И.Б.Бурдонов, А.С.Косачев.
Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования.
«Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №2, 2011, стр.89-98.
10 стр.

УДК 681.3.06

Семантики взаимодействия с отказами, дивергенцией и разрушением.
Часть 2. Условия конечного полного тестирования.

А.С. Косачев, И.Б. Бурдонов

Институт системного программирования РАН (ИСП РАН) , Москва, Россия

kos@ispras.ru, igor@ispras.ru

Исследуются методы практического тестирования конформности исследуемой системы спецификации. Изучаются практически приемлемые ограничения на семантику, реализацию и спецификацию, а также дополнительные тестовые возможности, которые позволяют 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.

Ключевые слова: Тестирование, конформность, трассы, отказы, состояния, недетерминизм, LTS.

Введение

Статья посвящена методам тестирования соответствия (конформности) исследуемой системы заданным требованиям (спецификации) на основе формальных моделей. Эти методы опираются на теорию конформности, которая излагается в предыдущей статье авторов «Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 1. Гипотеза о безопасности и безопасная конформность». Основные понятия и обозначения этой теории используются в данной работе без повторного определения. Основное внимание уделяется возможности применения этой теории на практике. Для этого изучаются практически приемлемые ограничения на семантику, реализацию и/или спецификацию, а также дополнительные тестовые возможности, которые позволяют 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.

1. Проблемы практического тестирования

Для практического применения конечными по времени должны быть как генерация тестов, так и тестирование по этим тестам. Из первого вытекает конечность полного набора тестов (или единого теста с рестартом). Для этого «почти» необходимы, хотя и недостаточны, конечность алфавита внешних действий $\boldsymbol{L}$ и конечность LTS-спецификации (числа ее переходов), что на практике вполне приемлемо. Конечность тестирования опирается на конечность полного набора тестов, конечность времени прогона каждого теста и конечность требуемого числа прогонов каждого теста.

Конечность времени прогона теста гарантируется для конечного теста «практическими предположениями» о семантике: 1) Любая конечная последовательность любых действий (как внешних, так и внутренних) совершается за конечное время, а бесконечная – за бесконечное время. 2) «Передача» тестового воздействия (нажатие кнопки) в реализацию и наблюдения от реализации выполняются за конечное время. Эти предположения гарантируют наблюдение внешнего действия, выполняемого реализацией, через конечное время после нажатия кнопки, разрешающей это действие.

Таким образом, остаются две основные проблемы: 1) конечность полного набора тестов и 2) конечность требуемого числа прогонов теста.

Эти проблемы не имеют решения в общем виде, поэтому такие решения приходится искать в частных случаях: либо ограничивая классы рассматриваемых спецификаций и/или реализаций, либо предполагая наличие дополнительных тестовых возможностей, либо сочетая одно с другим. Для первой проблемы одним из таких практически приемлемых ограничений является конечность реализации, что как правило оказывается необходимым, хотя и недостаточным для полного решения проблемы.

Отметим взаимную связь указанных двух проблем. Если число прогонов любых (или некоторых) тестов конечно, то это позволяет применять *адаптивное* тестирование, когда следующие тесты из набора выбираются в зависимости от результатов, полученных на предыдущих тестах. Например, если после всех прогонов теста обнаруживается, что после трассы σ нажатие кнопки P никогда (при любых «погодных условиях») не приводит

к наблюдению действия $z \in P$, то нам нет нужды прогонять тесты, построенные по трассам вида $\sigma \cdot \langle z \rangle \ldots$. Заметим, что такого рода *отрицательные наблюдения* [5] практически невозможны, если нет ограничений на число прогонов тестов, так как тогда такие наблюдения могут быть «вычислены» только после бесконечного числа прогонов тестов.

Также взаимосвязаны способы решения проблем: ограничения на классы рассматриваемых реализаций и спецификаций напрямую зависят от дополнительных тестовых возможностей (или их отсутствия). Одной из таких возможностей является опрос текущего состояния реализации. Если можно «подсматривать» состояния реализации, говорят о тестировании с открытым состоянием, в противном случае – о тестировании с закрытым состоянием. В следующих разделах мы рассмотрим эти два вида тестирования и соответствующие ограничения на классы реализаций и/или спецификаций.

