И.Б.Бурдонов, А.С.Косачев.
Обобщённые семантики тестового взаимодействия.
Труды Института системного программирования РАН, N 15, 2008, стр.69-106.
38 стр.

Бурдонов И.Б., Косачев А.С.

Обобщённые семантики тестового взаимодействия

Введение

В нашей работе [4] введён класс семантик тестового взаимодействия, основанного на наблюдениях двух типов: наблюдение внешнего действия, выполняемого тестируемой системой, и наблюдение отсутствия действий из некоторого множества действий, называемое отказом. При этом тестовое воздействие сводится к разрешению системе выполнять любое действие из заданного множества действий. Предполагалось, что отказ либо не наблюдается при данном тестовом воздействии ($\mathfrak{Q}$-отказ), либо совпадает с множеством разрешаемых действий ($\mathfrak{R}$-отказ). Такая семантика называлась $\mathfrak{R}/\mathfrak{Q}$-семантикой и задавалась двумя семействами подмножеств алфавита внешних действий: $\mathfrak{R}$ и $\mathfrak{Q}$. Подробное изложение теории конформности с доказательствами утверждений содержится в докторской диссертации И.Бурдонова [7], теория конформности для класса, так называемых $\beta\gamma\delta$-семантик излагается в нашей книге [6].

В этих работах предполагалось отсутствие приоритетов между действиями, которые тестируемая система может выполнять при данном тестовом воздействии: действие может выполняться независимо от того, какие ещё действия разрешаются тестовым воздействием. В то же время для реальных программных и аппаратных систем такая модель не всегда адекватно отражает требуемое поведение системы. В нашей статье [8] предлагается способ введения приоритетов в теорию конформности для $\mathfrak{R}/\mathfrak{Q}$-семантики: в модель системы, отношение конформности, методы генерации тестов и оператор композиции (сборки составной системы из взаимодействующих между собой компонентов).

В данной работе мы обобщаем семантику взаимодействия, допуская наблюдение отказов, не обязательно совпадающих с множеством разрешаемых действий (и даже не обязательно вложенных в него). Рассмотрение состоит из двух частей: сначала рассматриваются модели без приоритетов, а потом вводятся приоритеты.

1. 𝔓-семантика без приоритетов

1.1. Формализация взаимодействия

Верификация конформности понимается как проверка соответствия исследуемой системы заданным требованиям (Рис.1). 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. Для такой проверки предполагается, что реализационная модель существует (так называемая, тестовая гипотеза), хотя может быть неизвестной. В модельном мире по спецификации генерируются модельные тесты и определяется отношение «реализация *проходит* тест». Набор тестов *полон*, если реализация проходит каждый тест из набора тогда и только тогда, когда она конформна спецификации. Модельные тесты транслируются в реальные тестовые программы, которые прогоняются на тестируемой системе. Реальное отношение «*проходит*» должно адекватно отражаться в модельном отношении «*проходит*». Само отношение конформности и его тестирование (в частности, отношение «*проходит*») базируются на той или иной модели взаимодействия.

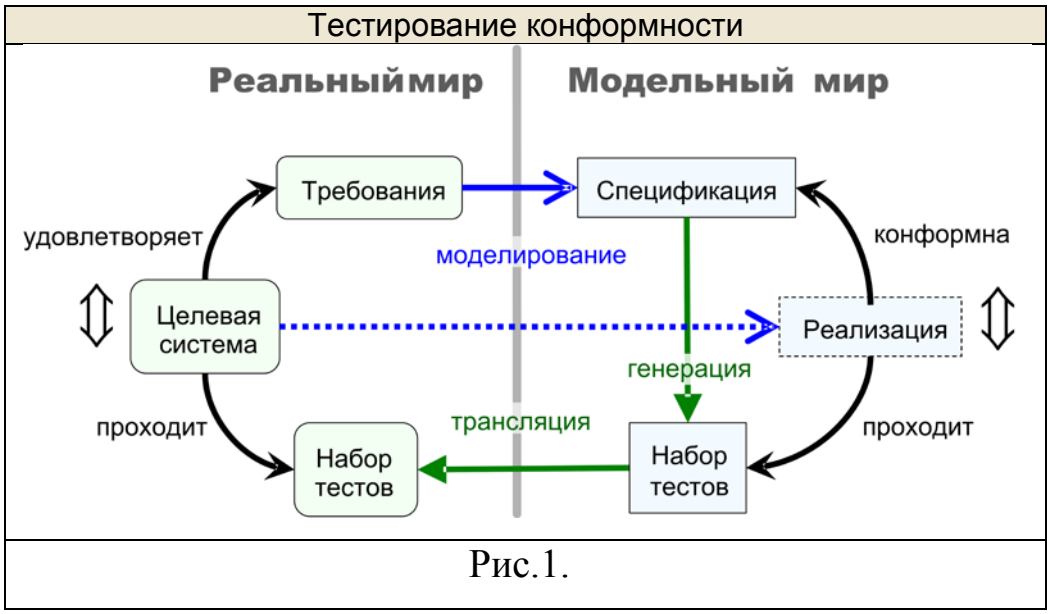


Рис.1.

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 [4,6,7,9,10,15]. Она представляет собой «чёрный ящик», внутри которого находится реализация (Рис.2). 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*, выполняемого реализацией, и наблюдение *отказа* как отсутствия каких бы то ни было действий из некоторого множества действий.

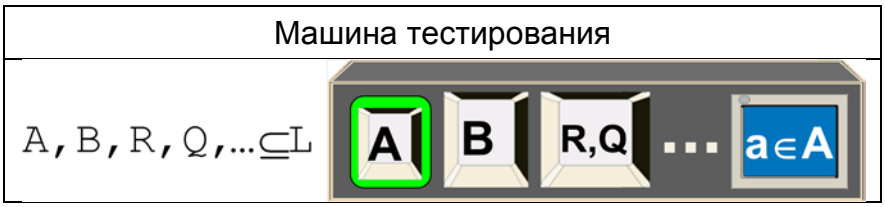


Рис.2.

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 A, но на кнопке «написано», вообще говоря, не одно действие, а множество разрешаемых действий A. 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 множество разрешаемых действий – это, вообще говоря, не любое подмножество множества всех внешних 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).

Также следует подчеркнуть, что наблюдаться может, вообще говоря, не любой отказ. 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 [1-6,11,12,13]. 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. Это даёт наблюдение отказа, называемого «частичной стационарностью» – отсутствие реакций из множества не всех реакций, а только тех, что относятся к данному выходному каналу [11,12].

Для наблюдения отказа нужно, чтобы оператор каким-то образом указал соответствующее множество внешних действий. В [4,6,7,8] предполагалось, что это множество совпадает с множеством разрешаемых действий: если оператор нажимает кнопку A, разрешая реализации выполнять действия из множества A, то наблюдаться может только отказ A, который означает, что реализация не может выполнить ни одно действие из A. В то же время существуют случаи, когда такого совпадения нет. Например, в реактивных системах без «торможения» реакций, когда стационарность наблюдаема, а блокировки стимулов не наблюдаемы, при посылке в реализацию стимула с одновременным приёмом всех реакций отказ может означать только стационарность – отсутствие реакций, ничего не говоря о том, могла бы реализация принять стимул или нет. Здесь множество разрешаемых действий – это «стимул + все реакции», а наблюдаемый отказ – «все реакции». Более того, «в ответ» на одно и то же тестовое воздействие (нажатие кнопки машины тестирования) может наблюдаться не обязательно какой-то один отказ. С тестовым воздействием может быть связано множество отказов, один из которых и будет наблюдаться. Например, в реактивных системах с несколькими выходными каналами посылка стимула с одновременным приёмом реакций по нескольким каналам может вызвать наблюдение частичной стационарности по одному из этих каналов.

В данной статье мы расширяем класс рассматриваемых семантик, предполагая, что оператор указывает не только множество разрешаемых действий, но и множество ожидаемых отказов, которые могут наблюдаться. Это означает, что на кнопке написано множество наблюдений, возможных после нажатия этой

кнопки: разрешаемые внешние действия и ожидаемые наблюдаемые отказы как множества действий.

Итак, семантика взаимодействия определяется алфавитом внешних действий L и набором кнопкок – семейством $\mathfrak{P}\subseteq\mathcal{P}(L\cup\mathcal{P}(L))$. Там, где это нужно для однозначного понимания, кнопку $P\in\mathfrak{P}$ мы будем называть $\mathfrak{P}$-кнопкой. Для кнопки $P\in\mathfrak{P}$ множество разрешаемых действий обозначим $P_q=P\cap L$, а множество ожидаемых наблюдаемых отказов $P_r=P\cap\mathcal{P}(L)$. Операции "$_r$" и "$_q$" распространим также на семейства кнопок: $\mathfrak{U}_q=\{P_q\,|\,P\in\mathfrak{U}\}$ и $\mathfrak{U}_r=\{P_r\,|\,P\in\mathfrak{U}\}$. Предполагается, что любое внешнее действие из алфавита разрешается некоторой кнопкой $\cup(\mathfrak{P}_q)=L$. Отказ, являющийся элементом семейства $\mathfrak{P}_r$, то есть наблюдаемый (в принципе) отказ, будем называть $\mathfrak{R}$-отказом. Такую семантику мы называем $\mathfrak{P}$-семантикой. Она отличается от $\mathfrak{R}/\mathfrak{Q}$-семантики в [4,6,7,8] тем, что для $\mathfrak{P}$-кнопки P не обязательно $P_r=\{P_q\}$, как для $\mathfrak{R}$-кнопки в $\mathfrak{R}/\mathfrak{Q}$-семантике, то есть $\mathfrak{R}$-отказов, ожидаемых при нажатии $\mathfrak{P}$-кнопки, может быть несколько и они не обязательно совпадают с множеством разрешённых действий. Если $P_r=\varnothing$, то такая $\mathfrak{P}$-кнопка соответствует $\mathfrak{Q}$-кнопке в $\mathfrak{R}/\mathfrak{Q}$-семантике, и такую $\mathfrak{P}$-кнопку мы также будем называть $\mathfrak{Q}$-кнопкой в $\mathfrak{P}$-семантике. Соответственно, $\mathfrak{P}$-кнопку P, содержащую отказы $P_r\neq\varnothing$, будем называть также $\mathfrak{R}$-кнопкой в $\mathfrak{P}$-семантике.

