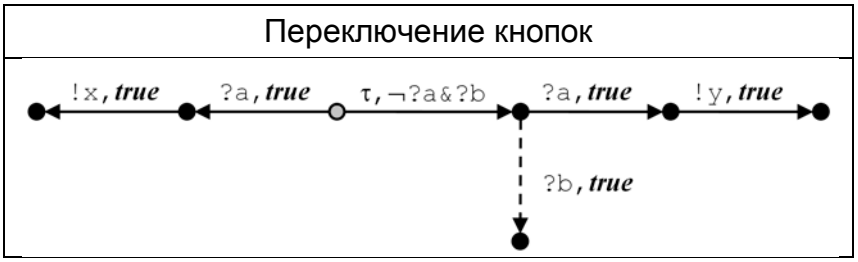


Рис.3.

Тем не менее, в пользу запрета на переключение кнопок имеются разумные аргументы и в случае наличия приоритетов. Дело в том, что переключение кнопок позволяет «обходить» ненаблюдаемый отказ: если оператор

переключает Ω-кнопку P на (любую) другую кнопку Q, то возникновение ненаблюдаемого отказа P не препятствует такому переключению (как на Рис.3, когда нет пунктирного перехода по стимулу ?b, а блокировка {?b} не наблюдаема). Другое дело, что, если возможен отказ P, то при безопасном тестировании мы не можем нажимать кнопку P без последующего переключения на другую кнопку, то есть с ожиданием наблюдения (на Рис.3 кнопку {?b} всегда нужно переключать на кнопку {?a} или какую-то другую). Тем самым, если можно переключать кнопки, условие безопасности Ω-кнопок более сложное (ниже мы его подробно рассмотрим). Если переключений кнопок нет, тестирование выглядит более привычно как чередующаяся последовательность тестовых воздействий (нажимается кнопка) и наблюдений. Кроме того, в этом случае к работе оператора предъявляется меньше временных требований.

2.4. Временные ограничения на работу оператора (теста)

Введение приоритетов усложняет работу оператора, налагая более сложные требования по времени. Если приоритетов нет, то оператор должен уметь достаточно быстро нажимать кнопку после включения машины или после предыдущего наблюдения. Заметим, что если оператор не успевает достаточно быстро нажать кнопку, ничего страшного не случится, поскольку машина успеет выполнить только одно или несколько τ-действий, которые (в машине без приоритетов) она может выполнить и в том случае, когда кнопка была нажата немедленно. Иными словами, мы требуем, чтобы оператор мог работать быстро, но не заставляем его всегда работать быстро.

Если приоритеты есть, то возможность наблюдения тех или иных поведений реализации требует не только достаточно быстрой скорости работы оператора, но также достаточно медленной, средней и т.д. В примере на Рис.4 стимул ?a может приниматься в трёх состояниях 1, 2 и 3, но реакции после этого различны: !x, !y или !z. Эти состояния связаны τ-переходами, которые выполнимы только, если тест не посылает стимул ?a. Поэтому реакция !x будет наблюдаться только, если оператор быстро нажмёт кнопку {?a}, реакция !z – только если оператор не будет торопиться, а реакция !y – только при средней скорости работы.

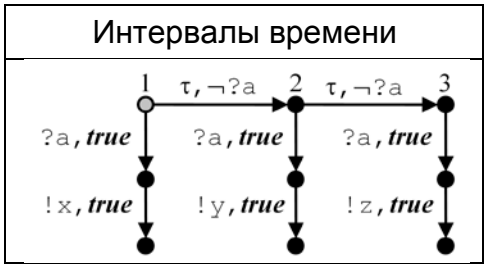


Рис.4.

Если есть переключение кнопок, то такое переключение также нужно уметь делать с различными интервалами времени, чтобы «заставить» реализацию проходить нужное число τ-переходов между двумя кнопками.

Таким образом, для машины с приоритетами следует учитывать временные задержки, которые делает оператор между наблюдением и последующим нажатием кнопки или между двумя нажатиями кнопок при их переключении без наблюдения. Мы можем считать, что в погодные условия включены также те факторы, которые влияют на «свободу воли» оператора, определяя те или иные временные задержки при нажатии кнопок. Это согласуется с тем, что оператор должен моделировать любую скорость работы окружения. Работа оператора моделирует выполнение тестовой программы на компьютере. Такая программа недетерминирована только на некотором уровне абстракции, когда мы отвлекаемся от других программ или аппаратуры, влияющих на её поведение.

2.5. Истории

Если приоритетов нет, возможность наблюдения действия после некоторой предыстории взаимодействия не зависит от того, какая именно нажимается кнопка, разрешающая это действие. При наличии приоритетов это становится важным, поскольку различным кнопкам соответствуют различные множества разрешённых действий, и при нажатии одной кнопки предикат может оказаться истинным, и действие может наблюдаться, а при нажатии другой – ложным, и действие не может наблюдаться. Поэтому теперь нужно запоминать не только наблюдения, но также те кнопки, которые нажимал оператор. Тем самым, результатом тестового эксперимента становится последовательность действий, отказов и кнопок. Такую последовательность мы будем называть *историей*. Чтобы в истории отличить кнопку от отказа (и то и другое – подмножество внешних действий), мы будем кнопку заключать в кавычки и писать “P”, а не просто P. Если не ограничиваться только безопасным тестированием, то мы должны включить в истории также разрушение и дивергенцию, но после них история не может продолжаться, аналогично трассам. Очевидно, что в истории каждому внешнему действию z непосредственно предшествует кнопка “P”,

разрешающая это действие $z \in P$, а каждому отказу R – $\mathfrak{R}$-кнопка “R”. Могут или не могут идти две кнопки подряд, зависит от того, разрешено или запрещено переключение кнопок.

Для заданной $\mathfrak{R}/\mathfrak{Q}$-семантики истории будем называть $\mathfrak{R}/\mathfrak{Q}$-историями. Определим их более формально.

Рассмотрим LTS с предикатами $\mathbf{S}$. Для множества разрешённых действий P переход $s \text{—} z, \pi \rightarrow s'$ будем называть *P-выполнимым*, если его предикат истинен $\pi(P)=\mathit{true}$. Будем говорить, что для множества разрешённых действий P τ-маршрут *P-выполним*, если все его переходы *P-выполнимы*.

