ПРОЕКТИРОВАНИЕ И ДИАГНОСТИКА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

УДК 681.3.06

И.Б. Бурдонов, А.С. Косачев

СЕМАНТИКИ ВЗАИМОДЕЙСТВИЯ С ОТКАЗАМИ, ДИВЕРГЕНЦИЕЙ И РАЗРУШЕНИЕМ. ЧАСТЬ 1. ГИПОТЕЗА О БЕЗОПАСНОСТИ И БЕЗОПАСНАЯ КОНФОРМНОСТЬ

Исследуются формальные методы тестирования конформности исследуемой системы спецификации. Семантика взаимодействия определяет тестовые возможности, сводимые к наблюдению действий и отказов (отсутствия действий). Семантика параметризуется семействами наблюдаемых и ненаблюдаемых отказов. Вводится разрушение – запрещённое действие, которого следует избегать при взаимодействии. Определяются понятие безопасного тестирования, реализационная гипотеза о безопасности и безопасная конформность. Рассматриваются теоретические аспекты генерации тестов по спецификации в заданной семантике.

Ключевые слова: *тестирование*, *конформность*, *семантика взаимодействия*, *трассы*, *отказы*, *LTS*.

Статья посвящена методам тестирования соответствия (конформности) исследуемой системы заданным требованиям (спецификации) на основе формальных моделей. Операционная семантика взаимодействия задаётся с помощью машины тестирования, определяющей тестовые возможности. Выделяется набор теоретически достаточно мощных и практически значимых возможностей, сводимый к наблюдению внешних действий и отказов (отсутствие внешних действий). Нововведениями являются: 1) Параметризация семантики семействами наблюдаемых и ненаблюдаемых отказов, что позволяет учитывать ограничения на (правильное) взаимодействие. 2) Разрушение – запрещённое действие, которое не должно выполняться в правильном взаимодействии. 3) Моделирование дивергенции Δ-действием, которого тоже следует избегать в правильном взаимодействии. Предлагаются основанные на такой семантике понятие безопасного тестирования, реализационная гипотеза о безопасности и отношение безопасной конформности. В теоретическом плане рассматривается генерация тестов по спецификации в заданной семантике взаимодействия исследуемого типа.

Формализация тестового эксперимента. Семантика взаимодействия и безопасное тестирование

Под «правильностью» исследуемой системы понимается ее соответствие заданным требованиям. В модельном мире система отображается в реализационную модель (реализацию), требования – в спецификационную модель (спецификацию), а их соответствие – в бинарное отношение конформности. Спецификация

всегда задана, а существование реализационной модели предполагается (тестовая гипотеза). Если реализация также задана явно, верификация конформности может быть выполнена аналитически. Для реализации, устройство которой неизвестно («черный ящик») или слишком сложно для анализа, применяется тестирование как проверка конформности в процессе тестовых экспериментов. Разумеется, в этом случае требования должны быть функциональными, то есть выражены в терминах взаимодействия системы с окружающим миром, который при тестировании подменяется тестом. Поэтому само отношение конформности и его тестирование основаны на той или иной семантике взаимодействия.

Такая семантика формализует тестовые возможности по управлению и наблюдению за поведением тестируемой системы. При тестировании наблюдается только такое поведение реализации, которое, во-первых, «спровоцировано» тестом (управление) и, во-вторых, наблюдаемо во внешнем взаимодействии. Такое взаимодействие моделируется с помощью так называемой машины тестирования [1 – 5]. Она представляет собой «чёрный ящик», внутри которого находится реализация. Управление сводится к тому, что оператор машины, выполняя тест (понимаемый как инструкция оператору), осуществляет тестовое воздействие, нажимая кнопки на клавиатуре машины, тем самым «разрешая» реализации выполнять те или иные действия, которые могут им наблюдаться. Наблюдения (на «дисплее» машины) бывают двух типов: наблюдение *внешнего* (*наблюдаемого*) *действия*, разрешённого оператором и выполняемого реализацией, и наблюдение *отказа* как отсутствия каких бы то ни было наблюдаемых действий из числа разрешенных нажатыми кнопками. Подчеркнём, что оператор разрешает реализации выполнять именно множество действий, а не обязательно одно действие. Мы предлагаем считать, что оператор может нажимать только одну кнопку, но каждой кнопке соответствует множество разрешаемых действий. После наблюдения (действия или отказа) кнопка отжимается и все внешние действия запрещаются. Далее оператор может нажать другую (или ту же самую) кнопку.