1.2. Безопасное тестирование

При тестировании возможно возникновение тупика, когда никакого наблюдения нет и неизвестно, будет оно через какое-то время или не будет никогда.

Это возможно при нажатии кнопки P, если реализация не может выполнить ни одно разрешённое действие из P_q, но отказов из P_r также нет – для каждого отказа $R\in P_r\setminus P_q$ реализация могла бы выполнить некоторое неразрешённое действие $z\in R\setminus P_q$. В $\mathfrak{R}/\mathfrak{Q}$-семантике это было возможно только при нажатии $\mathfrak{Q}$-кнопки, но в $\mathfrak{P}$-семантике это возможно также при нажатии $\mathfrak{R}$-кнопки P, которая содержит отказ, не вложенный во множество разрешённых действий, то есть, когда $\cup P_r\not\subseteq P_q$. Понятно, что, если мы хотим, чтобы тест заканчивался

через конечное время с вынесением соответствующего вердикта (***pass*** или ***fail***), то мы должны избегать при тестировании возникновения тупика.

Кроме внешних, наблюдаемых действий реализация может совершать внутренние, ненаблюдаемые (и, следовательно, неразличимые оператором) действия, которые обозначаются символом τ. Выполнение τ-действий не регулируется оператором – они всегда разрешены. Отказ может наблюдаться только тогда, когда реализация не может выполнять не только внешние действия из этого отказа, но и τ-действия. Предполагается, что любая конечная последовательность любых действий совершается за конечное время, а бесконечная последовательность – за бесконечное время. Бесконечная последовательность τ-действий называется *дивергенцией* («зацикливание») и обозначается символом Δ. Само по себе возникновение дивергенции не опасно, однако при наличии дивергенции любое продолжение тестирования, то есть нажатие любой кнопки, не гарантирует наблюдение через конечное время. Это объясняется тем, что оператор, ничего не наблюдая, не знает, случится ли такое наблюдение в будущем, или реализация так и будет бесконечно долго выполнять свои внутренние действия. Для конечных (по времени выполнения) тестов это плохо.

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. Если в реализации после выполнения некоторого внешнего действия возможно разрушение, нажатие кнопки, разрешающей это действие, может привести к разрушению. Опасность возникновения разрушения при тестировании подразумевается его семантикой.

Итак, поскольку мы ограничиваемся конечными по времени выполнения тестами и не хотим разрушать реализацию, мы должны избегать при тестировании возникновения тупиков, попыток выхода из дивергенции и разрушения. Такое тестирование называется безопасным.

Можно также отметить, что нажатие кнопки P с пустым множеством P_q разрешаемых действий не эквивалентно отсутствию нажатой кнопки. В обоих случаях все внешние действия запрещены, однако при нажатии $\mathfrak{R}$-кнопки P,

то есть $P_r \neq \varnothing$, возможно наблюдение отказа $R \in P_r$: оператор узнаёт об остановке машины, когда она не может выполнять внутренние действия, разрушение и действия из отказа R, даже если бы они были разрешены. Кнопка P с пустым множеством разрешённых действий P_q не может вызвать разрушение после действия (никакого действия быть не может), но она опасна, если есть дивергенция, как и любая другая кнопка. Кнопка $\{\varnothing\}$ запрещает все внешние действия и разрешает наблюдение только пустого отказа, означающего остановку машины, когда она не может выполнять внутренние действия и разрушение. Пустую кнопку $\varnothing$ вообще никогда нельзя нажимать, поскольку никакого наблюдения быть не может; такая кнопка соответствует отсутствию нажатой кнопки. Поэтому будем считать, что $\varnothing \notin \mathfrak{P}$.

1.3. 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 $\mathbf{S}=\mathrm{LTS}(V_S, L, E_S, s_0)$, где V_S – непустое множество состояний (вершин графа), L – алфавит внешних действий, $E_S \subseteq V_S \times (L \cup \{\tau,\gamma\}) \times V_S$ – 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 $s \text{—}z\!\!\rightarrow s'$. Обозначим $s \text{—}z\!\!\rightarrow =_{\text{def}} \exists s'\ s \text{—}z\!\!\rightarrow s'$ и $s \text{—}z\!\!\not\rightarrow =_{\text{def}} \nexists s'\ s \text{—}z\!\!\rightarrow s'$.

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 s LTS-модели $\mathbf{S}$ называется *стабильным*, если в нём не определены τ- и γ-переходы: $\boldsymbol{stab}(s,\mathbf{S}) =_{\text{def}} s \text{—}\tau\!\!\not\rightarrow \ \&\ s \text{—}\gamma\!\!\not\rightarrow$.

Состояние называется *дивергентным*, если в нём начинается бесконечная цепочка τ-переходов (в частности, τ-цикл, в том числе, τ-петля). Отказ порождается стабильным состоянием, в котором нет переходов по действиям из этого отказа. Если в данном состоянии при данной нажатой кнопке (или отсутствии нажатых кнопок) возможно выполнение нескольких действий, то выбирается одно из них недетерминированным образом. Также, если в стабильном состоянии при нажатой кнопке P возможно выполнение внешних

действий из P_q и имеется один или несколько отказов из P_r, то недетерминированным образом выбирается выполнение одного из этих действий или никакое действие не выполняется, а оператор наблюдает один из отказов. Это отличается от $\mathfrak{R}/\mathfrak{Q}$-семантики, где при отказе никакое действие не может выполняться, поскольку $P_r=\{P_q\}$.

Обозначим множество действий и отказов, порождаемых состоянием s LTS-модели $\mathbf{S}$, когда нажата кнопка P:

$$\boldsymbol{obs}(s,P,\mathbf{S}) =_{def} \{z\in P_q \mid s\text{—}z\rightarrow\}\cup\{R\in P_r \mid \forall z\in R\cup\{\tau,\gamma\}\ \ s\text{—}z\not\rightarrow\}.$$

Для получения трасс (последовательностей наблюдений) 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 обозначаем $\boldsymbol{F}(\mathbf{S})$. ***F***-трасса, все отказы которой принадлежат семейству $\mathfrak{P}_r$, называется $\mathfrak{R}$-трассой. Это те трассы, которые могут наблюдаться на машине тестирования в $\mathfrak{P}$-семантике (дивергенция и разрушение считаются условно наблюдаемыми). 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 $\mathfrak{P}$-семантике.

Обозначим множество действий и отказов, продолжающих трассу во множестве ***F***-трасс LTS-модели $\mathbf{S}$, то есть порождаемых всеми состояниями после трассы σ LTS-модели $\mathbf{S}$, после нажатия кнопки P:

$$\boldsymbol{obs}(\sigma,P,\mathbf{S}) =_{def} \{u\in P \mid \sigma\cdot\langle u\rangle\in\boldsymbol{F}(\mathbf{S})\} = \cup\{\boldsymbol{obs}(s,P,\mathbf{S}) \mid s\in(S\ \boldsymbol{after}\ \sigma)\}.$$

1.4. Гипотеза о безопасности и безопасная конформность

На уровне модели безопасное тестирование, прежде всего, предполагает формальное определение 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 $P \in \mathfrak{P}$ после $\mathfrak{R}$-трассы σ не может означать a) попытку выхода из дивергенции (после трассы нет дивергенции), b) не может вызывать разрушение (после действия, разрешаемого кнопкой), и c) не может привести к тупику. В произвольной $\mathfrak{P}$-семантике для определения условий a) и b) достаточно $\boldsymbol{F}$-трасс модели, но для определения условия c), вообще говоря, недостаточно $\boldsymbol{F}$-трасс, и приходится использовать LTS-модель. Последнее условие означает, что в каком бы стабильном состоянии после трассы не оказалась реализация, при нажатии кнопки P будет какое-либо наблюдение.

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

P ***safe in*** $\mathbf{I}$ ***after*** σ $=_{\text{def}}$
P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{I}$ ***after*** σ & $\forall s \in (\mathbf{I}$ ***after*** $\sigma)$ $(\boldsymbol{stab}(s, \mathbf{I}) \Rightarrow \boldsymbol{obs}(s, P, \mathbf{I}) \neq \varnothing)$.

Существует важный частный случай, когда вся информация, необходимая для определения отношения ***safe in***, содержится в $\boldsymbol{F}$-трассах модели. Это случай, когда для каждой кнопки P все $\mathfrak{R}$-отказы состоят только из разрешаемых действий $\cup P_r \subseteq P_q$:

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

В спецификации отношение безопасности кнопок, называемое ***safe by***, определяется не однозначно. Фактически, речь идёт о правилах, которым должно быть подчинено это отношение. Задание спецификации означает задание не только LTS-модели $\mathbf{S}$, но и отношения ***safe by***, подчиняющегося этим правилам. Таких правил два. Правило 1): если кнопка безопасна после трассы, то её нажатие не может вызвать разрушение и попытку выхода из дивергенции, а кроме того, хотя бы в одном (не обязательно в каждом) состоянии после трассы возможно наблюдение при нажатии этой кнопки (в этом состоянии нет тупика). Правило 2): если после трассы (хотя бы в одном состоянии после трассы) есть некоторое наблюдение, которое можно получить, нажимая кнопку, не приводящую к разрушению или попытке выхода из дивергенции, то одна из таких кнопок должна быть безопасна после трассы. Это правило требует «максимального» использования спецификации, то есть в ней не должно быть трасс, которые не могут быть проверены при тестировании, за исключением, конечно, таких трасс, проверка которых может вызвать разрушение или попытку выхода из дивергенции. Такое отношение ***safe by*** всегда существует: достаточно объявить безопасной каждую кнопку, которая не приводит к разрушению или попытке выхода из дивергенции и разрешает наблюдение, продолжающее трассу: $\forall P \in \mathfrak{P}$ $\forall u$

1) P ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow$
   P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** σ & $\exists s \in (\mathbf{S}$ ***after*** $\sigma)$ ***obs***$(s,P,\mathbf{S}) \neq \varnothing$.
2) P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** σ & $\exists s \in (\mathbf{S}$ ***after*** $\sigma)$ $u \in$ ***obs***$(s,P,\mathbf{S}) \Rightarrow$
   $\exists R \in \mathfrak{P}$ R ***safe by*** $\mathbf{S}$ ***after*** σ & $u \in$ ***obs***$(s,R,\mathbf{S})$.

