ТЕСТИРОВАНИЕ КОНФОРМНОСТИ С ОТКРЫТЫМ СОСТОЯНИЕМ

И.Б. Бурдонов, А.С. Косачев

1. Введение

Тестирование – это проверка в процессе эксперимента соответствия (конформности) реализации требованиям, заданным в виде спецификации. Для этого тест взаимодействует с реализацией, подменяя собой ее окружение. Тестирование полное, если оно однозначно отвечает на вопрос: есть в реализации ошибки или нет, где под ошибкой понимается нарушение требований, то есть неконформность. Для практического применения тестирование должно заканчиваться за конечное время. В общем случае тестирование оказывается либо неполным, либо бесконечным. Решение проблемы можно искать, сужая класс рассматриваемых реализаций и используя дополнительные тестовые возможности. В некоторых случаях удаётся построить тесты, которые конечны и полны, но только для заданных классов реализаций и спецификаций при дополнительных тестовых возможностях.

В статье рассматривается тестирование конечной реализации по конечной спецификации с двумя дополнительными предположениями: 1) тестирование с открытым состоянием – у нас есть возможность наблюдать состояния реализации, в которых она оказывается в процессе тестирования, 2) реализация ограниченно-недетерминирована – если одно и то же тестовое воздействие повторяется в одном и том же состоянии реализации достаточное, известное заранее число раз, то реализация демонстрирует все возможные варианты поведения. Для этого случая мы предлагаем алгоритмы конечного и полного тестирования и даём оценки числа тестовых воздействий и объёма вычислений.

2-ой раздел статьи содержит основные положения теории конформности, изложенной в [2-4]. В 3-ем разделе предлагается алгоритм тестирования, доказывается конечность и полнота тестирования и приводятся оценки сложности.

2. Теория конформности

2.1. Семантика взаимодействия и его безопасность

Отношение конформности основано на той или иной модели взаимодействия. Мы рассматриваем семантики взаимодействия, которые используют только внешнее, наблюдаемое поведение системы и не учитывают её внутреннее устройство, отражаемое на уровне LTS-модели в понятии состояния. Мы можем наблюдать только такое поведение реализации, которое, во-первых, «спровоцировано» окружением (управление) и, во-вторых, наблюдаемо во внешнем взаимодействии. Такая семантика взаимодействия может моделироваться с помощью так называемой машины тестирования [2-4,6,7]. Она представляет собой «чёрный ящик», внутри которого находится реализация. Управление сводится к тому, что оператор машины (моделируя работу окружения) последовательно нажимает кнопки на клавиатуре машины, «разрешая» реализации выполнять те или иные действия. Наблюдения (на «дисплее» машины) бывают двух типов: наблюдение выполняемого реализацией действия и наблюдение отказа как отсутствия выполняемых действий.

Подчеркнём, что при управлении оператор разрешает реализации выполнять именно множество действий, а не обязательно одно действие. Например, в реактивных системах получение реакций от реализации управляется одной кнопкой, разрешающей реализации послать любую реакцию.

Каждой кнопке соответствует своё множество разрешаемых действий. Действие может наблюдаться, если реализация может его выполнить и оно разрешено нажатой кнопкой. Отказ наблюдается, если реализация не может выполнить ни одно из действий, разрешаемых нажатой кнопкой.

Возможности взаимодействия определяются тем, какие «кнопочные» множества есть в машине, а также, для каких кнопок возможно наблюдение отказа. В данной работе для простоты изложения мы будем предполагать, что отказы наблюдаемы для всех кнопок. Тем самым, семантика взаимодействия определяется алфавитом внешних (наблюдаемых) действий **L** и набором **R** кнопочных множеств, причем $\cup\mathbf{R}=\mathbf{L}$. Такую семантику мы называем **R**-семантикой.

Кроме внешних действий реализация может совершать внутренние (ненаблюдаемые) действия, обозначаемые символом τ, которые всегда разрешены. Предполагается, что конечная последовательность действий совершается за конечное время, а бесконечная — за бесконечное время. 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 какую-нибудь кнопку, он не знает, нужно ли ждать наблюдения или бесконечно долго будут выполняться только внутренние действия. Поэтому нельзя ни продолжить взаимодействие, ни закончить его.