Тестовые возможности определяются тем, какие «кнопочные» множества есть на клавиатуре машины и для каких кнопок возможно наблюдение отказа. Тем самым, семантика взаимодействия задается алфавитом внешних действий $\boldsymbol{L}$ и двумя наборами кнопок машины тестирования: с наблюдением соответствующих отказов – семейство $\boldsymbol{R}\subseteq 2^{\boldsymbol{L}}$ и без наблюдения отказа – семейство $\boldsymbol{Q}\subseteq 2^{\boldsymbol{L}}$. Предполагается, что $\boldsymbol{R}\cap\boldsymbol{Q}=\varnothing$ и $\cup\boldsymbol{R}\cup\cup\boldsymbol{Q}=\boldsymbol{L}$. Такую семантику мы называем $\boldsymbol{R}/\boldsymbol{Q}$-семантикой. Если $\boldsymbol{Q}=\varnothing$ и $\boldsymbol{R}=2^{\boldsymbol{L}}$, то это хорошо известная *failure trace semantics* [3, 4, 6 – 8]. Еще один пример – это семантика популярного отношения ***ioco*** [8, 9], когда действия разбиваются на стимулы (input) и реакции (output) $\boldsymbol{L}=\boldsymbol{I}\cup\boldsymbol{O}$. Есть только одна $\boldsymbol{R}$-кнопка приема всех реакций $\boldsymbol{R}=\{\delta\}$, где $\delta=\boldsymbol{O}$, а соответствующий отказ δ называется *стационарностью* (*quiescence*). Каждый стимул x посылается в реализацию с помощью $\boldsymbol{Q}$-кнопки $\{x\}$, $\boldsymbol{Q}=\{\{x\} \mid x\in\boldsymbol{I}\}$.

Кроме внешних действий в реализации могут быть внутренние действия, обозначаемые символом τ. Эти действия ненаблюдаемы и всегда разрешены (при нажатии любой кнопки и при отсутствии нажатой кнопки).

Для выполнимости любого действия (как внешнего, так и внутреннего) необходимо, чтобы оно было определено в реализации и разрешено оператором. Если этого условия также и достаточно, то есть выполняться может любое действие, удовлетворяющее этому условию, то говорят, что в системе нет приоритетов. Тестирование систем с приоритетами исследуется в [10] и в данной статье не рассматривается.

«Практические предположения»: 1) Любая конечная последовательность любых действий совершается за конечное время, а бесконечная – за бесконечное. 2) «Передача» тестового воздействия (от кнопки) в реализацию и наблюдения от реализации (на дисплей машины) выполняются за конечное время. Эти предположения гарантируют возможность наблюдения внешнего действия, выполняемого реализацией, через конечное время после нажатия кнопки, разрешающей это действие. Это предположение часто используется для реализации наблюдения ***R***-отказа, но в усиленном варианте: время ожидания действия после нажатия кнопки не только конечно, но и ограничено. Вводится тайм-аут, истечение которого без наблюдения действия трактуется как отказ. Следует отметить, что это не единственный возможный способ реализации наблюдения отказа.

После нажатия ***R***-кнопки через конечное время оператор наблюдает или разрешенное этой кнопкой внешнее действие, или отказ. Однако при нажатии ***Q***-кнопки, если в реализации возможен отказ, то, поскольку этот отказ не наблюдаем, оператор не знает, нужно ли ему ждать наблюдения внешнего действия или такого действия не будет, поскольку возник отказ. Поэтому оператор не может ни продолжать тестирование, ни закончить его.

Бесконечная последовательность τ-действий («зацикливание») называется *дивергенцией* и обозначается символом Δ. Дивергенция сама по себе не опасна, но при попытке выхода из неё, когда оператор нажимает кнопку, он не знает, нужно ли ждать наблюдения или бесконечно долго будут выполняться только внутренние действия. Поэтому и в этом случае оператор не может ни продолжать тестирование, ни закончить его.