Пустая $\mathfrak{R}/\mathfrak{Q}$-история заканчивается в состояниях, достижимых из начального состояния по $\varnothing$-выполнимым τ-маршрутам, то есть, когда после включения машины не нажата никакая кнопка. Пусть $\mathfrak{R}/\mathfrak{Q}$-история σ заканчивается во множестве состояний $\mathbf{S}\ \mathit{after}\ \sigma$. Рассмотрим различные продолжения этой $\mathfrak{R}/\mathfrak{Q}$-истории. Мы будем предполагать, что $\mathfrak{R}/\mathfrak{Q}$-история не заканчивается разрушением или дивергенцией, поскольку после дивергенции и разрушения нет продолжений.

Продолжение кнопкой P, где $P \in \mathfrak{R} \cup \mathfrak{Q}$. Если допускается переключение кнопок, такое продолжение всегда возможно. Если переключения кнопок нет, $\mathfrak{R}/\mathfrak{Q}$-история не должна заканчиваться кнопкой. Переключение интерпретируется как отжатие первой кнопки, а потом нажатие второй кнопки. Поэтому сначала реализация может выполнить любой $\varnothing$-выполнимый τ-маршрут, начинающийся в состоянии из $\mathbf{S}\ \mathit{after}\ \sigma$, а затем продолжить выполнение любым P-выполнимым τ-маршрутом. Множество концов таких маршрутов и будет множеством состояний $\mathbf{S}\ \mathit{after}\ \sigma \cdot \langle \text{“P”} \rangle$. Заметим, что, если история не заканчивалась на кнопку, то концы всех $\varnothing$-выполнимых τ-маршрутов уже входят в $\mathbf{S}\ \mathit{after}\ \sigma$.

Продолжение внешним действием z. Такое продолжение возможно только в том случае, когда сама $\mathfrak{R}/\mathfrak{Q}$-история имеет вид $\sigma \cdot \langle \text{“P”} \rangle$, то есть заканчивается кнопкой P, разрешающей это действие $z \in P$. Наблюдение действия z происходит, когда совершается P-выполнимый переход по z из состояния

после предшествующей $\mathfrak{R}/\mathfrak{Q}$-истории, то есть перехода $s{-\!\!-}z,\pi{\to}s'$, где $s\in(\mathbf{S}\ \textit{after}\ \sigma\cdot\langle\text{“P”}\rangle)$ и $\pi(P)=\textit{true}$. В результате такого перехода кнопка автоматически отжимается, и далее могут выполняться $\varnothing$-выполнимые τ-маршруты до тех пор, пока не возникнет разрушение, пока не будет нажата кнопка (та же самая или другая), или пока оператор не выключит машину, заканчивая сеанс тестирования. Множество концов этих τ-маршрутов и является множеством $\mathbf{S}\ \textit{after}\ \sigma\cdot\langle\text{“P”},z\rangle$.

Продолжение $\mathfrak{R}$-отказом P. Такое продолжение возможно только в том случае, когда сама $\mathfrak{R}/\mathfrak{Q}$-история имеет вид $\sigma\cdot\langle\text{“P”}\rangle$. Отказ P возникает в таком состоянии $s\in(\mathbf{S}\ \textit{after}\ \sigma\cdot\langle\text{“P”}\rangle)$, в котором выполнено условие остановки машины: для каждого перехода (по любому действию, включая τ и γ) $s{-\!\!-}z,\pi{\to}s'$ должно быть $\pi(P)=\textit{false}$. После отказа кнопка отжимается и реализация может выполнить $\varnothing$-выполнимый τ-маршрут, начинающийся в одном из состояний, где наблюдался отказ. Множество концов этих τ-маршрутов и является множеством $\mathbf{S}\ \textit{after}\ \sigma\cdot\langle\text{“P”},P\rangle$.Заметим, что состояния, где наблюдался отказ, тоже входят в это множество (для пустого τ-маршрута).

Продолжение разрушением γ. Такое продолжение возможно только в том случае, когда в некотором состоянии $s\in(\mathbf{S}\ \textit{after}\ \sigma)$ переход $s{-\!\!-}\gamma,\pi{\to}s'$ *P-выполнимым*, если $\mathfrak{R}/\mathfrak{Q}$-история заканчивается кнопкой P (наблюдения ещё не было, и продолжает действовать кнопка P), или $\varnothing$-выполним в противном случае (после наблюдения не действует никакая кнопка). Поскольку после разрушения нет продолжения, нас не интересует множество состояний после такого продолжения.

Продолжение дивергенцией Δ. Поскольку опасна не сама дивергенция, а попытка выхода из неё, нас будет интересовать только такая дивергенция, которая выполнима при нажатой кнопке P. Такая дивергенция возникает после $\mathfrak{R}/\mathfrak{Q}$-истории вида $\sigma\cdot\langle\text{“P”}\rangle$, если есть бесконечный P-выполнимый τ-маршрут, начинающийся в состоянии из $\mathbf{S}\ \textit{after}\ \sigma\cdot\langle\text{“P”}\rangle$ (очевидно, достаточно считать, что маршрут начинается в состоянии из $\mathbf{S}\ \textit{after}\ \sigma$). В этом случае символ Δ будет продолжать $\mathfrak{R}/\mathfrak{Q}$-историю после кнопки P. Поскольку после

дивергенции нет продолжения, нас не интересует множество состояний после такого продолжения.

Теперь аналогично трассам определим *полные истории* или ***F*-истории** как $\mathfrak{R}/\mathfrak{Q}$-истории для $\mathfrak{R}=\mathcal{P}(\mathtt{L})$ и, соответственно, $\mathfrak{Q}=\varnothing$, когда любое подмножество внешних действий является $\mathfrak{R}$-кнопкой. Множество ***F*-историй** LTS $\mathtt{S}$ – обозначим так же, как множество ***F*-трасс**, – $\boldsymbol{F}(\mathtt{S})$, поскольку в дальнейшем мы будем рассматривать только истории, а не трассы. Теперь $\mathfrak{R}/\mathfrak{Q}$-история LTS – это такая её ***F*-история**, в которой встречаются кнопки только из семейств $\mathfrak{R}$ и $\mathfrak{Q}$, а отказы – это только $\mathfrak{R}$-отказы.