Заметим, что в последней строке вместо $u \in$ ***obs***$(s,R,\mathbf{S})$ можно написать просто $u \in R$.

В отличие от отношения ***safe in***, правила отношения ***safe by*** всегда могут быть определены только в терминах ***F***-трасс модели. Это можно объяснить тем, что спецификация должна говорить о безопасности или опасности реализации, а также о её конформности или неконформности, только на основании трасс.
$\forall P \in \mathfrak{P}$ $\forall u$

1) P ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow$
   P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** σ & ***obs***$(\sigma,P,\mathbf{S}) \neq \varnothing$.
2) P ***safe***$_{\gamma\Delta}$***in*** $\mathbf{S}$ ***after*** σ & $u \in$ ***obs***$(\sigma,P,\mathbf{S}) \Rightarrow$
   $\exists R \in \mathfrak{P}$ R ***safe by*** $\mathbf{S}$ ***after*** σ & $u \in$ ***obs***$(\sigma,R,\mathbf{S})$.

Заметим, что в последней строке вместо $u \in$ ***obs***$(\sigma,R,\mathbf{S})$ можно написать просто $u \in R$.

Безопасность кнопок определяет безопасность наблюдений (действий и $\mathfrak{R}$-отказов) после $\mathfrak{R}$-трассы. Наблюдение безопасно, если безопасна какая-нибудь кнопка, которой оно принадлежит. 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 ***SafeIn***$(\mathbf{I})$ и ***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:

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

Следует отметить, что гипотеза о безопасности не проверяема при тестировании и является его предусловием. После этого можно определить отношение (безопасной) *конформности*: реализация **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:

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

1.5. Отказы и множество разрешаемых действий

Выше мы указали на важный подкласс $\mathfrak{P}$-семантик *с ограничением вложенности*: для каждой кнопки P все $\mathfrak{R}$-отказы состоят только из разрешаемых действий $\cup \mathrm{P}_r \subseteq \mathrm{P}_q$. Этот подкласс семантик оказывается не эквивалентным классу всех $\mathfrak{P}$-семантик. Мы покажем, что существует такая семантика $\mathfrak{P}$, такая спецификация **S** и такая реализация **I**, которая безопасна $\mathbf{I}\ \textit{safe for}_{\mathfrak{P}}\ \mathbf{S}$, но не конформна $\mathbf{I}\ \textit{saco}_{\mathfrak{P}}\ \mathbf{S}$, а для любой семантики $\mathfrak{P}1$ с ограничением вложенности имеет место $\mathbf{I}\ \textit{saco}_{\mathfrak{P}1}\ \mathbf{S}$. Это означает, что семантики без ограничения вложенности обладают большей способностью различения моделей, чем семантики с таким ограничением.

Рассмотрим пример на Рис.3.

Здесь для семантики $\mathfrak{P}$ без ограничения вложенности реализация безопасна $\mathbf{I}\ \textit{safe for}_{\mathfrak{P}}\ \mathbf{S}$, но не конформна спецификации $\mathbf{I}\ \textit{saco}_{\mathfrak{P}}\ \mathbf{S}$. Действительно, трасса $\sigma=\langle\{\mathrm{b}\},\mathrm{a}\rangle$ безопасна в реализации и спецификации: $\sigma\in\textit{SafeBy}(\mathbf{S})\cap\textit{SafeIn}(\mathbf{I})$. Кнопка $\mathrm{P}=\{\mathrm{b}\}$ безопасна в спецификации после этой трассы: $\mathrm{P}\ \textit{safe by}\ \mathbf{S}\ \textit{after}\ \sigma$. В реализации после нажатия этой кнопки может наблюдаться действие: $\mathrm{b}\in\textit{obs}(\sigma,\mathrm{P},\mathbf{I})$, а в спецификации – не может: $\mathrm{b}\notin\textit{obs}(\sigma,\mathrm{P},\mathbf{S})$. Заметим, что после трассы $\mu=\langle\mathrm{a}\rangle$ любая кнопка из $\mathfrak{P}$

безопасна в спецификации (и в реализации) и любое наблюдение, возможное в реализации (действие a или b), возможно также в спецификации.

Семантики с ограничением вложенности могут содержать только следующие кнопки:
{a}, {a,{a}}
{b}, {b,{b}}, {a,b,{a}}, {a,b,{b}}, {a,b,{a,b}}, {a,b,{a},{b}}.
В спецификации в самом начале (после пустой трассы) безопасны только кнопки первой строки, поскольку действие b разрушающее. Следовательно, трасса $\sigma=\langle\{b\},a\rangle$ оказывается опасной в спецификации. Остаётся безопасной трасса $\mu=\langle a\rangle$, но после неё любое наблюдение, возможное в реализации (действие a или b), возможно также в спецификации. Тем самым, в любой семантике $\mathfrak{P}1$ с ограничением вложенности имеет место $\mathbf{I}\ saco_{\mathfrak{P}1}\ \mathbf{S}$.

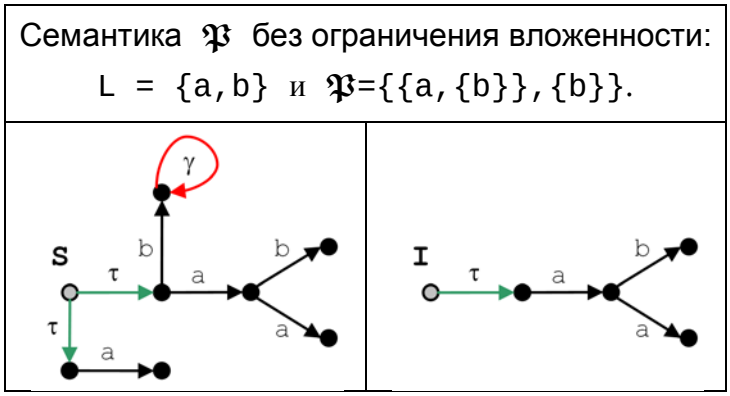


Рис.3.

1.6. 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) [14,16]. 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. Операцию «подчёркивание» распространим на множества действий: $\underline{A}=\{\underline{a}\mid a\in A\}$. Параллельное выполнение двух LTS в алфавитах A и B понимается так, что переходы по противоположным действиям z и $\underline{z}$, где $z\in A\cap\underline{B}$, 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 выполняется в одной из LTS при сохранении состояния другой LTS. Результатом композиции двух LTS $\mathbf{I}$ и $\mathbf{T}$ становится LTS $\mathbf{I} \between \mathbf{T}$ в алфавите $A \between B =_{def} (A \setminus \underline{B}) \cup (B \setminus \underline{A})$. Её состояния – это пары состояний it LTS-операндов, начальное состояние – это пара начальных состояний, а переходы порождаются следующими правилами вывода:

(1) $z \in (A \cup \{\tau,\gamma\}) \setminus \underline{B}$ & $i \text{—} z \to i`$ $\vdash$ $it \text{—} z \to i`t$,

(2) $z \in (B \cup \{\tau,\gamma\}) \setminus \underline{A}$ & $t \text{—} z \to t`$ $\vdash$ $it \text{—} z \to it`$,

(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 $\mathbf{I}$ в алфавите A и LTS-теста $\mathbf{T}$ в противоположном алфавите $B = \underline{A}$. Мы будем предполагать, что в тесте нет разрушения и τ-переходов. Для обнаружения $\mathfrak{R}$-отказа R в тесте (но не в реализации!) допускается специальный переход по $\mathfrak{R}$-отказу, который может срабатывать тогда, когда в реализации нет τ- и γ-переходов, а также переходов по действиям из R. Для удобства мы будем записывать в тесте переход не по R, а по множеству противоположных действий $\underline{R}$.

(4) $\forall z \in R \cup \{\tau,\gamma\}$ $i \text{—} z \not\to$ & $t \text{—} \underline{R} \to t`$ $\vdash$ $it \text{—} \tau \to it`$.

Заметим, что переход по $\mathfrak{R}$-отказу играет ту же роль, что θ-переход в $\mathfrak{R}/\mathfrak{Q}$-семантике. Но здесь есть два важных отличия. Во-первых, при нажатии $\mathfrak{P}$-кнопки P переход по $\mathfrak{R}$-отказу $R \in P_r$ может срабатывать даже тогда, когда в реализации могут выполняться действия $z \in P_q \setminus R$. В то же время θ-переход срабатывает только тогда, когда ни одно действие не может выполняться, поскольку $P_r = \{P_q\}$. Во-вторых, в $\mathfrak{P}$-семантике кнопка не однозначно определяет возможный отказ, поскольку множество P_r может содержать несколько отказов. Также множество разрешаемых действий, то есть действий, противоположных тем, которыми помечены переходы в состоянии теста, в $\mathfrak{P}$-семантике не однозначно определяет возможный в тесте переход по $\mathfrak{R}$-отказу, поскольку может быть несколько $\mathfrak{P}$-кнопок P с одним и тем же множеством разрешаемых действий P_q, но с разными множествами отказов P_r. В отличие от этого, в $\mathfrak{R}/\mathfrak{Q}$-семантике множество разрешаемых действий, образующее $\mathfrak{R}$-

кнопку, однозначно определяет соответствующий $\mathfrak{R}$-отказ, поскольку $P_r=\{P_q\}$. Это и является причиной того, почему в $\mathfrak{P}$-семантике мы говорим о переходе по $\mathfrak{R}$-отказу, а не о θ-переходе.

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 $\mathbf{I}|\downarrow\mathbf{T}$. 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, которые могут пониматься как однозначная инструкция оператору машины (без лишнего недетерминизма). Для этого множество наблюдений, для которых определены переходы в данном состоянии теста, должно совпадать с какой-нибудь кнопкой $P\in\mathfrak{P}$ (точнее, с $\underline{P}$, поскольку при композиции CCS тест определяется в противоположном алфавите). Множество внешних действий, для которых определены переходы в данном состоянии теста, должно быть множеством $\underline{P_q}$ действий, противоположных разрешаемым действиям, а множество отказов, по которым определены переходы, должно совпадать с множеством $\underline{P_r}$. Оператор, исполняя тест, однозначно определяет, какую кнопку ему нужно нажимать в данном состоянии теста, и для каждого возможного наблюдения у него есть «инструкция» по дальнейшему поведению, задаваемая соответствующим переходом теста по этому наблюдению.