Кроме этого мы вводим новое, также не регулируемое кнопками, действие, которое называем *разрушением* и обозначаем символом γ. Оно моделирует любое нежелательное поведение системы, в том числе и ее реальное разрушение, которого нельзя допускать при взаимодействии. Разрушение – один из способов интерпретации неспецифицированного поведения. Полная спецификация внешнего поведения системы требуется тогда, когда окружение может вести себя произвольным образом. Такое требование к реализации естественно, если это «система общего пользования»: в ней должна быть предусмотрена «защита от дурака». Однако при взаимодействии между внутренними компонентами или подсистемами, доступ к которым строго ограничен, взаимные проверки корректности обращений излишни. Особенно при обращении с параметрами сложной структуры и нетривиальными условиями корректности, когда накладные расходы на проверку неоправданно увеличивают трудозатраты на создание системы, её объём и время выполнения. Альтернативой является строгая спецификация предусловий вызова операций [11]. Проверке подлежит не поведение компонентов в ответ на некорректные обращения от других компонентов, а правильность обращения компонентов друг к другу. Иными словами, поскольку мы хотим убедиться в правильности реализации, а не окружения, нас не интересует поведение реализации при неправильном взаимодействии с ней. Разрушение в спецификации отмечает те ситуации, когда в реализации допускается любое поведение, в том числе и реальное разрушение. Семантика разрушения предполагает, что оно не должно возникать при правильном поведении окружения.

Тестирование, при котором не возникает разрушения, попыток выхода из дивергенции и ненаблюдаемых отказов, называется *безопасным*.

Модели реализации и спецификации

Для взаимодействия, основанного на наблюдениях, результат тестового эксперимента – это чередующаяся последовательность кнопок (тестовых воздействий) и наблюдений, которую мы называем (*тестовой*) *историей*. В силу семантики дивергенции и разрушения достаточно рассматривать только такие истории, в которых символы Δ и γ могут быть только последними символами. Поскольку дивергенция и разрушение всегда разрешены, им не предшествует в истории никакая кнопка. Любое другое наблюдение u (внешнее действие или $\boldsymbol{R}$-отказ) разрешается непосредственно предшествующей ему кнопкой P, то есть $u \in P$ или $u = P$ для $P \in \boldsymbol{R}$. Подпоследовательность истории, состоящая только из наблюдений (включая Δ и γ), называется трассой. Если приоритетов нет, возможность данного наблюдения после трассы определяется тем, что нажимаемая кнопка разрешает это наблюдение, и не зависит от того, какие еще наблюдения она разрешает. Поэтому множество историй системы однозначно восстанавливается по множеству ее трасс, называется *трассовой моделью* системы и является наиболее естественной ее моделью.

Если в трассах модели в качестве отказов могут встречаться любые подмножества алфавита внешних действий $\boldsymbol{L}$, такие трассы и такую трассовую модель мы называем *полными*. Соответствующая $2^{\boldsymbol{L}}/\varnothing$-семантика известна в литературе под названием *failure trace semantics*, а полные трассы без дивергенции и разрушения называются *failure traces*. Трасса, в которой все отказы наблюдаемы для заданной $\boldsymbol{R}/\boldsymbol{Q}$-семантики, то есть принадлежат $\boldsymbol{R}$, называется $\boldsymbol{R}$-*трассой*, а подмножество $\boldsymbol{R}$-трасс полной трассовой модели – $\boldsymbol{R}$-*моделью*. При безопасном тестировании в $\boldsymbol{R}/\boldsymbol{Q}$-семантике наблюдаться могут только $\boldsymbol{R}$-трассы, не содержащие символов Δ и γ. Для ***ioco***-семантики $\boldsymbol{R}$-трассы (без Δ и γ) называются *suspension traces*. Другие трассы полной трассовой модели нужны для указания того, какие $\boldsymbol{R}$-трассы безопасны (могут наблюдаться при безопасном тестировании), а какие нет. Для этого достаточно учитывать только трассы, в которых отказы принадлежат множеству $\boldsymbol{R} \cup \boldsymbol{Q}$.

Семантика взаимодействия запрещает некоторые трассы. Например, трасса должна быть *согласованной*: после отказа P не может быть действия $z \in P$, а также символов Δ и γ. Поэтому модель – это множество трасс, которое удовлетворяет некоторому необходимому и достаточному набору условий. Формальное определение этих условий и доказательство их взаимной независимости, необходимости и достаточности даны в [2].

Другой эквивалентной моделью реализации и спецификации является LTS (Labelled Transition System), которая определяется как ориентированный граф, вершины которого называются состояниями, а дуги помечены внешними действиями или символами τ или γ и называются переходами. Переход из (пре)состояния s в (пост)состояние $s`$ по символу $z$ обозначается $s — z \rightarrow s`$. Выделяется начальное состояние, с которого реализация начинает работать при каждом рестарте. После рестарта реализация выполняет последовательность смежных переходов по маршруту, начинающемуся в начальном состоянии, каждый переход которого помечен действием z, разрешаемым текущей нажатой кнопкой P машины тестирования, то есть $z \in P \cup \{\tau, \gamma\}$.