Кроме этого мы вводим специальное, 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 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. Взаимодействие, при котором не возникает разрушения и попыток выхода из дивергенции, называется безопасным.

2.2. LTS-модель и её трассы

LTS-модель – это совокупность $\mathbf{S}=LTS(V_s,\mathbf{L},E_s,s_0)$, где V_s – непустое множество состояний, $\mathbf{L}$ – алфавит внешних действий, $E_s\subseteq V_s\times(\mathbf{L}\cup\{\tau,\gamma\})\times V_s$ – множество переходов, $s_0\in V_s$ – начальное состояние. Переход из состояния s в состояние s' по действию z обозначается $s\text{—}z\!\!\rightarrow\!\!s'$. Обозначим $s\text{—}z\!\!\rightarrow =_{def}\exists s'\ s\text{—}z\!\!\rightarrow\!\!s'$. Маршрутом LTS называется последовательность смежных переходов: начало каждого перехода, кроме первого, совпадает с концом предыдущего перехода.

Шаг функционирования LTS внутри машины тестирования сводится к выполнению перехода, определённого в текущем состоянии и разрешаемого нажатой кнопкой (τ - и γ -переходы всегда разрешены). Если таких переходов несколько, выбирается один из них недетерминированным образом.

Состояние s *дивергентно* ($s\uparrow$), если в нём начинается бесконечная цепочка τ -переходов (в частности, τ -цикл); в противном случае состояние *конвергентно* ($s\downarrow$). Состояние *стабильно*, если из него не выходят τ - и γ -переходы. Отказ $P\in\mathbf{R}$ порождается стабильным состоянием, из которого нет переходов по действиям из P.

Добавим в каждом стабильном состоянии s виртуальные петли $s\text{—}P\!\!\rightarrow\!\!s$, помеченные порождаемыми отказами P, и Δ -переходы из дивергентных состояний $s\text{—}\Delta\!\!\rightarrow$. В полученной LTS рассмотрим маршруты, не продолжающиеся после Δ - и γ -переходов. Трассой назовём последовательность пометок на переходах такого маршрута с пропуском символов τ. Если маршрут с трассой σ начинается в состоянии s и заканчивается в состоянии s', то будем обозначать это $s\Rightarrow\sigma\Rightarrow s'$. Обозначим $s\Rightarrow\sigma\Rightarrow =_{def}\exists s'\ s\Rightarrow\sigma\Rightarrow s'$. Множество трасс, начинающихся в состоянии s, обозначим $\boldsymbol{T}(s)=\{\sigma\mid s\Rightarrow\sigma\Rightarrow\}$.

2.3. Гипотеза о безопасности и безопасная конформность

Определим отношение безопасности «*кнопка безопасна после трассы*»: нажатие кнопки P после трассы σ не вызывает попытку выхода из дивергенции (после трассы нет дивергенции) и не вызывает разрушения (после действия, разрешаемого кнопкой). При безопасном взаимодействии будут нажиматься только безопасные кнопки. Формально, кнопка P безопасна:

в состоянии s: P ***safe*** $s =_{def} s\downarrow\ \&\ \neg s\Rightarrow\langle\gamma\rangle\Rightarrow\ \&\ \forall\, z\in P\ \neg s\Rightarrow\langle z,\gamma\rangle\Rightarrow$;

во множестве состояний S: P ***safe*** $S =_{def}\forall s\in S\ P$ ***safe*** s;

после трассы σ, начинающейся в состоянии s: P ***safe*** s ***after*** $\sigma =_{def} P$ ***safe*** $(s$ ***after*** $\sigma)$,

где s ***after*** $\sigma =_{def}\{s'\mid s\Rightarrow\sigma\Rightarrow s'\}$.