Проблема конечности требуемого числа прогонов теста общая для этих двух видов тестирования. Гипотеза о глобальном тестировании дает только теоретическую возможность обнаружить любую ошибку в любой неконформной реализации. На практике нам, прежде всего, требуется конечность различимых «погодных условий», а также либо какие-то способы «управления погодой», либо гипотезы, ограничивающие возможные проявления недетерминизма реализации. Первое возможно в каких-то частных случаях, например, когда недетерминизм является следствием псевдопараллелизма, то есть псевдопараллельного выполнения нескольких детерминированных процессов на одном процессоре. Если мы можем вмешиваться в работу планировщика, мы тем самым можем «управлять погодой». Второй способ используется чаще всего в его экстремальном виде, когда ограничиваются только детерминированными реализациями (при недетерминированной спецификации).

Здесь мы должны уточнить понятие детерминизма. Обычно LTS-реализация определяется как детерминированная, если каждая ее трасса заканчивается ровно в одном состоянии. Однако в общем случае для тестирования в ***R/Q***-семантике дополнительно требуется, чтобы в каждом достижимом состоянии для каждой кнопки $P \in \boldsymbol{R} \cup \boldsymbol{Q}$ было определено не более одного перехода по действию $z \in P$. Это определение можно ослабить, если учитывать гипотезу о безопасности, определяемую спецификацией. В терминах тестов это можно сформулировать так: безопасно-тестируемая (удовлетворяющая гипотезе о безопасности) LTS-реализация детерминирована в ***R***/***Q***-семантике для заданной спецификации, если каждый тест при любом его прогоне либо заканчивается только с вердиктом ***fail*** в любом состоянии реализации, либо только с вердиктом ***pass*** в одном и том же состоянии реализации.

В данной статье мы исходим из следующей гипотезы о *t-недетерминизме*: если в любом состоянии *i* реализации любую кнопку *P* нажимать *t* раз, то реализация продемонстрирует все возможные варианты поведения, то есть будут получены все возможные пары (наблюдение,постсостояние). Эту гипотезу можно ослабить, если учитывать только те состояния *i* реализации, которые достижимы по безопасным трассам спецификации, и только те кнопки *P*, которые безопасны после таких трасс. Кроме того, если любое наблюдение после нажатия кнопки *P* в состоянии *i* ошибочно, то можно не налагать никаких ограничений на число таких наблюдений и постсостояний.

Остановимся на одном независимом аспекте проблемы конечности полного набора тестов: проблеме определения безопасных трасс в спецификации. Дело в том, что правила для отношения безопасности кнопок ***safe by*** позволяют после различных трасс, заканчивающихся в LTS-спецификации в одном множестве состояний (в детерминированной RTS-спецификации – в одном состоянии), по-разному определять безопасные кнопки. При наличии циклов мы получаем бесконечную цепочку конечных трасс с произвольным распределением безопасных кнопок после этих трасс. Отношение ***safe by***, при котором трассы, заканчивающиеся в одном множестве состояний LTS-спецификации (в одном состоянии RTS-спецификации), имеют одинаковые множества безопасных после них ***Q***-кнопок (и, тем самым, всех кнопок, поскольку безопасность ***R***-кнопок одинакова для всех таких трасс), будем называть *ограниченным*. Это дает нам возможность говорить о безопасности кнопки *P* (и соответствующих наблюдений) во множестве *S* состояний LTS-спецификации *P* ***safe*** *S* (в состоянии *s* RTS-спецификации *P* ***safe*** *s*). Для LTS-спецификации с конечным числом состояний конечно число множеств состояний (состояний соответствующей RTS-спецификации). Заметим, что при отсутствии **Q**-кнопок ***safe by*** = ***safe in*** и является ограниченным отношением безопасности.