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 наблюдением вставляется какая-нибудь безопасная (после префикса трассы) $\mathfrak{P}$-кнопка P, которой это наблюдение принадлежит. Перед каждым $\mathfrak{R}$-отказом R вставляется безопасная кнопка P такая, что $R\in P_r$, а перед каждым действием

z – безопасная кнопка P, разрешающая это действие, то есть $z \in P_q$. Безопасность трассы гарантирует 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 $\mathfrak{P}$-историей. По ней и строится LTS-тест (Рис.4). Его состояниями становятся расставленные кнопки, начальное состояние – это первая в трассе кнопка, символы переходов из состояния-кнопки P – это все возможные наблюдения (точнее, противоположные им): действия $z \in \underline{P_q}$ и отказы $R \in \underline{P_r}$. 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, фиксируют обнаруженные ошибки (выносят вердикт ***fail***, а не ***pass***). Любой строгий тест можно заменить на объединение примитивных тестов, которое обнаруживает те же самые ошибки.

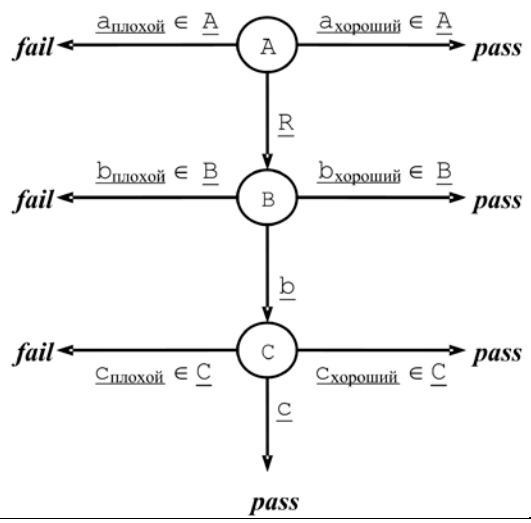
Примитивный тест для безопасной $\mathfrak{R}$-трассы σ	
	Для $R \in \mathfrak{P}_r$ и $b,c \in L$ $\sigma = \langle R,b,c \rangle \Rightarrow$ вставляем безопасные кнопки $\Rightarrow$ $\Rightarrow \langle A,R,B,b,C,c \rangle$ такие, что: $A,B,C \in \mathfrak{P}$, $R \in A$, $b \in B$, $c \in C$. Наблюдение $\underline{u}$ имеет индекс «$_{хороший}$», если наблюдение u есть в спецификации после префикса трассы; иначе – индекс «$_{плохой}$».

Рис.4.

На практике возникают две основные проблемы, связанные с применением теории тестирования конформности. Это проблема недетерминизма и глобального тестирования и проблема бесконечности полного тестового набора. Эти проблемы в общем виде рассматривались в [4,6,7,8] для $\mathfrak{R}/\mathfrak{Q}$-семантики. Обобщение до $\mathfrak{P}$-семантики не вносит в это рассмотрение ничего

принципиально нового. Поэтому мы не будем здесь останавливаться на этих проблемах.

2. 𝔓-семантика с приоритетами

До сих пор мы предполагали отсутствие приоритетов между действиями, которые тестируемая система может выполнять в данной ситуации: любое определённое в реализации и разрешённое оператором действие может быть выполнено (выбор одного из таких действий выполняется недетерминированным образом). 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. При наличии дивергенции запрос, поступающий извне, может бесконечно долго игнорироваться системой, если он имеет тот же приоритет, что внутренняя активность. Если речь идёт о составной системе, собранной из нескольких компонентов, то дивергенция может быть естественным результатом бесконечного 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 реакций реактивной системой. Для того, чтобы такую цепочку можно было прервать, заставив систему обрабатывать поступающий извне стимул, приём стимула должен иметь больший приоритет, чем выдача реакций.

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 стимулы, посылаемые тестом, не доходят до реализации, бесконечно накапливаясь в очереди. Соответственно, для выходной очереди это означает, что тест может не получать никаких реакций 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.

В существующих теориях тестирования конформности (conformance testing) подразумевается отсутствие приоритетов [10]. Это не даёт возможности проверять при тестировании выполнение тех требований к системе, которые могут быть выражены только в форме приоритетов. В [8] предложен способ введения приоритетов для $\mathfrak{R}/\mathfrak{Q}$-семантики. В данной статье мы распространяем этот способ на $\mathfrak{P}$-семантику. При таком распространении, на самом деле, мало что меняется, за исключением того, что при нажатии кнопки наблюдаемый отказ не обязательно совпадает с множеством разрешаемых оператором действий (от которого и зависят приоритеты).

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 $s \text{—} z \rightarrow s'$ 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. В то же время 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 $\Pi=\{\pi:\mathcal{P}(L)\to\mathbf{Bool}\}$: $S=LTS(V_s,L\times\Pi,E_s,s_0)$. Переход $s\text{—}z,\pi\to s'$ может выполняться только тогда, когда оператор разрешает такое множество внешних действий $Q\subseteq L$, что $z\in Q\cup\{\tau,\gamma\}$ и $\pi(Q)=\mathbf{true}$. Если есть несколько выполнимых действий, выполняется одно из них, выбираемое, по-прежнему, недетерминированным образом.

Предикат можно понимать как булевскую функцию от булевских переменных $z_1,z_2,\ldots$, взаимно-однозначно соответствующих внешним действиям из алфавита L. Например, для предиката $\pi=a\&\neg b\vee c$ переход $s\text{—}z,\pi\to s'$ может выполняться только тогда, когда оператор разрешил такое множество внешних действий Q, что $z\in Q\cup\{\tau,\gamma\}$ & $(a\in Q$ & $b\notin Q\ \vee\ c\in Q)$. 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 $s\text{—}z,\pi\to s'$ 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

$s—z,\eta_i→s`$. В свою очередь дизъюнкту η_i взаимно-однозначно соответствует множество Q_i тех действий, для которых $x_{ij}=z_j$. При композиции это множество является множеством действий, разрешаемых партнёром. Тем самым, мы можем считать, что на переходах написаны не произвольные предикаты, а множества разрешаемых действий $s—z,P_{iq}→s`$ или кнопки $s—z,P_i→s`$. Когда нажимается некоторая кнопка P_i, выполняться могут только переходы вида $s—z,P_{iq}→s`$ или $s—z,P_i→s`$, где $z∈P_{iq}∪\{\tau,\gamma\}$. Такой переход от LTS с предикатами к LTS с кнопками аналогичен переходу от расширенных автоматов (EFSM – Extended Finite State Machine) к обычным автоматам (FSM). Для заданной $\mathfrak{P}$-семантики речь идёт только о тех кнопках, которые принадлежать семейству $\mathfrak{P}$, переходы по другим кнопкам при тестировании, фактически, игнорируются – они не могут выполняться при тех тестовых возможностях, которые задаются семантикой.

2.2. Стабильность, отказы, разрушение и дивергенция

Для машины без приоритетов отказ R наблюдается в стабильном состоянии (состоянии без τ- и γ-переходов), в котором нет переходов по действиям из R. Если есть приоритеты, то меняется, прежде всего, само понятие стабильности. Оно становится условным, то есть зависящим от множества разрешённых действий P_q: состояние s LTS $\mathbf{S}$ стабильно, если для всех τ- и γ-переходов из этого состояния их предикаты от P_q ложны $\pi(P_q)=\mathit{false}$.

$\mathit{stab}(s,P,\mathbf{S}) =_{\text{def}} \nexists\pi\ \pi(P_q)\ \&\ (s—\tau,\pi→\ \vee\ s—\gamma,\pi→)$.

Соответственно меняется условие наблюдения отказа $R∈P_r$: отказ наблюдается в стабильном состоянии, в котором для всех переходов $s—z,\pi→$ по действиям $z∈R$ их предикаты ложны $\pi(P_q)=\mathit{false}$.

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 кнопка P с $P_q=\varnothing$, не разрешающая ни одного внешнего действия. 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 $R \in P_r$ означал невозможность выполнения внешних действий $z \in R$, а также τ- и γ-переходов, поскольку их предикаты стали ложны $\pi(P_q)=\mathbf{\mathit{false}}$. После отжатия кнопки P множество разрешённых внешних действий пусто, и теперь могут выполняться τ- и γ-переходы с предикатами $\pi(\varnothing)=\mathbf{\mathit{true}}$. Далее оператор может снова нажать ту же кнопку 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 $\pi(\varnothing)=\mathbf{\mathit{true}}$. 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 при отсутствии наблюдения по первому воздействию. Более подробно переключение кнопок рассматривается в следующем подразделе.

При отсутствии приоритетов разрушение возможно либо с самого начала до наажатия какой-либо кнопки, либо после выполнения некоторого внешнего действия, разрешённого нажатой кнопкой. В первом случае любое тестирование опасно (машину нельзя включать). При наличии приоритетов выполнимость перехода по разрушению $s \text{—} \gamma, \pi \rightarrow$, как для любого перехода, определяется предикатом π, который зависит от множества разрешённых действий. Поэтому следует говорить о выполнимом или не выполнимом разрушении в зависимости от нажатой кнопки. Разрушение может стать выполнимым при нажатии или отжатии кнопки, то есть разрушение возможно не только с самого начала или после внешнего действия, но также сразу после нажатия кнопки P, если $\pi(P_q)=\mathbf{\mathit{true}}$, или после наблюдения отказа из P_r, если $\pi(\varnothing)=\mathbf{\mathit{true}}$. В последнем случае, правда, разрушение было выполнимым и до нажатия кнопки P.

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.5 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, например, с помощью кнопки {?b}, а потом переключить эту кнопку на кнопку, посылающую стимул ?a, например, {?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, то есть нажималась не кнопка {?b}, а кнопка {?b,{?b}}, можно сначала дождаться блокировки {?b}, а потом послать стимул ?a.