Отказ P в LTS порождается в *стабильном* состоянии, из которого не выходят τ- и γ-переходы, при условии, что из этого состояния не выходят также переходы по действиям $z \in P$. Состояние *дивергентно*, если в нем начинается бесконечный τ-маршрут, то есть маршрут, все переходы которого помечены символом τ.

Для определения трасс LTS в стабильных состояниях добавляются виртуальные петли по порождаемым отказам, а в дивергентных состояниях – переходы по Δ. После этого трасса LTS определяется как последовательность пометок на переходах маршрута, начинающегося в начальном состоянии и не продолжающегося после Δ- или γ-перехода, с пропуском символа τ. LTS – наиболее наглядная модель, поскольку для любой LTS множество ее трасс является трассовой моделью. В то же время LTS обладает существенным неудобством, связанным с недетерминизмом. Недетерминизм в LTS проявляется как наличие τ-переходов и/или «веера» переходов $s—z\to s`$ из одного пресостояния $s$ по одному внешнему действию $z$, ведущих в разные постсостояния $s`$. Из-за этого трасса заканчивается, вообще говоря, не в одном состоянии, а во множестве состояний LTS. Если в семантике есть наблюдаемые отказы, то этот недетерминизм, вообще говоря, неустраним, поскольку трасса может продолжаться как отказом, так и действием из этого отказа, а это возможно только в разных состояниях, тем самым трасса не может заканчиваться в одном состоянии.

Авторы статьи предложили еще одну эквивалентную модель, в которой этого неудобства нет. Она называется RTS (Refusal Transition System) и представляет собой детерминированную LTS не в алфавите $\boldsymbol{L}\cup\{\tau,\gamma\}$, а в алфавите $\boldsymbol{L}\cup\boldsymbol{R}\cup\{\tau,\Delta,\gamma\}$, то есть в ней могут быть явные переходы по $\boldsymbol{R}$-отказам, а бесконечная цепочка τ-переходов $s—\tau\to\ldots$ заменена одним переходом $s—\Delta\to$. Правда, за это приходится расплачиваться наглядностью: не любая детерминированная LTS в таком алфавите является RTS, а лишь та, что удовлетворяет набору условий, определяемых семантикой взаимодействия. При преобразовании LTS в RTS состояниями RTS становятся множества состояний LTS в конце трасс, что аналогично известному алгоритму «детерминизации» порождающего автомата (или графа). Множество трасс LTS или RTS $\boldsymbol{S}$ будем обозначать $\boldsymbol{T}(\boldsymbol{S})$.

Гипотеза о безопасности и безопасная конформность

Как возможно безопасное тестирование, если реализация неизвестна? Например, если в начальном состоянии реализации определено разрушение, то такую реализацию не только нельзя тестировать, но даже запускать на выполнение. Выход в том, чтобы ограничиться реализациями, конформность которых заданной спецификации можно проверить при безопасном тестировании. Это ограничение формулируется как гипотеза о безопасности, и конформность определяется для класса реализаций, удовлетворяющих этой гипотезе. В силу эквивалентности трассовой, LTS- и RTS-моделей достаточно определить гипотезу о безопасности и конформность для трассовых моделей. Такие гипотезу и конформность можно назвать *трассовыми*, поскольку они зависят только от трасс реализации и спецификации, но не от их состояний и соответствия состояний.

Определим формально отношение безопасности «кнопка P безопасна в модели после трассы σ». При безопасном тестировании будут нажиматься только безопасные кнопки. Это отношение различно для реализации и спецификации. В полной трассовой реализации $\boldsymbol{I}$ отношение безопасности (***safe in***) означает, что 1) кнопка P после трассы σ *неразрушающая*: ее нажатие не может означать попытку выхода из дивергенции (трасса не продолжается дивергенцией) и не может вызывать разрушение (после действия, разрешаемого кнопкой), и 2) нажатие кнопки не может привести к ненаблюдаемому отказу (для $\boldsymbol{Q}$-кнопок): $\forall P\in\boldsymbol{R}\cup\boldsymbol{Q}\ \forall\sigma\in\boldsymbol{I}$

