И.Б.Бурдонов, А.С.Косачев.
Тестирование с преобразованием семантик.
Труды Института системного программирования РАН, N 17, 2009, стр.193-208.
16 стр.

Тестирование с преобразованием семантик

Бурдонов И.Б., Косачев А.С.
Институт системного программирования РАН,
{igor,kos}@ispras.ru

Аннотация. В статье представлен подход к формальному описанию тестирования в случае, когда тестовые семантики спецификации и реализации различаются. Для этого используется медиатор, осуществляющий преобразование спецификационных тестовых воздействий в реализационные и обратное преобразование реализационных наблюдений в спецификационные. Формально переопределяются условия тестирования и конформность. Также рассматривается тестирование с заданным преобразованием реализационных состояний в спецификационные. Определяются условия и способ построения теста, обнаруживающего все ошибки в реализации за конечное время.

1. Введение

Тестирование – это проверка в процессе эксперимента соответствия (конформности) реализации требованиям, заданным в виде спецификации. Это соответствие определяется семантикой тестового взаимодействия, которая описывает возможные тестовые воздействия и возможные наблюдения ответного поведения реализации. В теории конформности обычно предполагается, что реализация и спецификация заданы в одной семантике. Однако на практике обычно требуется некоторое преобразование спецификационных тестовых воздействий в реализационные тестовые воздействия и обратное преобразование реализационных наблюдений в спецификационные наблюдения. Программа, осуществляющая эти преобразования, называется медиатором. По сути, это означает, что реализация может быть задана в другой тестовой семантике, и медиатор осуществляет преобразование семантик.

В простейшем случае различие семантик только в разных способах представления одних и тех же действий в реализации и спецификации. Достаточно взаимно-однозначного преобразования алфавитов реализации и спецификации. В общем случае такого преобразования недостаточно, поскольку спецификация, как правило, определяется на более высоком уровне абстракции. Медиатор может оказаться довольно сложной программой, осуществляющей медиативные функции между реализацией в одной семантике и тестом, генерируемым по спецификации в другой семантике.

Кроме тестовых воздействий и наблюдений, медиатор может преобразовывать также и состояния: из реализационного в спецификационное. Это означает, что на множестве состояний реализации предполагается заданным отношение эквивалентности, и состояние спецификации соответствует классу эквивалентных состояний реализации. Это соответствие является не предусловием тестирования (реализационной гипотезой), а частью проверяемого условия.

Статья посвящена проблемам, возникающим при тестировании через медиатор. 2-ой раздел содержит основные положения теории конформности для совпадающих семантик реализации и спецификации. В 3-ем разделе рассматривается медиаторное преобразование тестовых воздействий и наблюдений, определяется устройство теста и медиатора и переформулируется определение конформности. 4-ый раздел посвящен преобразованию состояний, описываются соответствующие модификации медиатора и конформности. В 5-ом разделе способ полного (обнаруживающего все ошибки) тестирования за конечное время распространяется на случай тестирования через медиатор.

2. Теория конформности

2.1. Семантика взаимодействия и безопасное тестирование

Данная работа опирается на теорию тестирования конформности, изложенную в [3,4,5]. Напомним основные положения этой теории. Тестирование – это проверка в процессе эксперимента соответствия (конформности) реализации требованиям, заданным в виде спецификации.

Семантика тестового взаимодействия определяется в терминах *действий* и *кнопок*. Действие – это внешнее действие тестируемой системы (реализации), которое может наблюдаться при тестировании. Множество внешних действий называется алфавитом действий и обозначается $\mathbf{L}$. Кнопка – это подмножество $P \subseteq \mathbf{L}$; нажатие кнопки P моделирует тестовое воздействие на реализацию, сводящееся к разрешению выполнять любое действие из P. При нажатии кнопки P наблюдается либо действие $a \in P$, выполняемое реализацией, либо – для некоторых кнопок – отсутствие таких действий, называемое отказом P. Семантика взаимодействия задаётся алфавитом $\mathbf{L}$ и двумя наборами кнопок: с наблюдением соответствующих отказов – семейство $\mathbf{R} \subseteq \mathcal{P}(\mathbf{L})$ и без наблюдения отказов – семейство $\mathbf{Q} \subseteq \mathcal{P}(\mathbf{L})$. Предполагается, что $\mathbf{R} \cap \mathbf{Q} = \varnothing$ и $\cup\mathbf{R} \cup \cup\mathbf{Q} = \mathbf{L}$. Такая семантика называется **R**/**Q**-семантикой.