В дальнейшем будем предполагать, что алфавит ***L*** (и, тем самым, ***R***/***Q***-семантика), спецификация и реализация конечны, и реализация *t*-недетерминированная. Обозначим:

b — число кнопок,

n – максимальное число состояний LTS-реализации,

m – максимальное число переходов LTS-реализации, $m \le btn$,

k – максимальное число состояний LTS-спецификации,

K – максимальное число состояний RTS-спецификации.

Заметим, что для LTS-спецификации с числом состояний *k* в соответствующей RTS-спецификации $K \le 2^k - 1$.

И.Б.Бурдонов, А.С.Косачев.
Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования.
«Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №2, 2011, стр.89-98.
10 стр.

---

2. Тестирование с закрытым состоянием

В этом разделе рассматриваются наборы примитивных тестов, когда по окончанию одного теста выполняется рестарт и запускается другой тест (или тот же самый тест для следующего прогона). Решение проблемы конечности полного набора тестов будем искать двумя способами: сужая классы рассматриваемых спецификаций или реализаций. Для конечной семантики по каждой трассе генерируется конечное число примитивных тестов (тестов, сгенерированных по одной трассе спецификации). Поэтому достаточно конечности полного набора трасс, что для конечной семантики эквивалентно ограниченности длины трасс полного набора.

Ограничение на спецификацию. Если число ***R***-кнопок конечно, а в конечной спецификации нет циклов, то число трасс спецификации конечно, тем более конечен полный набор трасс. В то же время не любой цикл приводит к бесконечности полного набора трасс.

Для RTS-спецификации будем называть *демоническим* состояние s, после которого не бывает ошибок, то есть любая актуальная (встречающаяся в безопасно-тестируемых реализациях) тестовая трасса $\mu\cdot\lambda$, где трасса μ заканчивается в s, не является ошибкой: она имеется в спецификации. Для этого должны быть выполнены следующие условия. 1) Определен переход $s\text{—}u\rightarrow s`$ по каждому наблюдению $u$, безопасному в $s$, за исключением действий, запрещенных петлями по отказам, то есть действий $z\in P$, где $P\in\boldsymbol{R}$ и $s\text{—}P\rightarrow s$, и конечное состояние $s`$ также демоническое. 2) Если это переход по действию $u\in\boldsymbol{L}$, то в состоянии $s`$ нет петли по отказу $P\neq\varnothing$. 3) Если это переход по отказу $u\in\boldsymbol{R}$, то в состоянии $s`$ есть петли только по отказам $P\subseteq u$ и по отказам, по которым есть петли в s. Эти условия не учитывают актуальности тестовых трасс, поэтому в общем случае они только достаточны для демоничности состояния s. Для того, чтобы эти условия стали также необходимыми, требуется пополнение спецификации [2] с последующим удалением неактуальных безопасных трасс.

Для того, чтобы для конечной спецификации в конечном алфавите существовал полный набор трасс ограниченной длины необходимо и достаточно, чтобы в RTS-спецификации любой безопасный маршрут (маршрут с безопасной трассой) не содержал цикла, проходящего через недемонические состояния. Заметим, что в безопасном маршруте множество состояний любого цикла не может быть смесью демонических и недемонических состояний. Длина трасс такого полного набора имеет точную верхнюю оценку K.

Ограничение на размер реализации. Путь задана RTS-спецификация ***S*** и рассмотрим некоторую произвольную безопасно-тестируемую LTS-реализацию ***I***. Пусть σ безопасная трасса спецификации, а P - кнопка, безопасная в спецификации после трассы σ. Обозначим через s_σ (единственное) состояние спецификации, в котором заканчивается трасса σ, а через i_σ – одно из состояний реализации ***I***, в которых заканчивается трасса σ. Если для каждой тройки (s_σ,i_σ,P) в наборе тестов будет примитивный тест, сгенерированный по какой-нибудь трассе $\sigma`$ такой, что $s_\sigma=s_{\sigma`}$, в котором после трассы $\sigma`$ нажимается кнопка P, то такой набор тестов, очевидно, будет полным для ограниченного отношения ***safe by***. При однократном прогоне теста на реализации ***I*** мы получаем последовательность пар состояний (s_μ,i_μ), где μ - префикс трассы σ. Будем говорить, что тест *простой* для данной реализации ***I***, если хотя бы при одном его прогоне в полученной последовательности пар состояний нет одинаковых пар. Очевидно, что для полноты тестирования данной реализации ***I*** достаточно тестов, которые простые для этой реализации. Для полноты тестирования всех реализаций достаточно, чтобы в набор тестов входили все тесты, которые просты для той или иной реализации.