2.6. Безопасность и конформность без переключения кнопок

Поскольку выполнимость переходов LTS-модели с приоритетами зависит от предикатов на этих переходах, меняются отношения безопасности кнопок в реализации (***safe in***) и спецификации (***safe by***).

Если нет переключения кнопок, то отношения ***safe in*** и ***safe by*** определяются почти так же, как для машины без приоритетов, за тем исключением, что вместо $\mathfrak{R}$-трасс рассматриваются $\mathfrak{R}/\mathfrak{Q}$-истории, безопасность или опасность кнопки определяется только после $\mathfrak{R}/\mathfrak{Q}$-истории, не заканчивающейся кнопкой, продолжение внешним действием зависит от кнопки, дивергенция возможна лишь после кнопки, разрушения не должно быть не только после действия, но также после отказа (для $\mathfrak{R}$-кнопки) и сразу после нажатия кнопки.

Определение отношения безопасности в реализации без переключения кнопок:

$\mathtt{P}$ ***safe*$_\gamma$*in*** $\mathtt{I}$ ***after*** $\sigma =_{\text{def}} \sigma\cdot\langle\text{“P”},\gamma\rangle\notin\boldsymbol{F}(\mathtt{I})$ & $\forall \mathtt{u}\in\mathtt{P}\ \sigma\cdot\langle\text{“P”},\mathtt{u},\gamma\rangle\notin\boldsymbol{F}(\mathtt{I})$ & $(\mathtt{P}\in\mathfrak{R} \Rightarrow \sigma\cdot\langle\text{“P”},\mathtt{P},\gamma\rangle\notin\boldsymbol{F}(\mathtt{I}))$.

$\mathtt{P}$ ***safe*$_{\gamma\Delta}$*in*** $\mathtt{I}$ ***after*** $\sigma =_{\text{def}}$ $\mathtt{P}$ ***safe*$_\gamma$*in*** $\mathtt{I}$ ***after*** σ & $\sigma\cdot\langle\text{“P”},\Delta\rangle\notin\boldsymbol{F}(\mathtt{I})$.

$\mathtt{P}$ ***safe*$_1$*in*** $\mathtt{I}$ ***after*** $\sigma =_{\text{def}}$

$\mathtt{P}$ ***safe*$_{\gamma\Delta}$*in*** $\mathtt{I}$ ***after*** σ & $(\mathtt{P}\in\mathfrak{Q} \Rightarrow \sigma\cdot\langle\text{“P”},\mathtt{P}\rangle\notin\boldsymbol{F}(\mathtt{I}))$.

Требования к отношению безопасности в спецификации без переключения кнопок: $\forall \mathtt{R}\in\mathfrak{R}\ \forall \mathtt{z}\in\mathtt{L}\ \forall \mathtt{Q}\in\mathfrak{Q}$

1) $\mathtt{R}$ ***safe*$_1$*by*** $\mathtt{S}$ ***after*** $\sigma \Leftrightarrow \mathtt{R}$ ***safe*$_{\gamma\Delta}$*in*** $\mathtt{S}$ ***after*** σ,

2) $\exists \mathtt{T}\in\mathfrak{Q}\ \mathtt{T}$ ***safe*$_{\gamma\Delta}$*in*** $\mathtt{S}$ ***after*** σ & $\sigma\cdot\langle\text{“T”},\mathtt{z}\rangle\in\boldsymbol{F}(\mathtt{S})$

$\Rightarrow \exists P \in \mathfrak{R} \cup \mathfrak{Q}$ P ***safe₁by*** $\mathbf{S}$ ***after*** σ & $\sigma \cdot \langle \text{“P”}, z \rangle \in \boldsymbol{F}(\mathbf{S})$,

3) Q ***safe₁by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow Q$ ***safe$_{\gamma\Delta}$in*** $\mathbf{S}$ ***after*** σ & $\exists v \in Q$ $\sigma \cdot \langle \text{“Q”}, v \rangle \in \boldsymbol{F}(\mathbf{S})$.

На основе отношений безопасности кнопок в реализации и спецификации определяются безопасные действия, безопасные $\mathfrak{R}/\mathfrak{Q}$-истории ***Safe₁In***$(\mathbf{I})$ и ***Safe₁By***$(\mathbf{S})$, гипотеза о безопасности и безопасная конформность аналогично тому, как это делалось для трасс в случае машины без приоритетов. Отличия сводятся к следующему:

1) В определении безопасности $\mathfrak{R}/\mathfrak{Q}$-истории и в гипотезе о безопасности следует говорить не о трассе $\langle\gamma\rangle$, а об истории $\langle\gamma\rangle$, то есть о разрушении, выполнимом без нажатия кнопок ($\varnothing$-выполнимом).

$\mathbf{I}$ ***safe for*** $\mathbf{S}$ $=_{\text{def}}$ $(\langle \text{“}\varnothing\text{”}, \gamma \rangle \notin \boldsymbol{F}(\mathbf{S}) \Rightarrow \langle \text{“}\varnothing\text{”}, \gamma \rangle \notin \boldsymbol{F}(\mathbf{I}))$
& $\forall \sigma \in \boldsymbol{Safe_1By}(\mathbf{S}) \cap \boldsymbol{Safe_1In}(\mathbf{I})$ $\forall P \in \mathfrak{R} \cup \mathfrak{Q}$
$(P$ ***safe₁by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe₁in*** $\mathbf{I}$ ***after*** $\sigma)$.

2) В определении множества наблюдений, которые можно получить над моделью $\mathbf{T}$ при нажатии кнопки P после трассы σ, вместо продолжения трассы наблюдением нужно говорить о продолжении $\mathfrak{R}/\mathfrak{Q}$-истории кнопкой и наблюдением:

obs$(\sigma, P, \mathbf{T})$ $=_{\text{def}}$ $\{u \mid \sigma \cdot \langle \text{“P”}, u \rangle \in \boldsymbol{F}(\mathbf{T})$ & $(u \in P \vee u = P$ & $P \in \mathfrak{R})\}$.