Безопасность кнопок влечёт безопасность действий и отказов. Отказ P безопасен, если безопасна кнопка P. Действие z безопасно, если оно разрешается некоторой безопасной кнопкой P, то есть, $z\in P$. Трасса, начинающаяся в состоянии s, безопасна, если 1) $\neg s\Rightarrow\langle\gamma\rangle\Rightarrow$, 2) каждый символ трассы безопасен после непосредственно предшествующего ему префикса трассы. Множества безопасных трасс, начинающихся в состоянии s, обозначим $\boldsymbol{Safe}(s)$. По умолчанию трассы и маршруты начинаются в начальном состоянии LTS.

Требование безопасности взаимодействия выделяет класс реализаций, определяемый *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:

$\mathbf{I}$ ***safe for*** $\mathbf{S} =_{def}(\langle\gamma\rangle\notin\boldsymbol{T}(s_0)\ \Rightarrow\langle\gamma\rangle\notin\boldsymbol{T}(i_0))\ \&\ \forall\sigma\in\boldsymbol{Safe}(s_0)\cap\boldsymbol{T}(i_0)\ \ \forall P\in\mathbf{R}$

$(P$ ***safe*** s_0 ***after*** $\sigma\ \Rightarrow P$ ***safe*** i_0 ***after*** $\sigma)$.

Определим отношение *конформности*: реализация $\mathbf{I}$ *конформна* спецификации $\mathbf{S}$, если она безопасна и выполнено *тестируемое условие*: наблюдение, возможное в реализации в ответ на нажатие безопасной (в спецификации) кнопки, разрешается спецификацией:

$\mathbf{I}$ ***saco*** $\mathbf{S} =_{def}\mathbf{I}$ ***safe for*** $\mathbf{S}\ \&\ \forall\sigma\in\boldsymbol{Safe}(s_0)\cap\boldsymbol{T}(i_0)\ \ \forall P$ ***safe*** s_0 ***after*** σ

$\boldsymbol{obs}(i_0$ ***after*** $\sigma,P)\subseteq\boldsymbol{obs}(s_0$ ***after*** $\sigma,P)$,

где $\boldsymbol{obs}(M,P) =_{def}\{u\in P\cup\{P\}\mid\exists m\in M\ \&\ m\Rightarrow\langle u\rangle\Rightarrow\}$ – множество наблюдений, которые возможны в состояниях из множества M при нажатии кнопки P.

3. Алгоритм тестирования

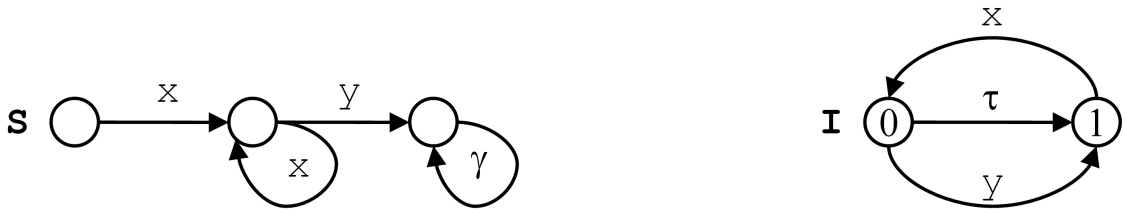
Тестирование применяется тогда, когда в нашем распоряжении имеется исследуемая система, с ней можно взаимодействовать, но ее модель, то есть, реализация, хотя и существует, но неизвестна. Иными словами, тест взаимодействует с неизвестной реализацией. Поскольку такое взаимодействие должно быть безопасным, а при неизвестной реализации гипотезу о безопасности невозможно проверить, приходится предполагать ее выполнение. Она становится предусловием тестирования, а его целью — проверка тестируемого условия. Взаимодействуя с реализацией, мы получаем знания о ее устройстве. При некоторых ограничениях на

семантику взаимодействия, спецификацию и реализацию, а также при наличии дополнительных тестовых возможностей этих знаний оказывается достаточно для полной проверки тестируемого условия за конечное время.

В данной статье мы будем предполагать конечность семантики (алфавита действий), спецификации и реализации (конечны множества состояний и переходов).