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

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

В полной трассовой спецификации $\boldsymbol{S}$ отношение безопасности (***safe by***) отличается лишь для $\boldsymbol{Q}$-кнопок: после трассы $\boldsymbol{Q}$-отказ Q может быть, если после трассы есть хотя бы одно действие $z \in Q$. Также, если действие разрешается хотя бы одной неразрушающей кнопкой, то оно должно разрешаться какой-нибудь безопасной кнопкой. В частности, если все неразрушающие кнопки, разрешающие действие, являются $\boldsymbol{Q}$-кнопками, то хотя бы одну из них нужно объявить безопасной. Такое отношение безопасности всегда существует: достаточно объявить безопасной каждую неразрушающую кнопку, разрешающую действие, продолжающее трассу. Но в целом указанные требования неоднозначно определяют отношение ***safe by***, и при задании спецификации $\boldsymbol{S}$ указывается конкретное отношение. Требования к ***safe by*** записываются так: $\forall R \in \boldsymbol{R}\ \forall z \in \boldsymbol{L}\ \forall Q \in \boldsymbol{Q}\ \forall \sigma \in \boldsymbol{S}$

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

$\exists P \in \boldsymbol{R} \cup \boldsymbol{Q}\ P$ ***safe***$_{\gamma\Delta}$ ***S after*** σ & $z \in P$ & $\sigma \cdot \langle z \rangle \in \boldsymbol{S} \Rightarrow \exists P` \in \boldsymbol{R} \cup \boldsymbol{Q}\ z \in P`$ &

$P`$ ***safe by S after*** σ,

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

Безопасность кнопок определяет безопасность наблюдений. $\boldsymbol{R}$-отказ R безопасен, если после трассы безопасна кнопка R. Действие z безопасно, если оно разрешается некоторой кнопкой, безопасной после трассы:

z ***safe in I after*** $\sigma =_{\mathbf{def}} \exists P \in \boldsymbol{R} \cup \boldsymbol{Q}\ z \in P$ & P ***safe in I after*** σ.

z ***safe by S after*** $\sigma =_{\mathbf{def}} \exists P \in \boldsymbol{R} \cup \boldsymbol{Q}\ z \in P$ & P ***safe by S after*** σ.

Теперь можно определить *безопасные* $\boldsymbol{R}$*-трассы*. $\boldsymbol{R}$-трасса безопасна, если эта трасса есть в модели и 1) модель не разрушается с самого начала (до нажатия первой кнопки), то есть, в ней нет трассы $\langle\gamma\rangle$, 2) каждый символ трассы безопасен после непосредственно предшествующего ему префикса трассы:

$\langle\gamma\rangle \notin \boldsymbol{I}$ & $\forall \mu\ \forall u$ ($\mu \cdot \langle u \rangle$ префикс $\sigma \Rightarrow u$ ***safe in I after*** μ),

$\langle\gamma\rangle \notin \boldsymbol{S}$ & $\forall \mu\ \forall u$ ($\mu \cdot \langle u \rangle$ префикс $\sigma \Rightarrow u$ ***safe by S after*** μ).

Множества безопасных трасс реализации $\boldsymbol{I}$ и спецификации $\boldsymbol{S}$ обозначим ***SafeIn***($\boldsymbol{I}$) и ***SafeBy***($\boldsymbol{S}$).

Заметим, что пустая $\boldsymbol{Q}$-кнопка всегда опасна как по отношению ***safe in***, так и по любому отношению ***safe by***. Такую кнопку нельзя нажимать при безопасном тестировании. Поэтому далее будем считать, что в семантике пустой $\boldsymbol{Q}$-кнопки нет: $\varnothing \notin \boldsymbol{Q}$. В то же время пустая $\boldsymbol{R}$-кнопка имеет смысл: она безопасна, если после трассы нет дивергенции, а наблюдение пустого $\boldsymbol{R}$-отказа означает, что реализация оказалась в стабильном состоянии.

Из определения отношений ***safe by*** и ***safe in*** видно, что множество безопасных трасс спецификации однозначно определяется множеством ее $\boldsymbol{R}$-трасс, а множество безопасных трасс реализации, кроме множества ее $\boldsymbol{R}$-трасс, дополнительно зависит от продолжения $\boldsymbol{R}$-трасс $\boldsymbol{Q}$-отказами.