$\mathbf{I}$ ***saco*** $\mathbf{S}$ $=_{\text{def}}$ $\mathbf{I}$ ***safe for*** $\mathbf{S}$
& $\forall \sigma \in \boldsymbol{Safe_1By}(\mathbf{S}) \cap \boldsymbol{Safe_1In}(\mathbf{I})$ $\forall P$ ***safe₁by*** $\mathbf{S}$ ***after*** σ
obs$(\sigma, P, \mathbf{I}) \subseteq$ ***obs***$(\sigma, P, \mathbf{S})$.

2.7. Безопасность и конформность с переключением кнопок

Если допускается переключение кнопок, мы можем обходить запрет на возникновение ненаблюдаемого отказа после нажатия $\mathfrak{Q}$-кнопки Q, а также дивергенцию, просто переключая её на другую кнопку P. Соответственно, модифицируются отношения безопасности: удаляются условия в определениях ***safe₁in*** и ***safe₁by***, подчёркнутые волнистой линией, и остаются условия, связанные только с разрушением.

Определение отношения безопасности в реализации с переключением кнопок:
P ***safe₂in*** $\mathbf{I}$ ***after*** σ $=_{\text{def}}$ P ***safe$_\gamma$in*** $\mathbf{I}$ ***after*** σ.

Требования к отношению безопасности в спецификации с переключением кнопок: $\forall R\in\mathfrak{R}\ \forall z\in L\ \forall Q\in\mathfrak{Q}$

1) R ***safe$_2$by*** **S** ***after*** $\sigma \Leftrightarrow R$ ***safe$_\gamma$in*** **S** ***after*** σ,

2) $\exists T\in\mathfrak{Q}\ T$ ***safe$_\gamma$in*** **S** ***after*** $\sigma\ \&\ \sigma\cdot\langle\text{“T”},z\rangle\in F(\mathbf{S})$

$\Rightarrow \exists P\in\mathfrak{R}\cup\mathfrak{Q}\ P$ ***safe$_2$by*** **S** ***after*** $\sigma\ \&\ \sigma\cdot\langle\text{“P”},z\rangle\in F(\mathbf{S})$,

3) Q ***safe$_2$by*** **S** ***after*** $\sigma \Rightarrow Q$ ***safe$_\gamma$in*** **S** ***after*** σ.

Однако возникает вопрос: сколько раз оператор может переключать кнопки? Здесь нужно учитывать, что целью нажатия кнопок является получение, в конечном счёте, некоторого наблюдения. Поскольку мы рассматриваем только конечные (по времени выполнения) тесты, цепочка переключений кнопок должна быть конечной, то есть заканчиваться нажатием кнопки, после которой оператор ожидает гарантированного наблюдения, что даёт ему возможность, в частности, закончить сеанс тестирования. Это означает, что все кнопки в цепочке, кроме последней, безопасны после непосредственно предшествующего им префикса истории по отношению ***safe$_2$in/by***, а последняя кнопка – по отношению ***safe$_1$in/by***:

P ***safe in*** **I** ***after*** $\sigma =_{\text{def}} \exists P_0{=}P, P_1, \ldots, P_n\in\mathfrak{R}\cup\mathfrak{Q}$

$\&\ \forall i{=}0..n{-}1\ P_i$ ***safe$_2$in*** **I** ***after*** $\sigma\cdot\langle\text{“}P_0\text{”},\text{“}P_1\text{”},\ldots,\text{“}P_{i-1}\text{”}\rangle$

$\&\ P_n$ ***safe$_1$in*** **I** ***after*** $\sigma\cdot\langle\text{“}P_0\text{”},\text{“}P_1\text{”},\ldots,\text{“}P_{n-1}\text{”}\rangle$,

P ***safe by*** **S** ***after*** $\sigma =_{\text{def}} \exists P_0{=}P, P_1, \ldots, P_n\in\mathfrak{R}\cup\mathfrak{Q}$

$\&\ \forall i{=}0..n{-}1\ P_i$ ***safe$_2$by*** **S** ***after*** $\sigma\cdot\langle\text{“}P_0\text{”},\text{“}P_1\text{”},\ldots,\text{“}P_{i-1}\text{”}\rangle$

$\&\ P_n$ ***safe$_1$by*** **S** ***after*** $\sigma\cdot\langle\text{“}P_0\text{”},\text{“}P_1\text{”},\ldots,\text{“}P_{n-1}\text{”}\rangle$.

Таким образом, отношение безопасности с индексом “*2*” определяет продолжение истории кнопкой, не вызывающей разрушение, а отношение безопасности с индексом “*1*” дополнительно запрещает ненаблюдаемый отказ и дивергенцию. Понятно, что 2-безопасная кнопка также и 1-безопасна, но обратное, вообще говоря, не верно. Для полной безопасности кнопки после истории требуется, чтобы она была 1-безопасна, и после неё можно было разместить конечную цепочку 1-безопасных кнопок, а затем 2-безопасную кнопку, гарантирующую наблюдение.

На основе отношений безопасности кнопок в реализации и спецификации определяются безопасные действия, безопасные истории, гипотеза о

безопасности и безопасная конформность аналогично тому, как это делалось для случая без переключения кнопок. Отличия сводятся к следующему:

1) В гипотезе о безопасности из i-безопасности кнопки в спецификации должна следовать тоже i-безопасность кнопки в реализации, где i=1,2:

$$\mathbf{I}\ \textit{safe for}\ \mathbf{S} =_{\text{def}} \quad (\langle\text{“}\varnothing\text{”},\gamma\rangle \notin \boldsymbol{F}(\mathbf{S}) \ \Rightarrow\ \langle\text{“}\varnothing\text{”},\gamma\rangle \notin \boldsymbol{F}(\mathbf{I}))$$
$$\&\ \forall\sigma\in \boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{SafeIn}(\mathbf{I})\ \ \forall \mathrm{P}\in\mathfrak{R}\cup\mathfrak{Q}$$
$$(\mathrm{P}\ \textit{safe}_i\ \textit{by}\ \mathbf{S}\ \textit{after}\ \sigma\ \Rightarrow\ \mathrm{P}\ \textit{safe}_i\ \textit{in}\ \mathbf{I}\ \textit{after}\ \sigma).$$