Оценим сверху длину N простого теста. Если число состояний реализации ***I*** не превосходит n, то, очевидно, $N\leq nK$. Если набор тестов состоит из всех тестов длины не больше $nK\leq n(2^k-1)$, то такой набор тестов содержит все простые тесты для любой реализации, число состояний которой не больше n. Эта верхняя оценка является точной по порядку: для любых k и n существует семантика с $O(k)$ действиями, LTS-спецификация с $O(k)$ состояниями и неконформная реализация с $O(n)$ состояниями, ошибка в которой не может быть обнаружена тестом длины меньше $O(n2^k)$. Можно показать, что даже для семантики с ограниченным число действий оценка остается суперполиномиальной. В частности, существует семантика с двумя действиями, для которой в точной верхней оценке показатель степени k заменяется на $c\boldsymbol{ln}^2k$. LTS-спецификации, на которых достигаются эти оценки, существенно недетерминированы. Понятно, что для детерминированной LTS-спецификации $K=k$ (а не 2^k). Однако даже минимальный недетерминизм LTS-спецификации (нет τ-переходов и только в одном состоянии только по одному действию имеются два перехода в разные состояния) оставляет оценки суперполиномиальными.

Ограничение на недетерминизм реализации. При тестировании с закрытым состоянием нужна усиленная гипотеза о t-недетерминизме: нас интересует число $T(\sigma,P)$ возможных поведений реализации при нажатии кнопки P не в состоянии реализации (которого мы не видим), а после трассы σ. Если число состояний

реализации ограничено числом n, то $T(\sigma,P)\le n$. Но можно использовать и независимую от n гипотезу о том, что $T(\sigma,P)$ ограничено некоторым числом T. Такую гипотезу мы вынуждены использовать, если ограничения налагаются только на спецификацию, но не на размер реализации.

Оценим сверху число прогонов примитивного теста, который задается чередующейся последовательностью кнопок и наблюдений $P_1,u_1,P_2,u_2,\ldots,P_N$ при заданном числе T. Последнюю кнопку в тесте мы должны нажимать T раз. Чтобы T раз гарантированно получить предпоследнее наблюдение u_{N-1}, предыдущую кнопку P_{N-1} достаточно нажимать не более T^2 раз. Чтобы T^2 раз гарантированно получить наблюдение u_{N-2}, предыдущую кнопку P_{N-2} достаточно нажимать не более T^3 раз. И так далее. Первую кнопку P_1 достаточно нажимать не более T^N раз. Тем самым, число прогонов теста оценивается как $O(T^N)$. Эта оценка достижима для некоторых реализаций, но, конечно, для каких-то (классов) реализаций она может оказаться гораздо меньше. В частности, для детерминированных реализаций оценка $O(N)$.

3. Тестирование с открытым состоянием

В этом разделе рассматривается не набор тестов, а один адаптивный тест с возможными рестартами в середине теста. Будем предполагать, что спецификация ***S*** задана как RTS. Для полноты тестирования нужно проверить все переходы реализации, лежащие на маршрутах с трассами, безопасными в спецификации. Для этого LTS-реализация должна быть сильно-связной: из каждого состояния достижимо по переходам каждое другое состояние (достаточно ограничиться состояниями, достижимыми из начального состояния по безопасным трассам спецификации). Рестарт понимается как одно из внешних действий, отличающихся только тем, что гарантированно переводит реализацию в начальное состояние. Переход по рестарту делает трассу пустой. Переходы по рестарту дополнительно учитываются при определении сильно-связности. Также требуется, чтобы начальное состояние реализации было стабильным или хотя бы в одном состоянии, достижимом по безопасной трассе спецификации, был определен рестарт, безопасный в спецификации после этой трассы. Действительно, в примере на Рис.1 реализация в начале тестирования может перейти в состояние 1. После этого без рестарта она попадет в состояние 0 только после трассы $\langle x\rangle$, после которой в спецификации кнопка $\{y\}$ опасна: ее нельзя нажимать при тестировании. Тем самым ошибка в реализации не будет обнаружена.