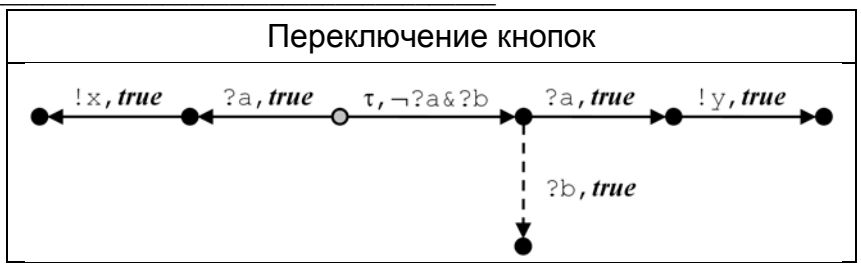


Рис.5.

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» тупики: если оператор переключает кнопку `P` на (любую) другую кнопку `Q`, то возникновение тупика при нажатии кнопки `P` не препятствует такому переключению (как на Рис.5, когда нет пунктирного перехода по стимулу `?b`, а блокировка `{?b}` не наблюдаема при посылке стимула `?b`). Другое дело, что, если возможен тупик, то при безопасном тестировании мы не можем нажимать кнопку `P` без последующего переключения на другую кнопку, то есть с ожиданием наблюдения (на Рис.5 без пунктирного перехода кнопку `{?b}` всегда нужно переключать на кнопку `{?a}` или какую-то другую). Тем самым, если можно переключать кнопки, условие безопасности 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.6 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 пошлёт стимул ?a, реакция !z – только если оператор не будет торопиться, а реакция !y – только при средней скорости работы.

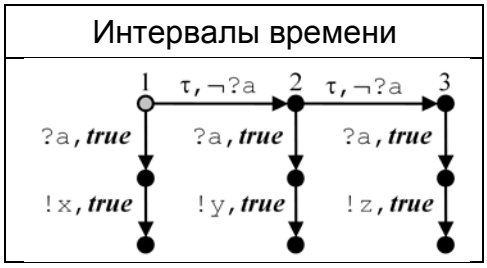


Рис.6.

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 Ω-кнопку P от отказа (и то и другое – подмножество внешних действий), мы будем любую 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, и после них история не может продолжаться, аналогично трассам. Очевидно, что в истории каждому внешнему действию z непосредственно предшествует кнопка "P", разрешающая это действие $z \in P_q$, а каждому отказу R – $\mathfrak{R}$-кнопка "P", допускающая этот отказ $R \in P_r$. Могут или не могут идти две кнопки подряд, зависит от того, разрешено или запрещено переключение кнопок.

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

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

Пустая $\mathfrak{P}$-история заканчивается в состояниях, достижимых из начального состояния по $\varnothing$-выполнимым τ-маршрутам, то есть после включения машины до нажатия какой-либо кнопки. Пусть $\mathfrak{P}$-история σ заканчивается во множестве состояний $\mathbf{S}\ \textbf{\textit{after}}\ \sigma$. Рассмотрим различные продолжения этой $\mathfrak{P}$-истории. Мы будем предполагать, что $\mathfrak{P}$-история не заканчивается разрушением или дивергенцией, поскольку после дивергенции и разрушения нет продолжений.

Продолжение кнопкой P, где $P \in \mathfrak{P}$. Если допускается переключение кнопок, такое продолжение всегда возможно. Если переключения кнопок нет, $\mathfrak{P}$-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 $\mathbf{S}\ \textbf{\textit{after}}\ \sigma$, а затем продолжить выполнение по любому P_q-выполнимому τ-маршруту. Множество концов таких маршрутов и будет множеством состояний $\mathbf{S}\ \textbf{\textit{after}}\ \sigma \cdot \langle \text{"P"} \rangle$. Заметим, что, если история не заканчивалась на кнопку, то концы всех $\varnothing$-выполнимых τ-маршрутов уже входят в $\mathbf{S}\ \textbf{\textit{after}}\ \sigma$.

Продолжение внешним действием z. Такое продолжение возможно только в том случае, когда сама $\mathfrak{P}$-история имеет вид $\sigma\cdot\langle\text{“P”}\rangle$, где $P\in\mathfrak{P}$ и $z\in P_q$, то есть заканчивается кнопкой P, разрешающей действие z. Наблюдение действия z происходит, когда совершается P_q-выполнимый переход по z из состояния после предшествующей $\mathfrak{P}$-истории, то есть переход $s\text{—}z,\pi\to s'$, где $s\in(S\ \textit{after}\ \sigma\cdot\langle\text{“P”}\rangle)$ и $\pi(P_q)=\textit{true}$. 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 $S\ \textit{after}\ \sigma\cdot\langle\text{“P”},z\rangle$.

Продолжение $\mathfrak{R}$-отказом R. Такое продолжение возможно только в том случае, когда сама $\mathfrak{P}$-история имеет вид $\sigma\cdot\langle\text{“P”}\rangle$, где $P\in\mathfrak{P}$ и $R\in P_r$. Отказ R возникает в таком состоянии $s\in(S\ \textit{after}\ \sigma\cdot\langle\text{“P”}\rangle)$, в котором выполнено условие: для каждого перехода $s\text{—}z,\pi\to s'$, где $z\in R\cup\{\tau,\gamma\}$ должно быть $\pi(P_q)=\textit{false}$. 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 $S\ \textit{after}\ \sigma\cdot\langle\text{“P”},R\rangle$.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 $s\in(S\ \textit{after}\ \sigma)$ переход $s\text{—}\gamma,\pi\to s'$ *P_q-выполнимым*, если $\mathfrak{P}$-история заканчивается кнопкой $P\in\mathfrak{P}$ (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. Заметим, что если разрушение может возникнуть после отказа при нажатии кнопки, то оно могло возникнуть и до нажатия этой кнопки: если возможна история $\sigma\cdot\langle\text{“P”},R,\gamma\rangle$, где $R\in P_r$, то возможна также история $\sigma\cdot\langle\gamma\rangle$.

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 $\mathfrak{P}$-истории вида $\sigma\cdot\langle\text{“}P\text{”}\rangle$, где $P\in\mathfrak{P}$, если есть бесконечный P_q-выполнимый τ-маршрут, начинающийся в состоянии из $\mathbf{S}$ ***after*** $\sigma\cdot\langle\text{“}P\text{”}\rangle$ (очевидно, достаточно считать, что маршрут начинается в состоянии из $\mathbf{S}$ ***after*** σ). В этом случае символ Δ будет продолжать $\mathfrak{P}$-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 $\mathfrak{P}$-истории для $\mathfrak{P}=\mathcal{P}(L\cup\mathcal{P}(L))$, когда любой набор внешних действий и множеств внешних действий является $\mathfrak{P}$-кнопкой. Множество ***F***-историй LTS $\mathbf{S}$ – обозначим так же, как множество ***F***-трасс, – $\boldsymbol{F}(\mathbf{S})$, поскольку в дальнейшем мы будем рассматривать только истории, а не трассы. Теперь $\mathfrak{P}$-история LTS – это такая её ***F***-история, в которой встречаются кнопки только из семейства $\mathfrak{P}$, а отказы – это только $\mathfrak{R}$-отказы (отказы из $\mathfrak{P}_r$).

Обозначим множество действий и отказов, порождаемых состоянием s LTS-модели $\mathbf{S}$, когда нажата кнопка P:

$$\boldsymbol{obs}(s,P,\mathbf{S}) =_{\text{def}} \{z\in P_q \mid \exists\pi\ \pi(P_q)\ \&\ s\text{—}z,\pi\rightarrow\} \cup \{R\in P_r \mid \forall z\in R\cup\{\tau,\gamma\}\ \nexists\pi\ \pi(P_q)\ \&\ s\text{—}z,\pi\rightarrow\}.$$

Обозначим множество действий и отказов, продолжающих историю во множестве ***F***-историй LTS-модели $\mathbf{S}$, то есть порождаемых всеми состояниями после истории σ LTS-модели $\mathbf{S}$, после нажатия кнопки P:

$$\boldsymbol{obs}(\sigma,P,\mathbf{S}) =_{\text{def}} \{u\in P \mid \sigma\cdot\langle\text{“}P\text{”},u\rangle\in\boldsymbol{F}(\mathbf{S})\} = \cup\{\boldsymbol{obs}(s,P,\mathbf{S}) \mid s\in(S\ \textit{after}\ \sigma)\}.$$

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 $\mathfrak{P}$-истории, безопасность или опасность

кнопки определяется только после $\mathfrak{P}$-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 сразу после нажатия кнопки и после $\mathfrak{R}$-отказа. В последнем случае, если после нажатия кнопки наблюдается отказ а потом (после автоматического отжатия кнопки) возникает разрушение, то это разрушение было выполнимо и до нажатия кнопки. В дальнейшем нас не будет интересовать безопасность кнопок после опасных историй, то есть таких, прохождение которых может вызывать разрушение. Поэтому случай разрушения после отказа можно не рассматривать.

Определение отношения безопасности в реализации без переключения кнопок:

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

P ***safe$_1$in*** $\mathbf{I}$ ***after*** $\sigma =_{\text{def}} P$ ***safe$_\gamma$in*** $\mathbf{I}$ ***after*** σ & $\sigma\cdot\langle\text{“P”},\Delta\rangle\notin\boldsymbol{F}(\mathbf{I})$
& $\forall s\in(\mathbf{I}$ ***after*** $\sigma\cdot\langle\text{“P”}\rangle)$ $(\boldsymbol{stab}(s,P,\mathbf{I}) \Rightarrow \boldsymbol{obs}(s,P,\mathbf{I})\neq\varnothing)$.

Случай, когда для каждой кнопки P все $\mathfrak{R}$-отказы состоят только из разрешаемых действий $\cup P_r\subseteq P_q$:

P ***safe$_1$in*** $\mathbf{I}$ ***after*** $\sigma =_{\text{def}} P$ ***safe$_\gamma$in*** $\mathbf{I}$ ***after*** σ & $\sigma\cdot\langle\text{“P”},\Delta\rangle\notin\boldsymbol{F}(\mathbf{I})$
& $(P_r=\varnothing \Rightarrow \sigma\cdot\langle\text{“P”},P_q\rangle\notin\boldsymbol{F}(\mathbf{I}))$.