2) В определении конформности вложенность множеств наблюдений должна требоваться только после 1-безопасных историй, то есть историй, заканчивающихся на 1-безопасную кнопку:

$$\mathbf{I}\ \textit{saco}\ \mathbf{S} =_{\text{def}} \quad \mathbf{I}\ \textit{safe for}\ \mathbf{S}$$
$$\&\ \forall\sigma\in \boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{SafeIn}(\mathbf{I})\ \ \forall \mathrm{P}\ \textit{safe}_1\textit{by}\ \mathbf{S}\ \textit{after}\ \sigma$$
$$\boldsymbol{obs}(\sigma,\mathrm{P},\mathbf{I})\subseteq\boldsymbol{obs}(\sigma,\mathrm{P},\mathbf{S}).$$

2.8. Параллельная композиция и генерация тестов

Рассмотрим композицию двух LTS с приоритетами $\mathbf{I}$ и $\mathbf{T}$ в алфавитах, соответственно, A и B. Возьмём любое композиционное состояние it. При композиции множество разрешённых внешних действий для LTS $\mathbf{I}$ в состоянии i – это множество противоположных внешних действий, по которым есть переходы из состояния t другой LTS $\mathbf{T}$, и наоборот. Поэтому, прежде всего, нам нужно пересчитать предикаты переходов из этих состояний. В силу коммутативности оператора композиции (с точностью до изоморфизма, то есть именования состояний it или ti), нам достаточно рассмотреть только пересчёт предикатов одной LTS, для определённости, LTS $\mathbf{I}$. Для перехода $\mathrm{i}\text{—}\mathrm{z},\pi_{\mathrm{i}}\rightarrow\mathrm{i}`$ мы должны в предикат π_{i}, понимаемый как булевская функция от булевских переменных-действий, подставить константное значение каждой переменной, соответствующей синхронному действию $\mathrm{z}\in\mathrm{A}\cap\underline{\mathrm{B}}$. Если есть переход $\mathrm{t}\text{—}\underline{\mathrm{z}},\pi_{\mathrm{t}}\rightarrow\mathrm{t}`$, то подставляется значение ***true***, иначе – ***false***. Получается новый предикат π_{it}. Заметим, что вычисление нового предиката на переходе из состояния i зависит от состояния t, с которым оно компонуется, то есть для разных состояний t будут разные предикаты π_{it}.

Новый предикат π_{it} может быть не константным, поскольку в нём могут остаться переменные, соответствующие асинхронным внешним действиям из $\mathrm{A}\setminus\underline{\mathrm{B}}$. Кроме того, теперь этот предикат следует понимать как предикат в

композиционном алфавите $A{\downarrow\!\downarrow}B=(A\setminus\underline{B})\cup(B\setminus\underline{A})$, хотя реально он не зависит от переменных, соответствующих действиям из $B\setminus\underline{A}$.

Асинхронный переход соответствует одному переходу в одном из LTS-операндов. Он может выполняться, если может выполняться наследуемый переход. Следовательно, предикат асинхронного композиционного перехода совпадает с предикатом наследуемого перехода после пересчёта, то есть не с исходным предикатом π_i, а с предикатом π_{it}. Синхронный переход – это одновременное выполнение переходов в каждом LTS-операнде. Он может выполняться, если могут выполняться оба перехода-операнда. Следовательно, предикат синхронного композиционного перехода равен конъюнкции пересчитанных переходов-операндов $\pi_{it}\&\pi_{ti}$. В целом композиционные переходы порождаются следующими правилами вывода:

(1*) $z\in\{\tau,\gamma\}\cup A\setminus\underline{B}$ & $i\text{—}z,\pi_i\to i'$ $\vdash it\text{—}z,\pi_{it}\to i't$,

(2*) $z\in\{\tau,\gamma\}\cup B\setminus\underline{A}$ & $t\text{—}z,\pi_t\to t'\vdash it\text{—}z,\pi_{ti}\to it'$,

(3*) $z\in A\cap\underline{B}$ & $i\text{—}z,\pi_i\to i'$ & $t\text{—}\underline{z},\pi_t\to t'\vdash it\text{—}\tau,\pi_{it}\&\pi_{ti}\to i't'$.

Как и в случае машины без приоритетов тестирование понимается как композиция LTS-реализации **I** в алфавите A и LTS-теста **T** в противоположном алфавите $B=\underline{A}$. Мы также будем предполагать, что в тесте нет разрушения. Переходы по внешним действиям в тесте не имеют предикатов, точнее их предикаты тождественно истинны. Поэтому в композиционной LTS все переходы (а это уже только τ- и γ-переходы) – это пересчитанные предикаты переходов реализации. Поскольку композиционный алфавит пуст, эти предикаты константны (***true*** или ***false***).

Для обнаружения отказов в тесте (но не в реализации!) также используются θ-переходы с тождественно истинными переходами. Если нет переключения кнопок, такой переход срабатывает тогда и только тогда, когда никакие другие переходы не могут выполняться:

(4*) $t\text{—}\theta\to t'$ & ***Deadlock***(i,t) $\vdash it\text{—}\tau\to it'$,

где ***Deadlock***(i,t) = $i\text{—}\tau,\pi_{it}\not\to$ & $i\text{—}\gamma,\pi_{it}\not\to$ & $t\text{—}\tau\not\to$

& $(\forall z\in A\cap\underline{B}\ i\text{—}z,\pi_{it}\not\to\ \vee\ t\text{—}\underline{z}\not\to)$.

Если допускается переключение кнопок, то в тесте оно отображается в виде τ-перехода “P”—τ→“Q” из состояния, соответствующего одной кнопке P, в

состояние, соответствующее другой кнопке Q. θ-переход определяется в состоянии "P", если кнопка P – это $\mathfrak{R}$-кнопка. Нужно, чтобы θ-переход мог срабатывать независимо от τ-перехода переключения кнопок "P"—τ→"Q": удаляется условие t—τ↛, подчёркнутое волнистой линией.