Еще одно ограничение касается степени недетерминизма реализации: реализация `t`-недетерминирована, если при повторении одного и того же тестового воздействия в одном и том же состоянии реализации `t` раз реализация демонстрирует все возможные варианты поведения. Это гарантирует получение всех возможных наблюдений при выполнении тестового воздействия `t` раз.

Для полноты тестирования нужно проверить все переходы реализации, лежащие на маршрутах с трассами, безопасными в спецификации. Для этого LTS должна быть сильно-связной: из каждого состояния достижимо по переходам каждое другое состояние (достаточно ограничиться состояниями, достижимыми из начального состояния по безопасным трассам спецификации). Если есть возможность рестарта реализации (повторный запуск теста) в некоторых ее состояниях, то это означает наличие переходов по рестарту, ведущих в начальное состояние, которые 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 (см. Рис.1). Кнопка рестарта отличается от других кнопок только тем, что переход по рестарту, если он есть и безопасен, ведет в начальное состояние реализации/спецификации. Переход по рестарту делает трассу пустой.



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

I ***saco*** **S**: {y} ***safe*** s_0 ***after*** ⟨⟩ & y∈***obs***((i$_0$ ***after*** ⟨⟩,{y}) & y∉***obs***((i$_0$ ***after*** ⟨⟩,{y}). Если в начале тестирования реализация окажется в состоянии 1, то без рестарта она попадёт в состояние 0 только после трассы, содержащей {x}, после которой в спецификации кнопка {y} опасна: её нельзя нажимать при тестировании. Ошибка не обнаружится.

Рис.1 Рестарт

Кроме того, мы предполагаем вести тестирование с открытым состоянием: есть возможность наблюдать текущиее состояние реализации в процессе тестирования.

При этих ограничениях и тестовых возможностях мы будем поэтапно строить реализацию с одновременной проверкой тестируемого условия. Предлагаемый алгоритм является модификацией алгоритма в [5]. Основные идеи доказательств правильности алгоритма и оценок сложности примерно те же самые и для экономии места здесь опущены.

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

R(S)={P∈**R**|P ***safe*** S} – множество безопасных кнопок;

A(S)=(∪{P∪{P}|P ***safe*** S})∪{τ} – множество, состоящее из безопасных наблюдений и символа τ;

B(S,u)=∪{s ***after*** ⟨u⟩ |s∈S} – множество постсостояний переходов по наблюдению `u` из состояний из `S`; доопределим **B**(S,τ)=S. Если таких переходов нет, **B**(S,u)=∅.

Фактически, при этом строится power-LTS, состояниями которой являются множества состояний спецификации S=s_0 ***after*** σ, где σ∈ ***Safe***(s_0), а переход S—u→S` означает, что S`=**B**(S,u)≠∅.

LTS-реализация **I** строится в процессе тестирования. В начале тестирования и после каждого перехода опрашиваем состояние реализации. Переход i—z→i` по внешнему действию `z` добавляется, когда

после нажатия кнопки P в состоянии i наблюдается действие $z \in P$ и постсостояние $i`$. Если наблюдается отказ с *тем же самым* постсостоянием $i`=i$, то добавляется виртуальный переход по отказу $i—P→i$. Если отказ P наблюдается с *другим* постсостоянием $i` \neq i$, то добавляются переходы $i—\tau→i`—P→i`$. Вместе с каждым переходом по внешнему действию $i—z→i`$ будем хранить кнопку $P(i—z→i`)$, нажатие которой вызвало этот переход.

При построении $\mathbf{I}$ с каждым построенным состоянием i будем связывать семейство $\mathbf{S}(i)$ множеств состояний спецификации: $\mathbf{S}(i) = \{s_0 \textit{ after } \sigma \mid \sigma \in \textit{Safe}(s_0) \ \& \ i \in (i_0 \textit{ after } \sigma)\}$. Это семейство состоит из множеств состояний спецификации в конце трасс, безопасных в спецификации, имеющихся в реализации и заканчивающихся там в состоянии i. В процессе тестирования мы будем строить эти семейства $\mathbf{S}(i)$, постепенно добавляя в них множества $s_0 \textit{ after } \sigma$.