Требование безопасности тестирования выделяет класс *безопасно-тестируемых* реализаций ***SafeImp***($\boldsymbol{R}/\boldsymbol{Q}$,$\boldsymbol{S}$,***safe by***), которые могут быть безопасно протестированы для проверки их конформности заданной спецификации $\boldsymbol{S}$ с заданным отношением ***safe by*** в заданной $\boldsymbol{R}/\boldsymbol{Q}$-семантике. Этот класс определяется *гипотезой о безопасности*: реализация $\boldsymbol{I}$ *безопасно-тестируема* для спецификации $\boldsymbol{S}$, 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:

$\boldsymbol{I}$ ***safe for*** $\boldsymbol{S} =_{\mathbf{def}} (\langle\gamma\rangle \notin \boldsymbol{S} \Rightarrow \langle\gamma\rangle \notin \boldsymbol{I})$ & $\forall \sigma \in \boldsymbol{SafeBy}(\boldsymbol{S}) \cap \boldsymbol{I}\ \forall P \in \boldsymbol{R} \cup \boldsymbol{Q}$

(P ***safe by S after*** $\sigma \Rightarrow P$ ***safe in I after*** σ).

Заметим, что для ***ioco***-семантики мы разрешаем в безопасно-тестируемых реализациях «безопасные» блокировки стимулов после трасс, которые в спецификации этими стимулами не продолжаются. Это более либерально, чем требование всюду определенности реализации по стимулам, предлагаемое автором отношения ***ioco*** Яном Тритмансом [8,9]. Этим мы устраняем «несогласованность» отношения ***ioco***, когда конформная всюду определенная по стимулам реализация A и реализация B, отличающаяся от A только наличием «безопасных» блокировок, неразличимы при ***ioco***-тестировании, однако A конформна, а B заведомо не конформна, так как не всюду определена по стимулам и тем самым не входит в домен отношения ***ioco***.

Теперь можно определить отношение (безопасной) *конформности*: реализация ***I*** (*безопасно*) *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:

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

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

Гипотеза о безопасности не проверяема при тестировании, являясь его предусловием; проверяется лишь тестируемое условие. Отношение ***saco*** определяет класс конформных реализаций $\boldsymbol{ConfImp}(\boldsymbol{R}/\boldsymbol{Q},\boldsymbol{S},\boldsymbol{safe\ by})$.

Генерация тестов

В терминах машины тестирования тест – это инструкция оператору машины. В каждом пункте инструкции указана кнопка, которую оператор должен нажимать, и для каждого наблюдения – пункт инструкции, который должен выполняться следующим, или вердикт (***pass*** или ***fail***), если тестирование нужно закончить. В тесте после кнопки P допускается только такое наблюдение u, которое разрешается кнопкой P, то есть $u\in P \vee u=P\in\boldsymbol{R}$.

Тест задается префикс-замкнутым множеством конечных историй, в котором 1) каждая максимальная история заканчивается наблюдением, и ей приписан вердикт; 2) после каждой немаксимальной истории, заканчивающейся кнопкой, следуют только те наблюдения, которые разрешаются этой кнопкой, и обязательно следуют те из них, что встречаются в безопасно-тестируемых реализациях после подтрассы этой истории.

Тест безопасен тогда и только тогда, когда в каждой его истории каждая кнопка безопасна в спецификации после подтрассы непосредственно предшествующего этой кнопке префикса истории. Иными словами, тест безопасен, если подтрассы его историй являются тестовыми, где *тестовая трасса* – это безопасная трасса или трасса $\sigma\cdot\langle u\rangle$, где трасса σ безопасна в спецификации, а наблюдение u безопасно после σ в спецификации (но не обязательно имеется в ней). Достаточно ограничиться только *актуальными* тестовыми трассами, под которыми понимаются трассы, встречающиеся в безопасно-тестируемых реализациях. В частности, трасса должна быть согласованной, хотя в спецификации могут быть согласованные, но неактуальные тестовые трассы.

Реализация *проходит* тест, если её тестирование этим тестом всегда заканчивается с вердиктом ***pass***. 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. Для определения конформности любой безопасно-тестируемой реализации ставится задача генерации полного набора тестов по спецификации.

Полный набор тестов всегда существует, например, им является набор всех *примитивных* тестов [2]. Такой тест строится по немаксимальной (по отношению «является префиксом») безопасной ***R***-трассе спецификации. Для этого в трассу вставляются кнопки: перед отказом R – кнопка R, перед действием z – любая кнопка P, разрешающая действие z и безопасная после соответствующего префикса трассы, а после всей трассы – любая безопасная после нее кнопка $P`$. Безопасность трассы гарантирует безопасность кнопки $R$ и наличие безопасной кнопки $P$, а немаксимальность трассы (среди безопасных) гарантирует наличие последней кнопки $P`$. Выбор кнопок, вообще говоря, неоднозначен: по одной безопасной трассе спецификации генерируется множество примитивных тестов. Однако множества тестов, сгенерированных по разным трассам, не пересекаются.