Мы будем рассматривать только безопасные реализации и безопасные тесты. Под безопасным тестом здесь понимается тест, при взаимодействии которого с любой безопасной реализацией не возникают разрушение, дивергенция, выполнимая после нажатия кнопки, а также тупики. Такие тесты строятся на основе безопасных историй спецификации. Тестовая история – это либо безопасная история спецификации, либо безопасная история спецификации, заканчивающаяся кнопкой, которая продолжена наблюдением (действием или $\mathfrak{R}$-отказом), отсутствующим после этой истории в спецификации. Также мы будем рассматривать только такие тесты, которые заканчиваются через конечное время; мы будем называть их конечными тестами. Для безопасного LTS-теста это означает отсутствие в нём бесконечных маршрутов.

В композиции теста с реализацией все предикаты константны, мы можем удалить все переходы с ложными предикатами. Если реализация безопасна, а тест конечен и безопасен, то оставшиеся γ-переходы недостижимы. Как и для машины без приоритетов выполнению теста соответствует прохождение τ-маршрута, начинающегося в начальном состоянии композиции и заканчивающегося в композиционном состоянии it, где t терминальное состояние теста, которому назначен вердикт ***pass*** или ***fail***. Заметим, что в композиции могут быть бесконечные маршруты, но они не могут проходиться при тестировании. Действительно, поскольку тест конечен, в таком маршруте, начиная с некоторого места, состояние теста не меняется: дальше идут только асинхронные τ-переходы реализации. В процессе тестирования оператор машины всегда за конечное время дожидается наблюдения, нажимает или переключает кнопку, что означает смену состояния теста и, тем самым, прекращение выполнения бесконечной цепочки τ-переходов реализации. Здесь мы опираемся на то, что за конечное время может выполняться только конечная цепочка переходов. Оператор может также выключить машину (окончание тестирования) , что происходит через конечное время после наблюдения.

Тест должен взаимодействовать с реализацией, согласно той $\mathfrak{R}$/$\mathfrak{Q}$-семантики, в которой рассматривалась спецификация, по которой тест сгенерирован. Поэтому в каждом состоянии теста множество действий, по которым

определены переходы из состояния, должно соответствовать $\mathfrak{R}$- или $\mathfrak{Q}$-кнопке, причём для $\mathfrak{R}$-кнопки дополнительно должен быть определён θ-переход. Мы приняли допущение о том, что сразу после включения машины до нажатия первой кнопки, после любого наблюдения, а также при переключении кнопок в реализации могут выполняться ∅-выполнимые τ- и γ-переходы. Это допущение является частью семантики взаимодействия, которую должен соблюдать тест. Поэтому нужно ввести в тест дополнительные *пустые* состояния, соответствующие ситуации, когда нет нажатых кнопок. Эти состояния теста должны позволять реализации совершать ∅-выполнимые τ-переходы. При отжатых кнопках множество разрешённых действий пусто, поэтому в пустом состоянии могут быть определены только τ-переходы. Эти τ-переходы, в конечном счёте, должны приводить к непустым состояниям, соответствующим той или иной кнопке (пустая $\mathfrak{R}$-кнопка соответствует состоянию, в котором определён θ-переход, а пустую $\mathfrak{Q}$-кнопку мы запретили). Такой пустой кнопкой должно быть начальное состояние и постсостояние каждого перехода по наблюдению, если это не терминальное состояние.

Примитивный тест строится аналогично тому, как это делается для машины без приоритетов. Но есть три отличия. 1) Без приоритетов мы строили тест по безопасной $\mathfrak{R}$-трассе, превращая её в одну из $\mathfrak{R}/\mathfrak{Q}$-историй, а теперь сразу начинаем с некоторой безопасной $\mathfrak{R}/\mathfrak{Q}$-истории. 2) Если в истории есть переключение с кнопки P на кнопку Q, то в тесте проводится τ-переход “P”—τ→“Q”. 3) Добавляются пустые состояния. По-прежнему, набор всех примитивных тестов полон, а любой строгий тест можно заменить на объединение примитивных тестов, которое обнаруживает те же самые ошибки.

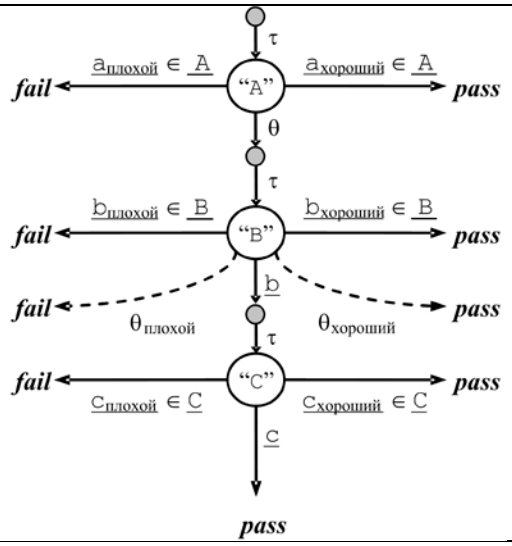
Примитивный тест для безопасной $\mathfrak{R}/\mathfrak{Q}$-истории σ	
	$\sigma=\langle$"A",A,"B",b,"C",c$\rangle$, где $A,B\in\mathfrak{R}$, $C\in\mathfrak{Q}$. Внешнее действие имеет индекс «$_{\text{хорошо}}$», если оно есть в спецификации после префикса истории; иначе – индекс «$_{\text{плохо}}$». Пунктиром показаны взаимоисключающие θ-переходы, ведущие из одного состояния. Индексы «$_{\text{хорошо}}$» и «$_{\text{плохо}}$» означают наличие или отсутствие продолжения префикса истории соответствующим $\mathfrak{R}$-отказом. Маленькими серыми кружками показаны пустые состояния.

Рис.5.

2.9. Примеры задания приоритетов

Покажем, как задаются приоритеты с помощью предикатов на переходов LTS-модели для примеров, приведённых во введении.