Кнопка P *допустима в* i, если она безопасна хотя бы в одном множестве $S \in \mathbf{S}(i)$. 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 $\mathbf{I}$ есть виртуальный переход по отказу $i—P→i$, или 2) $c(P,i)=t$. Это означает, что уже получены все возможные переходы из состояния i при нажатии кнопки P. Состояние *полно*, если каждая допустимая в нем кнопка полна.

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

$\mathbf{R}(i)=\cup\{\mathbf{R}(S) \mid S \in \mathbf{S}(i)\}$ – множество кнопок, допустимых в состоянии i;

$\mathbf{C}(i)=\langle i—u→i` \rangle$ — множество переходов из состояния i.

Если тестируемое условие выполнено, то должно быть: $\forall i—u→i` \in \mathbf{C}(i) \quad \forall S \in \mathbf{S}(i) \ u \in \mathbf{A}(S) \Rightarrow \mathbf{B}(S,u) \neq \varnothing$. Мы будем проверять это на каждом шаге тестирования.

В начале тестирования после опроса состояния в $\mathbf{I}$ есть только одно состояние $i \in (i_0 \textit{ after } \blacksquare)$ и $\mathbf{S}(i)=\{s_0 \textit{ after } \blacksquare\}$, $\mathbf{R}(i)=\mathbf{R}(s_0 \textit{ after } \blacksquare)$, $\mathbf{C}(i)=\varnothing$ и $c(P,i)=0$ для каждой допустимой кнопки.

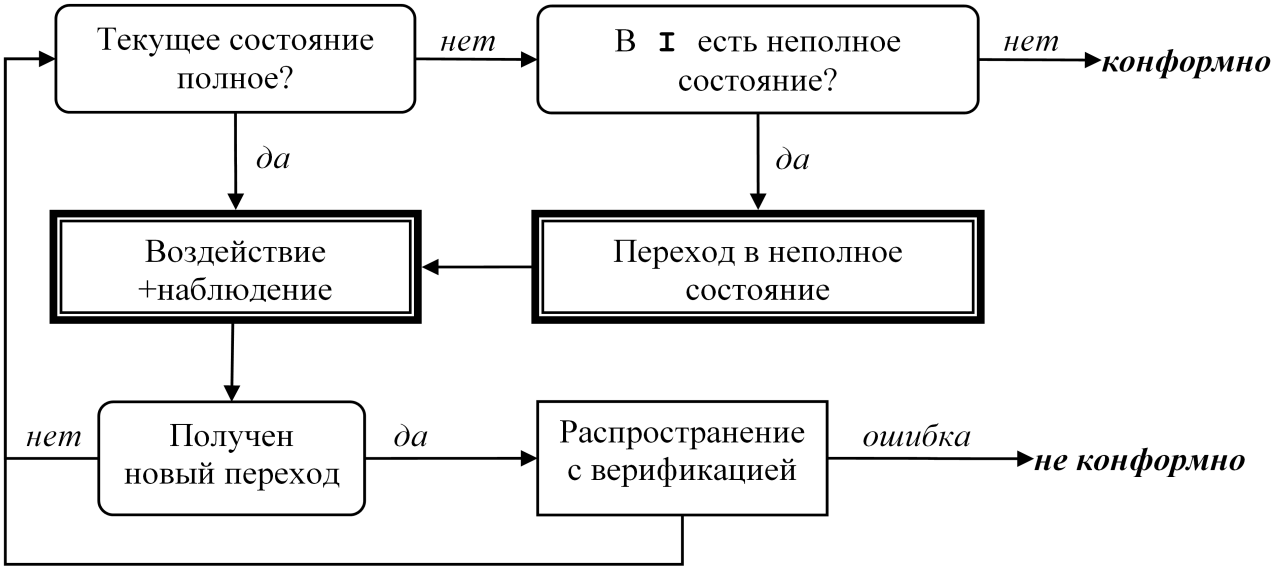


Рис.2 Общая схема алгоритма

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

Рассмотрим переход в неполное состояние. В графе LTS $\mathbf{I}$ всегда существует лес деревьев, покрывающих все состояния и ориентированных к своим корням, которыми являются все неполные состояния. Выберем любой такой лес и будем двигаться по его переходам $i—u→i`$, нажимая кнопки $P(i—u→i`)$. 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 $i``—u→i```$, и будем нажимать кнопку $P(i``—u→i```)$. Доказательство достижения неполного состояния за конечное число шагов см. в [5].