Если наблюдение, полученное после нажатия кнопки, продолжает трассу, тест продолжается (немаксимальная в тесте история). Наблюдение, полученное после нажатия последней кнопки, и любое наблюдение, «ответвляющееся» от трассы, всегда заканчивают тестирование (максимальная в тесте история). Вердикт ***pass*** выносится, если полученная ***R***-трасса (подтрасса максимальной истории) есть в спецификации, а вердикт ***fail*** – если нет. Такие вердикты соответствуют *строгим* тестам, которые, во-первых, значимые (не фиксируют ложных ошибок) и, во-вторых, не пропускают обнаруженных ошибок.

Любой строгий тест как множество историй равен объединению некоторого множества примитивных тестов, то есть они обнаруживают те же самые ошибки. Поэтому можно ограничиться только примитивными тестами. Но и среди примитивных тестов могут быть заведомо «лишние». Например, достаточно генерировать тесты только по трассам, в которых нет повторных (не разделенных действиями) отказов: если наблюдается отказ P, то повторное нажатие кнопки P гарантированно дает отказ. Кроме того, в спецификации могут быть безопасные *неконформные* трассы, то есть трассы, которые не могут встречаться в конформных реализациях.

Как сказано выше, в любой момент времени реализация выполняет любое внешнее действие, которое определено в ней и разрешено оператором, или любое внутреннее действие. Если таких действий несколько, выбирается одно из них недетерминированным образом. Предполагается, что недетерминизм реализации – это явление уровня абстракции, которое определяется тестовыми возможностями по наблюдению и управлению, то есть семантикой взаимодействия. Иными словами, поведение реализации недетерминировано, поскольку оно зависит от неких не учитываемых факторов – «погодных условий», которые определяют выбор выполняемого действия детерминировано. Для того чтобы тестирование было полным, любые погодные условия должны быть воспроизводимы в тестовом эксперименте, причём для каждого теста. Если это так, то есть при бесконечной последовательности прогонов теста проявятся все возможные погодные условия, тестирование называется *глобальным* [5]. Мы абстрагируемся от количества вариантов погодных условий, здесь важна лишь потенциальная возможность проверить поведение системы при любых погодных условиях и любом поведении оператора. Без этого мы не можем быть уверены, что провели тестовые испытания каждого

теста для всех возможных погодных условий, то есть при любом возможном недетерминированном поведении реализации.

Если гипотеза о глобальном тестировании верна, то для того, чтобы тесты можно было прогонять при всех возможных погодных условий, набор тестов должен быть перечислим. Тогда тесты прогоняются по мере их перечисления так, чтобы каждый перечисленный тест прогонялся неограниченное число раз в «диагональном» процессе перечисления тестов и последовательности прогонов каждого теста. Поскольку все тесты конечны, каждый из них заканчивается через конечное время, после чего выполняется рестарт системы, и прогоняется следующий тест или повторно один из уже перечисленных тестов. Если реализация неконформна, то полнота набора тестов гарантирует обнаружение ошибки через конечное время. Однако конформность реализации не может быть обнаружена за конечное время, если набор тестов бесконечен или глобальное тестирование требует не конечного, а бесконечного числа прогонов тестов. Но это уже проблема практического тестирования.

Отметим, что последовательность прогонов тестов, чередующихся с рестартом системы, можно рассматривать как один тест, в котором, кроме тестовых воздействий, встречается рестарт системы. Вместо конечности каждого теста в наборе тестов мы должны потребовать конечность каждого отрезка единого теста между рестартами, то есть отсутствие в нем бесконечного постфикса без рестарта. Предполагается, что рестарт системы всегда выполняется правильно (система действительно «сбрасывается» в начальное состояние), и его не нужно тестировать. Тем самым, различие между перечислимым набором тестов и одним тестом с рестартами условное и определяется удобством организации тестирующей системы.