Требования к отношению безопасности в спецификации без переключения кнопок: $\forall P\in\mathfrak{P}\ \forall u$

1) P ***safe$_1$by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ & $\sigma\cdot\langle\text{“P”},\Delta\rangle\notin\boldsymbol{F}(\mathbf{S})$
& $\exists s\in(\mathbf{S}$ ***after*** $\sigma\cdot\langle\text{“P”}\rangle)$ $\boldsymbol{obs}(s,P,\mathbf{S})\neq\varnothing$.
2) P ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ & $\sigma\cdot\langle\text{“P”},\Delta\rangle\notin\boldsymbol{F}(\mathbf{S})$ & $\exists s\in(\mathbf{S}$ ***after*** $\sigma\cdot\langle\text{“P”}\rangle)$
$u\in\boldsymbol{obs}(s,P,\mathbf{S}) \Rightarrow \exists R\in\mathfrak{P}\ R$ ***safe$_1$by*** $\mathbf{S}$ ***after*** σ & $u\in\boldsymbol{obs}(s,R,\mathbf{S})$.

Аналогично случаю отсутствия приоритетов, правила отношения ***safe by*** всегда могут быть определены только в терминах ***F***-историй модели:
$\forall P\in\mathfrak{P}\ \forall u$

1) P ***safe$_1$by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ & $\sigma{\cdot}\langle\text{“P”}, \Delta\rangle \notin \boldsymbol{F}(\mathbf{S})$
& $\exists u \in P\ \sigma{\cdot}\langle\text{“P”}, u\rangle \in \boldsymbol{F}(\mathbf{S})$.

2) P ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ & $\sigma{\cdot}\langle\text{“P”}, \Delta\rangle \notin \boldsymbol{F}(\mathbf{S})$
& $u \in \boldsymbol{obs}(\sigma, P, \mathbf{S}) \Rightarrow \exists R \in \mathfrak{P}$ R ***safe$_1$by*** $\mathbf{S}$ ***after*** σ & $u \in \boldsymbol{obs}(\sigma, R, \mathbf{S})$.

На основе отношений безопасности кнопок в реализации и спецификации определяются безопасные наблюдения, безопасные $\mathfrak{P}$-истории ***Safe$_1$In***$(\mathbf{I})$ и ***Safe$_1$By***$(\mathbf{S})$, гипотеза о безопасности и безопасная конформность аналогично тому, как это делалось для трасс без приоритетов.

$\mathbf{I}$ ***safe for*** $\mathbf{S}$ $=_{\text{def}}$ $(\langle\gamma\rangle \notin \boldsymbol{F}(\mathbf{S}) \Rightarrow \langle\gamma\rangle \notin \boldsymbol{F}(\mathbf{I}))$
& $\forall \sigma \in \boldsymbol{Safe_1By}(\mathbf{S}) \cap \boldsymbol{Safe_1In}(\mathbf{I})\ \forall P \in \mathfrak{P}$
$(P$ ***safe$_1$by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe$_1$in*** $\mathbf{I}$ ***after*** $\sigma)$.

$\mathbf{I}$ ***saco*** $\mathbf{S}$ $=_{\text{def}}$ $\mathbf{I}$ ***safe for*** $\mathbf{S}$
& $\forall \sigma \in \boldsymbol{Safe_1By}(\mathbf{S}) \cap \boldsymbol{Safe_1In}(\mathbf{I})\ \forall P$ ***safe$_1$by*** $\mathbf{S}$ ***after*** σ
$\boldsymbol{obs}(\sigma, P, \mathbf{I}) \subseteq \boldsymbol{obs}(\sigma, P, \mathbf{S})$.

2.7. Безопасность и конформность с переключением кнопок

Если допускается переключение кнопок, мы можем обходить запрет на возникновение тупика, а также выполняемую дивергенцию, при нажатии кнопки, просто переключая её на другую кнопку. Соответственно, модифицируются отношения безопасности: удаляются условия, подчёркнутые волнистой линией, и остаются условия, связанные только с разрушением.

Определение отношения безопасности в реализации с переключением кнопок:
P ***safe$_2$in*** $\mathbf{I}$ ***after*** $\sigma =_{\text{def}} P$ ***safe$_\gamma$in*** $\mathbf{I}$ ***after*** σ.

Требования к отношению безопасности в спецификации без переключения кнопок: $\forall P \in \mathfrak{P}\ \forall u$

1) P ***safe$_2$by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ.

2) P ***safe$_\gamma$in*** $\mathbf{S}$ ***after*** σ & $\exists s \in (\mathbf{S}$ ***after*** $\sigma{\cdot}\langle\text{“P”}\rangle)$
$u \in \boldsymbol{obs}(s, P, \mathbf{S}) \Rightarrow \exists R \in \mathfrak{P}$ R ***safe$_2$by*** $\mathbf{S}$ ***after*** σ & $u \in \boldsymbol{obs}(s, R, \mathbf{S})$.

Или, в терминах $\boldsymbol{F}$-историй модели: $\forall P \in \mathfrak{P}\ \forall u$

1) P ***safe$_2$by*** **S** ***after*** σ ⇒ P ***safe$_{\gamma\Delta}$in*** **S** ***after*** σ.
2) P ***safe$_\gamma$in*** **S** ***after*** σ
   & u∈***obs***(σ,P,**S**) ⇒ ∃R∈𝔓 R ***safe$_2$by*** **S** ***after*** σ & u∈***obs***(σ,R,**S**).

Однако возникает вопрос: сколько раз оператор может переключать кнопки? Мы исходим из следующей парадигмы тестирования: целью тестовых воздействий (нажатия кнопок) является получение наблюдений. Тест не может содержать цепочки переключений кнопок, которая не приводит гарантированно к какому-нибудь наблюдению. Поскольку мы рассматриваем только конечные (по времени выполнения) тесты, цепочка переключений кнопок должна быть конечной и заканчиваться нажатием кнопки, после которой оператору гарантируется получение какого-нибудь 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*** σ =$_{\text{def}}$ ∃P_0=P,P_1,…,P_n∈𝔓
& ∀i=0..n-1 P_i ***safe$_2$in*** **I** ***after*** σ·⟨"P_0","P_1",…,"P_{i-1}"⟩
& P_n ***safe$_1$in*** **I** ***after*** σ·⟨"P_0","P_1",…,"P_{n-1}"⟩,

P ***safe by*** **S** ***after*** σ =$_{\text{def}}$ ∃P_0=P,P_1,…,P_n∈𝔓
& ∀i=0..n-1 P_i ***safe$_2$by*** **S** ***after*** σ·⟨"P_0","P_1",…,"P_{i-1}"⟩
& P_n ***safe$_1$by*** **S** ***after*** σ·⟨"P_0","P_1",…,"P_{n-1}"⟩.

Таким образом, отношение безопасности с индексом "$_2$" определяет продолжение истории кнопкой, не вызывающей разрушение, а отношение безопасности с индексом "$_1$" дополнительно запрещает тупик и попытку выхода из выполняемой дивергенции. Понятно, что 1-безопасная кнопка также и 2-безопасна, но обратное, вообще говоря, не верно. Для полной безопасности кнопки после истории требуется, чтобы она была 2-безопасна, и, если она не 1-безопасна, то после неё можно было разместить конечную (в том числе пустую) цепочку 2-безопасных кнопок, а затем 1-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. Для n=1,2:

$$\mathbf{I}\ \textit{safe for}\ \mathbf{S} =_{\text{def}}\quad (\langle\gamma\rangle\notin \textit{F}(\mathbf{S}) \Rightarrow \langle\gamma\rangle\notin \textit{F}(\mathbf{I}))$$
$$\&\ \forall\sigma\in \textit{SafeBy}(\mathbf{S})\cap \textit{SafeIn}(\mathbf{I})\ \ \forall \mathrm{P}\in\mathfrak{P}$$
$$(\mathrm{P}\ \textit{safe}_n\textit{by}\ \mathbf{S}\ \textit{after}\ \sigma \Rightarrow \mathrm{P}\ \textit{safe}_n\textit{in}\ \mathbf{I}\ \textit{after}\ \sigma);$$
$$\mathbf{I}\ \textit{saco}\ \mathbf{S} =_{\text{def}}\quad \mathbf{I}\ \textit{safe for}\ \mathbf{S}$$
$$\&\ \forall\sigma\in \textit{SafeBy}(\mathbf{S})\cap \textit{SafeIn}(\mathbf{I})\ \ \forall \mathrm{P}\ \textit{safe}_n\textit{by}\ \mathbf{S}\ \textit{after}\ \sigma$$
$$\textit{obs}(\sigma,\mathrm{P},\mathbf{I})\subseteq \textit{obs}(\sigma,\mathrm{P},\mathbf{S}).$$

Итак, мы определили безопасность и конформность для $\mathfrak{P}$-семантики с приоритетами в двух модификациях: с переключением и без переключения кнопок. Очевидно, что $\mathfrak{P}$-семантика без приоритетов является частным случаем $\mathfrak{P}$-семантики с приоритетами, когда предикаты всех переходов тождественно истинны. При этом не важно, рассматривается семантика с переключением или без переключения кнопок.

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 $\mathbf{I}$. Пересчёт предикатов другой LTS делается аналогично. 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, вообще говоря, 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 $A\!\downarrow\!B=(A\setminus\underline{B})\cup(B\setminus\underline{A})$, хотя реально он не зависит от переменных, соответствующих действиям из $B\setminus\underline{A}$. (Предикат π_{ti}, наоборот, может зависеть от этих переменных, но не зависит от переменных, соответствующих действиям из $A\setminus\underline{B}$).

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

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

(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 в тесте не имеют предикатов, точнее их предикаты константно 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***).

Если нет переключения кнопок, то в тесте нет τ-переходов. Точнее такой переход может быть только в состоянии, соответствующем отсутствию нажатой кнопки: $t\text{—}\tau\!\to$ влечёт $\forall z\neq\tau$ $t\text{—}z\!\not\to$. Оператор выжидает, прежде, чем нажать кнопку. Таких τ-переходов из состояния t может быть несколько – оператор выбирает, какую кнопку ему нажать. Заметим, однако, что наличие в тесте τ-переходов только создаёт лишний нетедерминизм и не увеличивает мощности тестирования. Для обнаружения $\mathfrak{R}$-отказа R в тесте (но не в реализации!) используется переход по $\mathfrak{R}$-отказу аналогично случаю без приоритетов.