Выход из дивергенции. 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). Переход по стимулу имеет тождественно истинный предикат, а переход по реакции имеет предикат π, истинный на любом подмножестве действий, не содержащем стимулов: $\pi(U)=(\forall ?x\ ?x\notin U)$. Обычно также подразумевается, что внутренняя активность менее приоритетна, чем приём стимула, то есть τ-переход имеет такой же предикат, как переход по реакции.

Приоритет выдачи над приёмом в неограниченных очередях. Переход по реакции имеет тождественно истинный предикат, а переход по стимулу имеет предикат π, истинный на любом подмножестве действий, не содержащем реакцию: $\pi(U)=(\forall !y\ !y\notin U)$. Обычно также подразумевается, что внутренняя активность менее приоритетна, чем выдача реакции, то есть τ-переход имеет такой же предикат, как переход по стимулу.

Прерывание цепочки действий. Переход по команде «отменить» (cancel) имеет тождественно истинный предикат, а все остальные переходы имеют предикат π, истинный на любом подмножестве действий, не содержащем “cancel”: $\pi(U)=(\text{cancel}\notin U)$.

Приоритетная обработка входных воздействий. Множество стимулов разбивается на непересекающиеся подмножества $X_1, X_2, \ldots$ так, что стимулы из подмножества с большим индексом имеют больший приоритет. Предикат π_i на переходе по стимулу из X_i истинен на любом подмножестве действий, не содержащем стимулы из подмножества с большим номером: $\pi_i(U)=(\forall j>i\ U\cap X_j=\varnothing)$. Возможна также дифференциация переходов из некоторого состояния по одному и тому же стимулу в зависимости от наличия или отсутствия менее приоритетных стимулов. Например, один переход по стимулу из X_i выполняется, если окружение предлагает менее приоритетные стимулы $\pi_{i1}(U)=\pi_i(U)\&(\exists j<i\ U\cap X_j\neq\varnothing)$, в предположении, что это предложение сохранится, и эти стимулы можно будет обработать потом, а другой переход выполняется, если менее приоритетных стимулов нет $\pi_{i2}(U)=\pi_i(U)\&(\forall j<i\ U\cap X_j=\varnothing)$. Если в состоянии нет переходов по стимулам, то такая дифференциация возможна и между переходами по реакциям и/или τ-переходам.

Возможна реализация и более экзотических приоритетов. Например, циклический приоритет движения по сторонам света: идём на север, если нельзя идти на восток; идём на восток, если нельзя идти на юг; идём на юг, если нельзя идти на запад; идём на запад, если нельзя идти на север. Если разрешены все четыре направления, выбирается любое. Кроме этого случая, равноприоритетными оказываются только противоположные направления при отсутствии остальных. Предикат перехода на север выглядит так $\pi_{\text{север}}(U)=(\text{восток}\notin U\ \vee\ U=\{\text{север},\text{восток},\text{юг},\text{запад}\})$. Аналогично устроены предикаты переходов на восток, юг и запад.

Заключение

Можно рассматривать семантики, в которых при включении машины, после наблюдения и при переключении кнопок может не допускаться выполнение реализацией τ- и γ-переходов, даже если они $\varnothing$-выполнимы. Можно считать, что сразу после включения машины и сразу после наблюдения реализация стоит, и может выполнять какие-то действия только после нажатия кнопки. Также переключение кнопок не интерпретируется как отжатие первой кнопки (с разрешением $\varnothing$-выполнимых τ- и γ-действий), а потом нажатие второй

кнопки. Иными словами, после включения машины, после наблюдения и между двумя кнопками при переключении кнопок нет никакого «пустого» промежутка. Такая семантика, очевидно, предполагает более сильные тестовые возможности, чем слабая семантика, рассматриваемая в данной статье. Эти семантики имеют разные требования по безопасности и конформности.

Любое поведение, которое можно наблюдать при сильной семантике можно наблюдать и при слабой семантике: достаточно подобрать подходящие погодные условия, когда оператор успевает нажать или переключить кнопку достаточно быстро. Верно и обратное: поведение при слабой семантике наблюдается при сильной семантике, если добавить пустую кнопку и явно нажимать её. Однако условия безопасности для этих семантик разные. При слабой семантике мы всегда должны рассчитывать на возможность выполнения τ- и γ-действий (при наличии приоритетов, они должны быть $\varnothing$-выполнимы) после наблюдения по кнопке P, а такие действия могут давать дивергенцию или разрушение; тем самым, кнопка P будет опасной. При сильной семантике мы можем просто не нажимать в этой ситуации пустую кнопку после такого наблюдения, поскольку она опасна, а кнопка P будет безопасной. Отсюда же вытекают и соответствующие различия в конформности: реализация может быть опасной при слабой семантике и, следовательно, не конформной, но безопасной и конформной при сильной семантике. При тех же условиях безопасности (например, когда в спецификации нет дивергенции, разрушения и ненаблюдаемых отказов) и при наличии приоритетов сильная семантика предъявляет более жёсткие условия конформности. Это объясняется тем, что мы получаем возможность различать реализации, в которых некое действие b, разрешаемой кнопкой B, выполняется сразу после действия a или через промежуточную $\varnothing$-выполнимую, но не B-выполнимую τ-активность.

Кроме генерации тестов, важнейшей проблемой теории конформности является проблема монотонности – сохранения конформности при композиции. В общем случае композиция реализаций, конформных своим спецификациям, может быть не конформна композиции этих спецификаций. Частным, но важным, случаем этой проблемы является проблема асинхронного тестирования, когда имеется два компонента: реализация и известная среда передачи. Здесь также композиция конформной реализации со средой может быть не конформна композиции спецификации с этой средой.