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

После этого выполняется блок «Распространие с верификацией». Для работы этого блока создается вспомогательный список **W** пар (S,j—u→j`), где S∈**S**(j) и j—u→j`∈**C**(j). В самом начале множество **W** содержит все пары (S,j—u→j`) для каждого нового перехода j—u→j` и S∈**S**(j).

Опишем шаг работы блока. Если список **W** пуст, выходим из блока. Иначе, выбираем первый элемент (S,i—u→i`) из списка **W**, удаляя его из списка. Проверяем тестируемое условие: если i—u→i`∈**C**(i) & u∈**A**(S) & **B**(S,u)=∅, фиксируется ошибка и алгоритм заканчивается с вердиктом «неконформно». Если ошибки нет, и **B**(S,u)∈**S**(i`), шаг заканчивается. В противном случае добавляем **B**(S,u) в **S**(i`) и помещаем в список **W** все пары (**B**(S,u),i`—u→i``), где i`—u→i``∈**C**(i`); 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.

Доказательство полноты теста, работающего по описанному алгоритму, аналогично [5].

Приведем оценки сложности алгоритма (доказательство см. в [5]). При тестировании обычно наиболее важным считается число тестовых воздействий. Оценка $O_T=O(bkt^k)$ для $t>1$, и $O_T=O(bk^2)$ для $t=1$, где b — число кнопок, k — число состояний реализации. 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 k. Приведем два таких случая:

1. Сильно-Δ -связные LTS-реализации [1,5]. Это такие LTS, в которых для любой пары состояний a и b можно в каждом состоянии i найти такую кнопку $P(i)$, что, нажимая только такие кнопки, мы проходим тот или иной путь из a в b при любом недетерминированном поведении LTS. Детерминированные LTS — это частный случай сильно-Δ -связных LTS.

2. Недетерминизм может быть следствием повышения уровня абстракции при моделировании детерминированной исследуемой системы реализацией 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 $i \blacksquare u \blacksquare i`$, 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 $i \blacksquare u \blacksquare i`$.

Оценка объема вычислений содержит три слагаемых: 1) вычисления, необходимые для поиска опрошенного состояния среди пройденных при каждом тестовом воздействии, $k\cdot O_T=O(bk^2t^k)$ для $t>1$ или $k\cdot O_T=O(bk^3)$ для $t=1$; 2) построение лесов деревьев $O(b^2tk^2)$; 3) вычисления в блоке «Распространение с верификацией» $O(m2^n)$. Сомножитель 2^n — это максимально возможное число состояний power-LTS спецификации. Если power-преобразование не увеличивает числа состояний, в частности, если спецификация детерминирована, этот сомножитель заменяется на n.

ЛИТЕРАТУРА:

1. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин "Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай" // Программирование. 2004. No. 1
2. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин "Формализация тестового эксперимента" // Программирование, 2007, No. 5
3. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин "Теория соответствия для систем с блокировками и разрушением". «Наука», 2008
4. И.Б. Бурдонов "Теория конформности для функционального тестирования программных систем на основе формальных моделей". Диссертация на соискание учёной степени д.ф.-м.н., Москва, 2008
5. И.Б. Бурдонов, А.С. Косачев "Полное тестирование с открытым состоянием ограниченно недетерминированных систем". Труды ИСП РАН, т. 16, 2009
6. R.J. van Glabbeek "The linear time - branching time spectrum II; the semantics of sequential processes with silent moves". CONCUR'93 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993, pp. 66–81
7. R. Milner "Modal characterization of observable machine behaviour". CAAP 81 (G. Astesiano and C. Bohm, ed.), LNCS 112, Springer-Verlag, 1981, pp. 25–34