$\boldsymbol{R}=\{\{x\},\{y\}\}$, ***S*** – спецификация, ***I*** – реализация

I saeθ S: $\{y\}$ ***safe S after*** ε & $y\in$***obs***$(\varepsilon,\{y\},\boldsymbol{I})$ & $y\notin$***obs***$((\varepsilon,\{y\},\boldsymbol{S})$

Рис.1 Рестарт

Получая наблюдения и опрашивая состояния реализации, мы будем поэтапно строить реализацию с одновременной проверкой тестируемого условия. Предлагаемый алгоритм является модификацией алгоритма в [3]. Основные идеи доказательств правильности алгоритма и оценок сложности примерно те же самые и для экономии места здесь опущены.

Предварительно строятся структуры для спецификации ***S***, которые не зависят от реализации и используются без модификации для верификации любой реализации в той же ***R***/***Q***-семантике. Рассматриваются состояния s в конце безопасных трасс, для каждого из которых определяем:

A$(s)=\{P\in\boldsymbol{R}\cup\boldsymbol{Q}\mid P$ ***safe*** $s\}$ — множество кнопок, безопасных в состоянии s;

B$(s)=(\cup\boldsymbol{A}(s))\cup(\boldsymbol{R}\cap\boldsymbol{A}(s))\cup\{\tau\}$ — множество, состоящее из безопасных наблюдений и символа τ;

C$(s,u)=s`$, где $s`$ состояние после перехода $s—u\to s`$ по наблюдению u из состояния s; доопределим ***C***$(s,\tau)=s$. Если таких переходов нет, ***C***$(s,u)=*$, где $*$ не совпадает ни с одним состоянием спецификации.

LTS-реализация ***I*** строится в процессе тестирования. Более точно: строится LTS-модель, имеющая такое же как в реализации множество трасс, безопасных в спецификации, и такое же множество состояний, достижимых по этим трассам. В начале тестирования и после каждого перехода опрашиваем состояние реализации. Переход $i—z\to i`$ по внешнему действию $z$ добавляется, когда после опроса состояния $i$ нажимается кнопка $P$, после чего наблюдается действие $z\in P$, а затем опрашивается постсостояние $i`$. Если наблюдается отказ с *тем же самым* постсостоянием $i`=i$, то добавляется виртуальный переход-петля по отказу $i—P\to i$. Если отказ P наблюдается с

другим постсостоянием $i` \neq i$, то добавляются переходы $i \text{—}\tau\to i` \text{—}P\to i`$. Вместе с каждым переходом по внешнему действию $i \text{—}z\to i`$ будем хранить кнопку $P(i \text{—}z\to i`)$, нажатие которой вызвало этот переход.

При построении $\boldsymbol{I}$ с каждым построенным состоянием i будем связывать множество $\boldsymbol{S}(i)$ состояний спецификации: $\boldsymbol{S}(i)=\cup\{\boldsymbol{S}\ \textbf{after}\ \sigma | \sigma\in \textbf{Safe}(\boldsymbol{T}(\boldsymbol{S})) \ \&\ i\in(\boldsymbol{I}\ \textbf{after}\ \sigma)\}$ (в детерминированной RTS-спецификации множество $\boldsymbol{S}\ \textbf{after}\ \sigma$ состояний после трассы σ состоит из одного состояния). Множество $\boldsymbol{S}(i)$ состоит из состояний спецификации в конце трасс, безопасных в спецификации, имеющихся в реализации и заканчивающихся там в состоянии i. В процессе тестирования мы будем строить эти множества $\boldsymbol{S}(i)$, постепенно добавляя в них состояния $s`$, где $\{s`\}=\boldsymbol{S}\ \textbf{after}\ \sigma$.