(4) $\forall z \in R \cup \{\tau,\gamma\}\ \nexists \pi\ \pi(P_q)$ & $i \text{—}z,\pi\rightarrow$ & $t \text{—}\underline{R}\rightarrow t'$ & $t \text{—}\tau\not\rightarrow$
$\vdash it \text{—}\tau\rightarrow it'$, где $P_q=\{z \in L \mid t \text{—}\underline{z}\rightarrow\}$.

Такому состоянию теста t соответствует кнопка $P=P_q \cup P_r$, где P_r множество $\mathfrak{R}$-отказов R, для которых в состоянии определены переходы $t \text{—}\underline{R}\rightarrow$.

Если допускается переключение кнопок, то в тесте оно отображается в виде τ-перехода из состояния, соответствующего одной кнопке, в состояние, соответствующее другой кнопке. В этом случае нужно, чтобы $\underline{R}$-переход мог срабатывать независимо от этого τ-перехода. Иными словами, из правила вывода (4) удаляется условие, подчёркнутое волнистой линией.

В композиции реализации и теста все предикаты константны, мы можем удалить все переходы с ложными предикатами. После этого при безопасном тестировании оставшиеся γ-переходы должны быть недостижимы. Тогда как и для машины без приоритетов выполнению теста соответствует прохождение τ-маршрута, начинающегося в начальном состоянии композиции и заканчивающегося в терминальном состоянии, которому назначен вердикт ***pass*** или ***fail***.

Примитивный тест строится точно так же, как для машины без приоритетов. Отличие лишь в том, что без приоритетов мы строили тест по безопасной $\mathfrak{R}$-трассе, превращая её в одну из $\mathfrak{P}$-историй, а теперь сразу начинаем с некоторой безопасной $\mathfrak{P}$-истории. Кроме того, если в истории есть переключение с кнопки P на кнопку Q, то в тесте проводится τ-переход “P”—τ→“Q”. По-прежнему, набор всех примитивных тестов полон, а любой управляемый строгий тест можно заменить на объединение примитивных тестов, которое обнаруживает те же самые ошибки.

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 реакций: $\pi(U)=(\forall !y\ !y\notin U)$. 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 $X_1, X_2, \ldots$ с линейным приоритетом: 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 стимулов 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 $\pi_{\text{север}}(U)=(\text{восток}\notin U \vee U=\{\text{север,восток,юг,запад}\})$. Аналогично устроены предикаты переходов на восток, юг и запад.

2.10. «Торговля» между партнёрами при композиции

Композиция LTS с приоритетами основана на пересчёте предикатов переходов, ведущих из состояния `i` одного операнда, в зависимости от множества действий, разрешаемых соответствующим состоянием `t` другого операнда. Разрешаемое действие – это такое синхронное действие `z`, для которого в состоянии `t` есть переход по противоположному действию $\underline{z}$. Пересчитанный предикат может стать тождественно ложным, то есть переход с таким предикатом никогда не будет выполняться. Поэтому, если учитывать только те действия, по которым есть переходы с нетождественно ложными предикатами, то пересчёт предикатов меняет множество действий, разрешаемых состоянием `i`, что, в свою очередь, могло бы повлиять на пересчёт предикатов в состоянии `t`. А тогда, в свою очередь, меняется множество действий, разрешаемых состоянием `t`, что требует нового пересчёта предикатов переходов в состоянии `i` и нового изменения множества действий, разрешаемых состоянием `i`. И так далее. Этот процесс «торговли» между партнёрами может быть бесконечным или заканчиваться, если они «договорятся» придти к какому-то согласованному решению. Необходимость такой «торговли» и её реализацию с помощью τ-переходов с предикатами мы покажем на нескольких примерах.

Сначала рассмотрим ещё раз пример приоритета приёма над выдачей: переход по стимулу имеет тождественно истинный предикат, а переход по реакции (или τ-переход) имеет предикат, истинный на любом подмножестве действий, не содержащем стимулов. Если LTS с такими приоритетами готова выполнить как приём стимула `?x`, так и выдачу реакции `!y`, а её партнёр выполняет только выдачу `!x` или только приём `?y`, то однозначно определяется передача сообщения `x` из второй LTS в первую или, соответственно, сообщения `y` из первой LTS во вторую. Также, если партнёр готов как принимать `?y`, так и выдавать `!x`, но эти действия имеют имеют равные приоритеты или выдача приоритетнее приёма, то в композиции будет гарантированно осуществляться передача сообщения `x`. Однако, если обе LTS применяют одинаковую стратегию – приоритет приёма над выдачей, то возникнет тупик: каждый хочет принимать, но не хочет выдавать. Эта стратегия изображена на Рис.7 в строке 1. Если нас это не устраивает, необходима «торговля»: выяснив, что партнёр и принимает и выдаёт, мы должны только принимать, но не выдавать. Эта стратегии изображена на Рис.7 в строке 2.

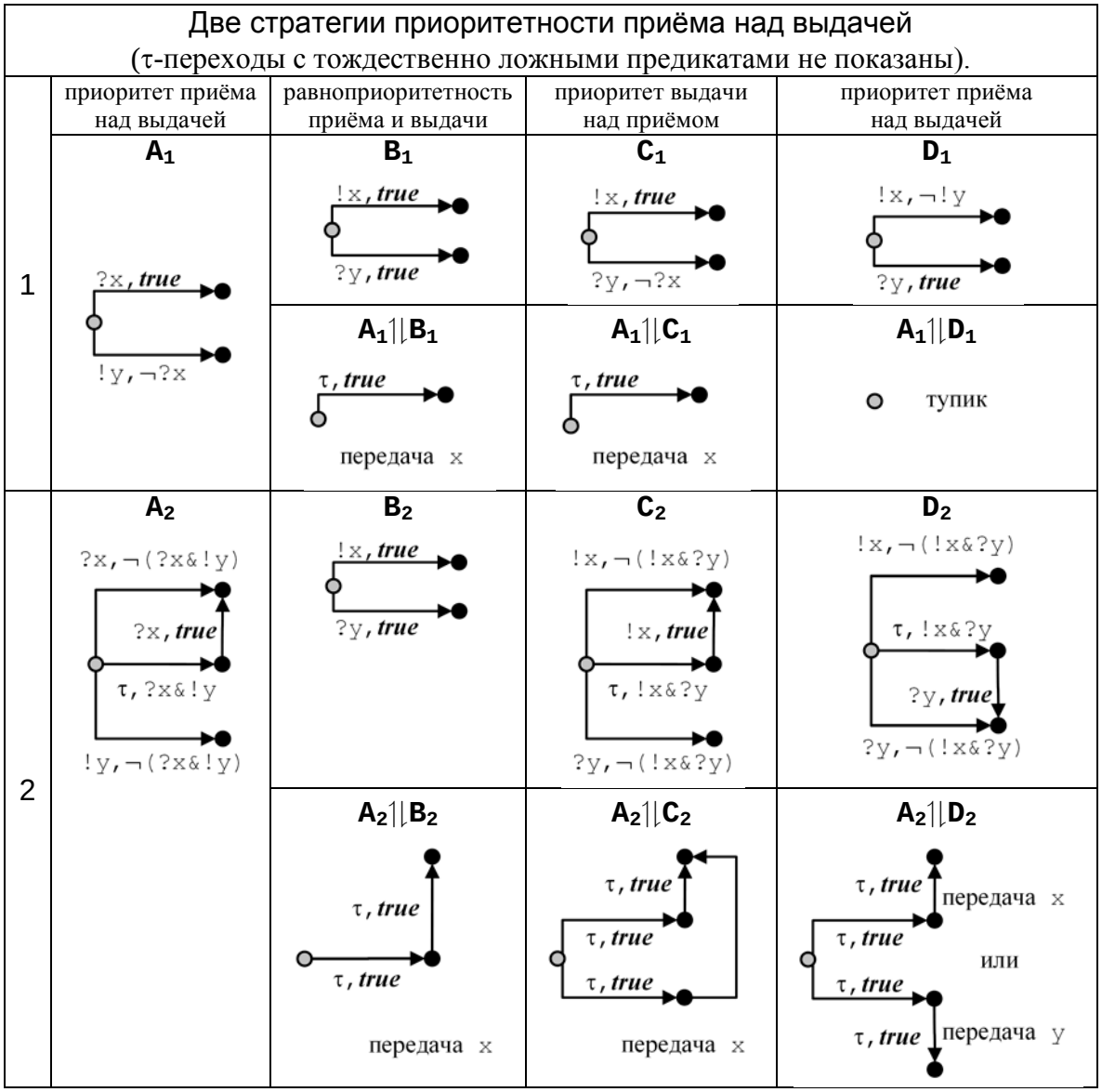


Рис.7.

Аналогичные две стратегии возможны для симметричного примера приоритета выдачи на приёмом.

Теперь рассмотрим случай, когда задан циклический приоритет действий: если данное действие не может быть выполнено, то предпринимается попытка выполнить следующее по приоритету действие, и так далее. (Нам будет безразлично, является ли действие передачей стимула, выдачей реакции или ещё каким-то действием.) На Рис.8 в строке 1 показан пример, когда таких действий три: `0`, `1` и `2` с приоритетами `0>1>2>0`; начальное действие, с которого начинается торговля, – действие `0`. Если партнёр придерживается аналогичной стратегии торговли, но начинает с действия `2`, то при композиции

возможен бесконечный цикл торговли. Чтобы его избежать, стратегия может быть скорректирована так, чтобы приоритеты перестали быть циклическими (строка 2): после того, как перебраны все действия и все они отклонены партнёром, принимается решение (показано пунктиром) «согласиться» на те действия, которые последний раз предлагал партнёр.