Заключение

Предлагаемые в статье параметризованная ***R***/***Q***-семантика взаимодействия, гипотеза о безопасности, безопасная конформность и метод генерации тестов составляют основу теории конформности [2], которая обобщает многие встречающиеся в литературе и используемые на практике конформности и методы их тестирования. В частности, *failure trace semantics* [3, 4, 6 – 8, 12] и семантика популярного отношения ***ioco*** [8,9] являются частными случаями ***R***/***Q***-семантики. В настоящее время эта теория развита в нескольких направлениях. 1) Пополнение спецификации, решающее проблему нерефлексивности спецификации по отношению конформности, а также проблему безопасных неконформных и тестовых неактуальных трасс спецификации [2]. 2) Монотонное преобразование спецификации, решающее проблему верификации имеющейся спецификации системы (её согласованности со спецификациями компонентов), и проблему генерации спецификации системы по спецификациям компонентов [2]. 3) Введение в модель приоритетов и соответствующая модификация конформности и генерации тестов [10]. 4) Распространение предлагаемого подхода на симуляции – конформности, основанные не только на трассах наблюдений, но и на соответствии состояний реализации и спецификации [13]. Также исследованы вопросы эквивалентности семантик взаимодействия [14] и преобразования семантик при тестировании с использованием программ-медиаторов, осуществляющих преобразование тестовых воздействий и наблюдений [15].

Другого рода исследования связаны с применением этой теории на практике. Основная задача, решаемая здесь, – это поиск практически приемлемых ограни-

чений на семантику, реализацию и/или спецификацию, а также дополнительных тестовых возможностей, которые позволяли бы проводить полное тестирование за конечное время. Сюда относятся ограничения на размер реализации, возможность наблюдения текущего состояния реализации в процессе тестирования и ограничения на недетерминизм реализации [13, 15, 16].

ЛИТЕРАТУРА

1. *Бурдонов И.Б.*, *Косачев А.С.*, *Кулямин В.В.* Формализация тестового эксперимента // Программирование. 2007. № 5. С. 3–32.
2. *Бурдонов И.Б.* Теория конформности для функционального тестирования программных систем на основе формальных моделей: дис. ... д.ф.-м.н. М., 2008. 596 с. URL: http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
3. *van Glabbeek R.J.* The linear time – branching time spectrum // CONCUR'90, LNCS 458 / J.C.M. Baeten and J.W. Klop, editors. Springer-Verlag, 1990. P. 278–297.
4. *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. P. 66–81.
5. *Milner R.* Modal characterization of observable machine behaviour // Proceedings CAAP 81, LNCS 112 / G. Astesiano & C. Bohm, editors. Springer. P. 25–34.
6. *Baeten J.C.M.* Procesalgebra. – Programmatuurkunde. Kluwer. Deventer. In Dutch. 1986.
7. *Phillips I.* Refusal testing // Theoretical Computer Science. 1987. No. 2. P. 241–284.
8. *Tretmans J.* Test generation with inputs, outputs and repetitive quiescence // Software-Concepts and Tools. 1996. V. 17. Issue 3.
9. *Tretmans J.* Conformance testing with labelled transition systems: implementation relations and test generation // Computer Networks and ISDN Systems. 1996. No. 1. P. 49–79.
10. *Бурдонов И.Б.*, *Косачев А.С.* Системы с приоритетами: конформность, тестирование, композиция // Программирование. 2009. № 4. С. 24–40.
11. *Hoare C.A.R.* An axiomatic basis for computer programming // Communications of the ACM. 1969. No. 10. P. 576–585.
12. *Langerak R.* A testing theory for LOTOS using deadlock detection // Protocol Specification, Testing, and Verification IX / E. Brinksma, G. Scollo, and C.A. Vissers, editors. North-Holland, 1990. P. 87–98.
13. *Бурдонов И.Б.*, *Косачев А.С.* Тестирование конформности на основе соответствия состояний // Труды Института системного программирования РАН. 2010. № 18.
14. *Бурдонов И.Б.*, *Косачев А.С.* Эквивалентные семантики взаимодействия // Там же. 2008. № 14.1. С. 55–72.
15. *Бурдонов И.Б.*, *Косачев А.С.* Тестирование с преобразованием семантик // Там же. 2009. № 17. С. 193–208.
16. *Бурдонов И.Б.*, *Косачев А.С.* Полное тестирование с открытым состоянием ограниченно недетерминированных систем // Программирование. 2009. № 6. С. 3–18.

Бурдонов Игорь Борисович
Косачев Александр Сергеевич
Институт системного программирования РАН (г. Москва).
E-mail: igor@ispras.ru; kos@ispras.ru

Поступила в редакцию 4 октября 2010 г.