Кнопка P *допустима в* i, если она безопасна хотя бы в одном состоянии $s\in\boldsymbol{S}(i)$. Только допустимые кнопки будут нажиматься в состоянии i. Для каждой допустимой кнопки P определим счётчик $c(P,i)$ числа ее нажатий в состоянии i. Кнопка P *полна в состоянии* i, если 1) $c(P,i)=1$ и в $\boldsymbol{I}$ есть виртуальный переход-петля по отказу $i \text{—}P\to i$, или 2) $c(P,i)=t$. В обоих случаях уже получены все возможные переходы из состояния i при нажатии кнопки P. Состояние *полно*, если каждая допустимая в нем кнопка полна.

Кроме $\boldsymbol{S}(i)$ формируются следующие структуры данных для каждого состояния i:

$\boldsymbol{A}(i)=\cup\{\boldsymbol{A}(s) | s\in\boldsymbol{S}(i)\}$ — множество кнопок, допустимых в состоянии i;

$\boldsymbol{D}(i)=\{i \text{—}z\to i`\}$ — множество переходов из состояния i.

Если тестируемое условие выполнено, то должно быть: $\forall i \text{—}u\to i` \in \boldsymbol{D}(i)\ \forall s\in\boldsymbol{S}(i)\ u\in\boldsymbol{B}(s) \Rightarrow \boldsymbol{C}(s,u)\neq *$. Мы будем проверять это на каждом шаге тестирования.

В начале тестирования после опроса состояния в $\boldsymbol{I}$ есть только одно состояние $i\in(\boldsymbol{I}\ \textbf{after}\ \varepsilon)$, где ε пустая трасса, и для этого состояния $\boldsymbol{S}(i)=\{s_0\}$, $\boldsymbol{A}(i)=\boldsymbol{A}(s_0)$, $\boldsymbol{D}(i)=\varnothing$ и $c(P,i)=0$ для каждой допустимой кнопки P.

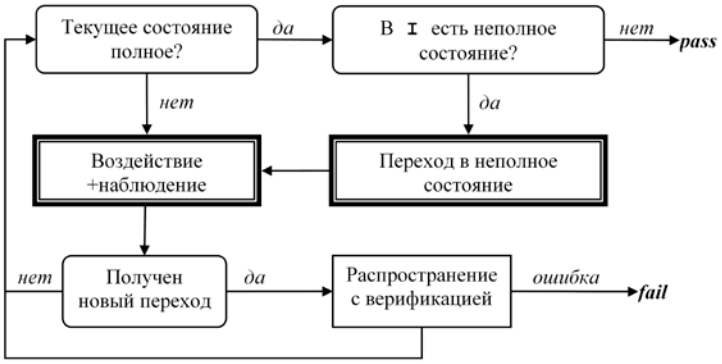

Рис.2 Общая схема алгоритма

На рис.2 изображена общая схема работы алгоритма. Если текущее состояние полное, то для продолжения тестирования нужно перейти в неполное состояние. Если таких состояний нет, алгоритм заканчивается с вердиктом ***pass***.

Рассмотрим переход в неполное состояние. В графе LTS $\boldsymbol{I}$ всегда существует множество деревьев, покрывающих все состояния так, что из каждого состояния выходит не более одного перехода, принадлежащего деревьям, и ориентированных к своим корням, которыми являются все неполные состояния. Выберем любое такое множество деревьев и будем двигаться по его переходам $i \text{—}u\to i`$, нажимая кнопки $P(i \text{—}u\to i`)$. Из-за недетерминизма мы можем оказаться не в $i`$, а в другом состоянии $i``$. Если это не листовое состояние дерева, то в нем определен переход дерева $i`` \text{—}u\to i```$, и будем нажимать кнопку $P(i`` \text{—}u\to i```)$. Доказательство достижения неполного состояния за конечное число шагов см. в [3].

Если текущее состояние i неполное, то выбираем неполную кнопку $P\in\boldsymbol{A}(i)$, нажимаем ее и получаем один переход $i \text{—}u\to i`$ или два перехода $i \text{—}\tau\to i` \text{—}P\to i`$. Постсостояние $i`$ становится новым текущим состоянием. Корректируем счетчик $c(P,i)$. Если получен новый переход, то корректируем $\boldsymbol{I}$ и $\boldsymbol{D}(i)$.