Для машин без приоритетов эта проблема решается с помощью, так называемого, монотонного преобразования спецификаций: композиция конформных реализаций оказывается конформной композиции преобразованных спецификаций. Или, для асинхронного тестирования:

композиция конформной реализации со средой конформна композиции преобразованной спецификации с этой средой. Монотонное преобразование выполняется для $\mathfrak{R}/\mathfrak{Q}$-семантик, в которых все отказы наблюдаемы, то есть $\mathfrak{Q}=\varnothing$. В общем случае $\mathfrak{R}/\mathfrak{Q}$-семантики сначала выполняется, так называемое, пополнение спецификации. Пополненная спецификация эквивалентна (имеет тот же класс безопасных и тот же класс конформных реализаций) исходной спецификации в $\mathfrak{R}/\mathfrak{Q}$-семантике, а кроме того, эквивалентна сама себе в $\mathfrak{R}\cup\mathfrak{Q}/\varnothing$-семантике. Пополнение решает также проблему рефлексивности («самоприменимости») спецификации, которая в $\mathfrak{R}/\mathfrak{Q}$-семантике может быть не конформна сама себе. Тем самым, совокупность преобразования пополнения и монотонного преобразования решает общую проблему монотонности и рефлексивности для любой $\mathfrak{R}/\mathfrak{Q}$-семантики [13,15,16].

Для машин с приоритетами проблемы монотонности и рефлексивности ещё не решены.

Литература

1. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. «Программирование». 2000. No. 2.
2. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Асинхронные автоматы: классификация и тестирование. Труды ИСП РАН, т. 4, 2003.
3. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай. «Программирование». 2003. No. 5.
4. Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов. «Программирование», 2003, No. 6.
5. Kuliamin V.V., Kossatchev A.S., Petrenko A.K., Pakoulin N.V., Bourdonov I.B. Integration of Functional and Timed Testing of Real-Time and Concurrent Systems. Perspectives of System Informatics // LNCS. No. 2890, Springer-Verlag, 2003.
6. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. «Программирование». 2004. No. 1.
7. Бурдонов И.Б. Обход неизвестного ориентированного графа конечным роботом. «Программирование», 2004, No. 4.

8. Бурдонов И.Б. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. «Программирование», 2004, No. 6.
9. Бурдонов И.Б. Исследование одно/двунаправленных распределённых сетей конечным роботом. Труды Всероссийской научной конференции "Научный сервис в сети ИНТЕРНЕТ". 2004.
10. Бурдонов И.Б., Косачев А.С. Тестирование компонентов распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
11. Бурдонов И.Б., Косачев А.С. Верификация композиции распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
12. Bourdonov I., Kossatchev A., Kuliamin V. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. Proc. Of MBT 2006, Vienna, Austria, March 2006.
13. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, No. 5.
14. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Безопасность, верификация и теория конформности. Материалы Второй международной научной конференции по проблемам безопасности и противодействия терроризму, Москва, МНЦМО, 2007.
15. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.
16. Бурдонов И.Б. Теория конформности для функционального тестирования программных систем на основе формальных моделей. Диссертация на соискание учёной степени д.ф.-м.н., Москва, 2007. http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
17. Василевский М.П. О распознавании неисправностей автомата. Кибернетика, т. 9, № 4, стр. 93–108, 1973.
18. Aho A.V., Dahbura A.T., Lee D., Uyar M.Ü. An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours. IEEE Transactions on Communications, 39(11):1604–1615, 1991.
19. Blass A., Gurevich Y., Nachmanson L., Veanes M. Play to Test Microsoft Research. Technical Report MSR-TR-2005-04, January 2005. 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005). Edinburgh, July 2005.

20. Edmonds J. Johnson E.L. Matching. Euler Tours, and the Chinese Postman. Math. Programming 5, 88-124 (1973).

21. Fujiwara S. Bochmann G.v. Testing Nondeterministic Finite State Machine with Fault Coverage. IFIP Transactions, Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991, Ed. By Jan Kroon, Rudolf J. Hei-jink, and Ed Brinksma, 1992, North-Holland, pp. 267-280.

22. 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.

23. 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.

24. Goodenough J.B. Gerhart S.L. Toward a theory of test data selection. IEEE Trans. Software Eng., vol. SE-1, no. 2, pp. 156- 173, June 1975.

25. Grochtmann M., Grimm K. Classification trees for partition testing. Software Testing, Verification and Reliability, 3:63-82, 1993.

26. Heerink L., Tretmans J. Refusal Testing for Classes of Transition Systems with Inputs and Outputs. In T.Mizuno, N.Shiratori, T.Higashino, A.Togashi, eds. Formal Description Techniques and Protocol Specification, Testing and Verification. Chapman & Hill, 1997.

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

28. Lee D., Yannakakis M. Testing Finite State Machines: State Identification and Verification. IEEE Trans. on Computers, Vol. 43, No. 3, March 1994, pp. 306-320.

29. Lee D., Yannakakis M. Principles and Methods of Testing Finite State Machines – A Survey. Proceedings of the IEEE 84, No. 8, 1090–1123, 1996.

30. Legeard B., Peureux F., Utting M. Automated boundary testing from Z and B. In Proc. of the Int. Conf. on Formal Methods Europe, FME'02, volume 2391 of LNCS, Copenhaguen, Denmark, pages 21--40, July 2002. Springer.

31. Lestiennes G., Gaudel M.-C. Test de systemes reactifs non receptifs. Journal Europeen des Systemes Automatises, Modelisation des Systemes Reactifs, pp. 255–270. Hermes, 2005.

32. Milner R. A Calculus of Communicating Processes. LNCS, vol. 92, Springer-Verlag, 1980.

33. Milner R. Modal characterization of observable machine behaviour. In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer, pp. 25-34.

34. Milner R. Communication and Concurrency. Prentice-Hall, 1989.

35. Petrenko A., Yevtushenko N., Bochmann G.v. Testing deterministic implementations from nondeterministic FSM specifications. Selected proceedings of the IFIP TC6 9-th international workshop on Testing of communicating systems, September 1996.

36. Petrenko A., Yevtushenko N., Huo J.L. Testing Transition Systems with Input and Output Testers. Proc. IFIP TC6/WG6.1 15th Int. Conf. Testing of Communicating Systems, TestCom'2003, pp. 129-145. Sophia Antipolis, France, May 26-29, 2003.

37. Zhu, Hall, May. Software unit test coverage and adequacy. ACM Computing Surveys, v.29, n.4, 1997.