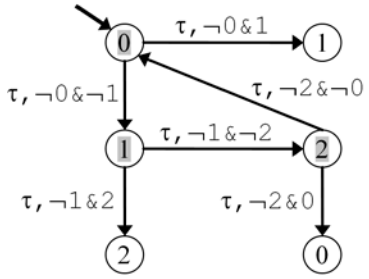
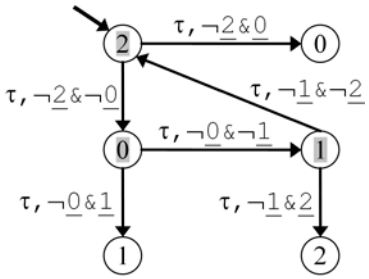
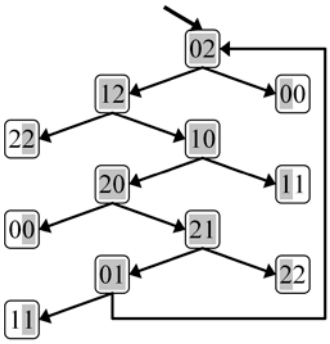
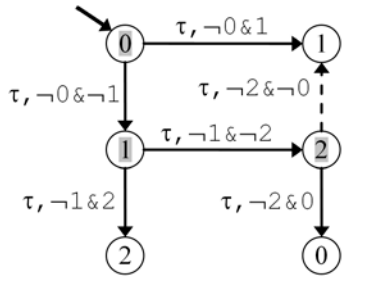
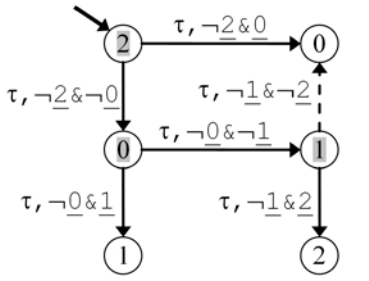
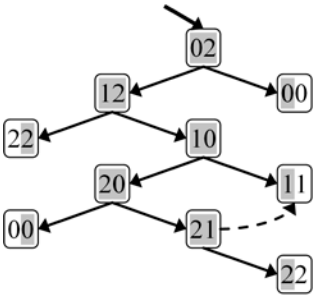
<table>
<tr><td colspan="4">Две стратегии циклического приоритета
(τ-переходы с тождественно ложными предикатами не показаны).</td></tr>
<tr><td>1</td><td>приоритеты: 0>1>2>0
A₁
 
</td><td>приоритеты: 2>0>1>2
B₁
 
</td><td>A₁⫛B₁
 
</td></tr>
<tr><td>2</td><td>приоритеты: 0>1>2
A₂
 
</td><td>приоритеты: 2>0>1
B₂
 
</td><td>A₂⫛B₁ = A₂⫛B₂
 
</td></tr>
<tr><td></td><td>В состоянии “i” определён переход по действию i с предикатом true.</td><td>В состоянии “i” определён переход по действию i с предикатом true.</td><td>В состоянии “ii” определён синхронный τ-переход (по действиям i и i) с предикатом true.</td></tr>
</table>

Рис.8.

В общем случае торговля выглядит следующим образом. Пусть в состоянии системы s определены переходы $s \text{—\!—} z,\pi \to s_{z,\pi}$ по действиям $z \in P_0$ с нетождественно ложными предикатами π. Система «узнаёт», что в её полном окружении (которое имеет противоположный алфавит и с которым она образует замкнутую систему с пустым алфавитом) определены переходы с нетождественно ложными предикатами по множеству действий Q_0. После пересчёта предикатов в системе получается множество действий P_1, а в окружении – Q_1. Пересчёт предикатов происходит недетерминированно в системе или в её окружении. Если сначала пересчитываются предикаты

системы, то мы получаем пару множеств P_1, Q_0; в противном случае – пару P_0, Q_1. Далее в первом случае происходит пересчёт предикатов в окружении и мы получаем пару P_1, Q_2, где Q_2 пересчитано на основе P_1, а во втором случае – в системе и мы получаем пару P_2, Q_1, где P_2 пересчитано на основе Q_1. И так далее (см. Рис.9 слева).

На каждом шаге у нас есть пара множеств P_i, Q_{i+1} или P_{i+1}, Q_i. В первом случае следующий пересчёт происходит в системе и следующая пара – это P_{i+2}, Q_{i+1}; во втором случае следующий пересчёт происходит в окружении и следующая пара – это P_{i+1}, Q_{i+2}. Торговля заканчивается, когда множество при пересчёте не меняется: при переходе от пары P_i, Q_{i+1} к паре P_{i+2}, Q_{i+1}, если $P_{i+2}=P_i$, или при переходе от пары P_{i+1}, Q_i к паре P_{i+1}, Q_{i+2}, если $Q_{i+2}=Q_i$.

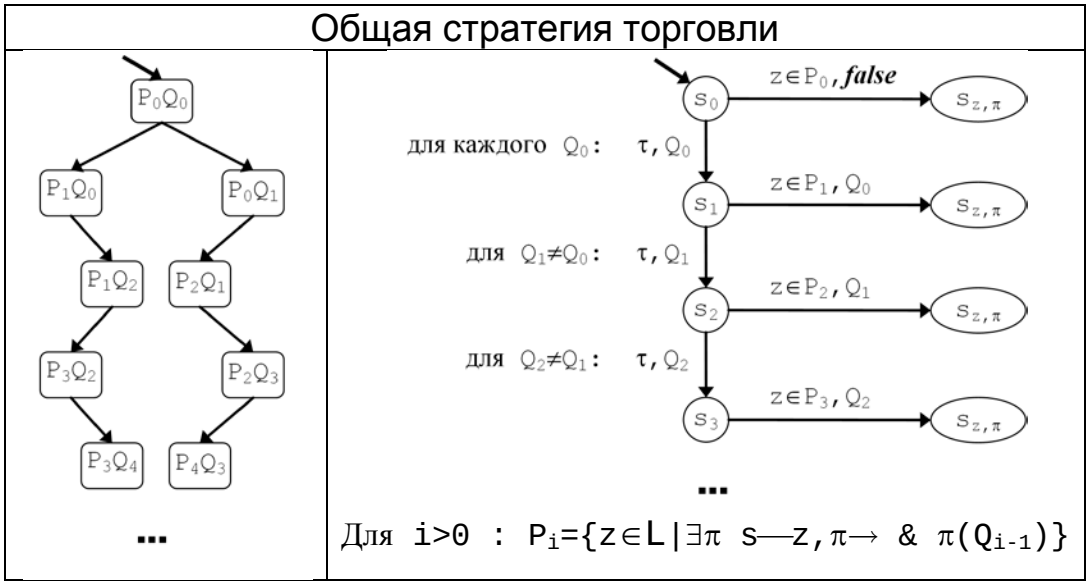


Рис.9.

Моделирование такой торговли с помощью τ-переходов с предикатами может быть выполнено следующим образом (см. Рис.9 справа). Мы будем использовать переходы по множеству разрешаемых действий: $—z, Q_i→$ означает переход $—z, \pi→$ по предикату π, который истинен только на множестве Q_i: $\pi(Q) = (Q=Q_i)$. Состояние s преобразуется во множество состояний s_i. Сначала имеем состояние s_0, в котором определены переходы $s_0—z, \mathit{false}→s_{z,\pi}$ по внешним действиям $z∈P_0$ с тождественно ложными предикатами, а также τ-переход $s_0—\tau, Q_0→s_1$ по множеству разрешаемых действий Q_0. Такой τ-переход, естественно, необходимо иметь для каждого Q_0. В конце τ-перехода находится состояние s_1, соответствующее множеству

действий P_1, которое получается после пересчёта предикатов по множеству Q_0. В этом состоянии определяются переходы $s_1 — z, Q_0 \to s_{z,\pi}$ по действиям из $z \in P_1$ по множеству Q_0. Также в состоянии 1 определяется τ-переход $s_1 — \tau, Q_1 \to s_2$ по множеству разрешаемых действий $Q_1 \neq Q_0$ (для каждого такого Q_1). В конце этого второго τ-перехода находится состояние s_2, соответствующее множеству действий P_2, которое получается после пересчёта предикатов по множеству Q_1. В этом состоянии определяются переходы $s_2 — z, Q_1 \to s_{z,\pi}$ по действиям $z \in P_2$ по множеству Q_1. Также в состоянии 2 определяется τ-переход $s_2 — \tau, Q_1 \to s_3$ по множеству разрешаемых действий $Q_2 \neq Q_1$ (для каждого такого Q_2). И так далее. Естественно, для конкретной стратегии торговли какие-то из этих состояний могут отождествляться. Если возникает бесконечная цепочка τ-переходов (в частности, цикл), то такая дивергенция соответствует «зацикливанию» при торговле.

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 ∅-выполнимы. Можно считать, что сразу после включения машины и сразу после наблюдения машина стоит, и может выполнять какие-то действия только после нажатия кнопки. Также переключение кнопок не интерпретируется как отжатие первой кнопки (с разрешением ∅-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 ∅-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 $\mathfrak{R}/\mathfrak{Q}$-семантики 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 [4,6,7].

Для $\mathfrak{R}/\mathfrak{Q}$-семантик с приоритетами проблемы монотонности и рефлексивности ещё не решены. Также эти проблемы не решены в общем случае $\mathfrak{P}$-семантики: как с приоритетами, так и без них..

Литература

1. Бурдонов И.Б., Косачев А.С. Тестирование компонентов распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
2. Бурдонов И.Б., Косачев А.С. Верификация композиции распределенной системы. Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Изд-во МГУ, 2005.
3. 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.
4. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, No. 5.
5. 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.
6. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.
7. Бурдонов И.Б. Теория конформности для функционального тестирования программных систем на основе формальных моделей. Диссертация на соискание учёной степени д.ф.-м.н., Москва, 2008. http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
8. Бурдонов И.Б., Косачев А.С. Системы с приоритетами: конформность, тестирование, композиция. Труды ИСП РАН, т. 14, 2008.
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. 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.
12. Heerink L. Ins and Outs in Refusal Testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1998.
13. 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.
14. Milner R. A Calculus of Communicating Processes. LNCS, vol. 92, Springer-Verlag, 1980.
15. Milner R. Modal characterization of observable machine behaviour. In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer, pp. 25-34.
16. Milner R. Communication and Concurrency. Prentice-Hall, 1989.