После этого выполняется блок «Распространие с верификацией». Для работы этого блока создается вспомогательный список $\boldsymbol{W}$ всех пар $(s, j \text{—}u\to j`)$, где $s\in\boldsymbol{S}(j)$ и $j \text{—}u\to j`\in\boldsymbol{D}(j)$. В самом начале множество $\boldsymbol{W}$ содержит все пары $(s, j \text{—}u\to j`)$ для каждого нового перехода $j \text{—}u\to j`$ и $s\in\boldsymbol{S}(j)$.

Опишем шаг работы блока. Если список $\boldsymbol{W}$ пуст, выходим из блока. Иначе, выбираем первый элемент $(s, j \text{—}u\to j`)$ из списка $\boldsymbol{W}$, удаляя его из списка. Проверяем тестируемое условие: если $j \text{—}u\to j`\in\boldsymbol{D}(j)\ \&\ u\in\boldsymbol{B}(s)\ \&\ \boldsymbol{C}(s,u)=*$, фиксируется ошибка и алгоритм заканчивается с вердиктом ***fail***. Если

И.Б.Бурдонов, А.С.Косачев.
Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2. Условия конечного полного тестирования.
«Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №2, 2011, стр.89-98.
10 стр.

ошибки нет и $\boldsymbol{C}(s,u)\in\boldsymbol{S}(j`)$, шаг заканчивается. В противном случае добавляем $\boldsymbol{C}(s,u)$ в $\boldsymbol{S}(j`)$ и помещаем в список $\boldsymbol{W}$ все пары $(\boldsymbol{C}(s,u), j`\text{—}u\to j``)$, где $j`\text{—}u\to j``\in\boldsymbol{D}(j`)$; шаг заканчивается.

Отметим, что при таком тестировании верифицируются не только наблюдения, полученные после *реальных* трасс, пройденных при тестировании, но и возможные наблюдения после *потенциальных* трасс, то есть наблюдения и трассы, про которые установлено, что они есть в реализации. Это даёт существенную экономию числа тестовых воздействий, необходимых для проверки конформности: мы выполняем множество проверок без реального тестирования, основываясь на полученном знании о поведении реализации. Например, если при тестировании получены две трассы $\mu_1\cdot\lambda$ и μ_2, где трассы μ_1 и μ_2 заканчиваются в реализации в одном состоянии i, то мы можем проверить обе трассы: как $\mu_1\cdot\lambda$, которую реально прошли при тестировании, так и потенциальную трассу $\mu_2\cdot\lambda$. Это преимущество дает дополнительная тестовая возможность опроса состояния реализации. По определению конформности мы должны проверять безопасные (точнее, тестовые) трассы спецификации, имеющиеся в реализации. При тестировании с закрытым состоянием, получив такую трассу, мы мало что знаем о том, какие еще трассы есть в реализации (гарантированно только те, что получаются из этой трассы удалением отказов). При тестировании с открытым состоянием мы получаем не просто трассу, а маршрут в реализации, то есть трассу с состояниями.

Доказательство полноты теста, работающего по описанному алгоритму, аналогично [3].

Приведем оценки сложности алгоритма (доказательство см. в [3]). При тестировании обычно наиболее важным считается число тестовых воздействий. Оценка $O_T=O(bnt^n)$ для $t>1$, и $O_T=O(bn^2)$для $t=1$. Последняя оценка достигается не только для детерминированной реализации, но и во всех случаях, когда переход в неполное состояние можно гарантированно выполнить, проходя *путь* (маршрут без самопересечений), длина которого ограничена n. Приведем три таких случая:

1. Упомянутый выше случай недетерминизма как следствия псевдопараллелизма, когда мы можем вмешиваться в работу планировщика процессов, тем самым «управляя погодой».

2. Сильно-Δ-связные LTS-реализации [1,3]. Это такие LTS, в которых для любой пары состояний i_1 и i_2 можно в каждом состоянии i найти такую кнопку $P(i)$, что, нажимая только такие кнопки, мы гарантированно окажемся в состоянии i_2, хотя путь из i_1 в i_2, который мы проходим, зависит от недетерминированного поведения LTS. Детерминированные LTS — это частный случай сильно-Δ-связных LTS.

3. Недетерминизм может быть следствием повышения уровня абстракции при моделировании детерминированной исследуемой системы такой реализацией, которая оказывается недетерминированной в той семантике взаимодействия, которая используется в спецификации. При тестировании связь уровней абстракции осуществляется промежуточной программой (медиатором). В этом случае при нажатии в состоянии i кнопки P медиатор добавляет дополнительный параметр p (возможно, зависящий от i), от которого абстрагируется модель. Если есть возможность вместе с наблюдением перехода $i\text{—}u\to i`$, где $u\in P$, получить от медиатора этот параметр $p$, то при повторном нажатии в состоянии $i$ кнопки $P$ можно сообщить медиатору параметр $p$, чтобы гарантированно выполнился тот же переход $i\text{—}u\to i`$.

Оценка объема вычислений содержит три слагаемых: 1) вычисления, необходимые для поиска опрошенного состояния среди пройденных при каждом тестовом воздействии, $nO_T=O(bn^2t^n)$ для $t>1$ или $nO_T=O(bn^3)$для $t=1$; 2) построение множества деревьев $O(b^2tn^2)$; 3) вычисления в блоке «Распространение с верификацией» $O(mK)$.

Заключение

В данной статье рассмотрены методы тестирования конформности при ограничениях на размер семантики, спецификации и реализации, а также ограничении недетерминизма реализации. При этих ограничениях эти методы позволяют проводить полное тестирование за конечное время. В рамках этих ограничений целью дальнейших исследований может быть задача минимизации тестового набора.

Пополнение спецификации [2] позволяет избавиться от неконформных и неактуальных безопасных трасс спецификации, тем самым уменьшая множество трасс, по которым нужно генерировать тесты. Существуют даже курьёзные примеры нетривиальных спецификаций, в которых бесконечное число безопасных трасс, но все они неконформны: без учета неконформности трасс полный набор тестов будет бесконечным, а с учетом таких трасс тестирование вообще излишне, так как аналитически доказывается отсутствие конформных реализаций.

Для минимизации набора тестов нужно исследовать зависимость между ошибками, когда из наличия в любой безопасно-тестируемой реализации одной ошибки следует наличие в ней другой ошибки. Понятно, что для полноты тестирования достаточно искать только те ошибки, которые минимальны по этому отношению следования (это отношение является предпорядком: рефлексивно и транзитивно).

Для полноты тестирования достаточно найти хотя бы одну ошибку в неконформной реализации. В то же время тестирование является лишь этапом в жизненном цикле разработки целевой системы [4]. За

тестированием обычно следует фаза исправления ошибок (в реализации, а иногда и в спецификации) и повторное тестирование. Поэтому для уменьшения числа итераций жизненного цикла можно поставить задачу исследования методов тестирования, которое должно обнаруживать как можно больше ошибок, а также ситуаций, где ошибок нет, чтобы предоставить разработчику как можно больше информации. Такое тестирование и соответствующий набор тестов можно называть *тотальными*. Если полное тестирование выполняется «до первой ошибки», то тотальное – пока не будут обнаружены все имеющиеся в реализации ошибки. Тотальное тестирование является полным, но обратное, вообще говоря, не верно. Для минимизации тотального набора тестов полезно искать только те ошибки, которые неэквивалентны по предпорядку следования ошибок.

Другие направления дальнейших исследований могут быть связаны с другими ограничениями на семантику, спецификацию и реализацию и/или с другими дополнительными тестовыми возможностями.

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. Диссертация на соискание учёной степени д.ф.-м.н., Москва, 2008. – 596 с. http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
3. *Бурдонов И.Б., Косачев А.С.* Полное тестирование с открытым состоянием ограниченно недетерминированных систем // Программирование. – 2009. – №6. – С. 3-18.
4. *Кулямин В.В.* Технологии программирования. Компонентный подход. – М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 463 с.
5. *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.