Кроме внешних действий реализация может совершать внутренние (ненаблюдаемые) действия, обозначаемые τ. Эти действия всегда разрешены. Предполагается, что любая конечная последовательность любых действий совершается за конечное время, а бесконечная – за бесконечное время. Бесконечная последовательность τ-действий («зацикливание») называется *дивергенцией* и обозначается Δ. Кроме этого вводится специальное, также не регулируемое кнопками, действие, которое называется *разрушением* и обозначается γ. Оно моделирует любое запрещённое при тестировании поведение реализации. Дивергенция сама по себе не опасна, но при попытке выхода из неё, когда нажимается любая кнопка, неизвестно, нужно ли ждать наблюдения или бесконечно долго будут выполняться только внутренние действия. Поэтому нельзя ни продолжить тестирование, ни закончить его. Тестирование, при котором не возникает разрушения и попыток выхода из дивергенции, называется безопасным.

2.2. LTS-модель и трассы наблюдений

В качестве модели реализации и спецификации используется *система помеченных переходов* (LTS – Labelled Transition System) – ориентированный граф с выделенной начальной вершиной, дуги которого помечены некоторыми символами. Формально, LTS – это совокупность $\mathbf{S}=\mathrm{LTS}(V_S,\mathbf{L},E_S,s_0)$, где V_S – непустое множество состояний (вершин графа), $\mathbf{L}$ – алфавит внешних действий, $E_S\subseteq V_S\times(\mathbf{L}\cup\{\tau,\gamma\})\times V_S$ – множество переходов (помеченных дуг графа), $s_0\in V_S$ – начальное состояние (начальная вершина графа). Переход из состояния s в состояние s' по действию z обозначается $s\text{—}z\to s'$. Обозначим $s\text{—}z\to \triangleq \exists s'\ s\text{—}z\to s'$ и $s\text{—}z\not\to \triangleq \nexists s'\ s\text{—}z\to s'$. Маршрутом LTS называется последовательность смежных переходов: начало каждого перехода, кроме первого, совпадает с концом предыдущего перехода.

Выполнение LTS в машине тестирования сводится к выполнению того или иного перехода, определённого в текущем состоянии и разрешаемого нажатой кнопкой (τ- и γ-переходы всегда разрешены).

Состояние *стабильно*, если из него не выходят τ- и γ-переходы, и *дивергентно*, если в нём начинается бесконечная цепочка τ-переходов (в частности, τ-цикл). Отказ $P\in\mathbf{R}$ порождается стабильным состоянием, из которого нет переходов по действиям из P. Переход $s\text{—}z\to s'$ *разрушающий*, если из s' по цепочке (возможно пустой) τ-переходов достижимо начало γ-перехода.

Добавим в каждом стабильном состоянии LTS **S** виртуальные петли, помеченные порождаемыми отказами, и Δ-переходы из дивергентных состояний. В полученной LTS рассмотрим маршруты, начинающиеся в начальном состоянии и не продолжающиеся после Δ- и γ-переходов. **R**-трассой LTS **S** назовём последовательность пометок на переходах такого маршрута с пропуском символов τ. Множество **R**-трасс LTS **S** обозначим $\boldsymbol{T}(\mathbf{S})$.

2.3. Гипотеза о безопасности и безопасная конформность

Если нажатие кнопки P после **R**-трассы σ не вызывает попытку выхода из дивергенции (после трассы нет дивергенции) и не вызывает разрушение (после действия, разрешаемого кнопкой), то кнопка P называется безопасной в LTS после σ. При безопасном тестировании нажимаются только безопасные кнопки. Формально, кнопка P безопасна:

в состоянии S: P ***safe*** s, если состояние s не дивергентно и в каждом состоянии s', достижимом из s по τ-переходам, нет γ-переходов, а все переходы $s' \text{—} z \rightarrow s''$, где $z \in P$ неразрушающие;

во множестве состояний S: P ***safe*** $S \triangleq \forall s \in S\ P$ ***safe*** s;

после трассы σ: P ***safe*** $\mathbf{S}$ ***after*** $\sigma \triangleq P$ ***safe*** $(\mathbf{S}$ ***after*** $\sigma)$,

где $\mathbf{S}$ ***after*** σ – множество состояний LTS **S** после трассы σ, то есть состояний, достижимых из начального состояния по трассе σ.

Безопасность кнопок влечёт безопасность действий и отказов после трассы. Отказ R ***safe*** $(\mathbf{S}$ ***after*** $\sigma)$, если после трассы σ безопасна кнопка R. Действие z ***safe*** $(\mathbf{S}$ ***after*** $\sigma)$, если оно разрешается $(z \in P)$ некоторой кнопкой P ***safe*** $(\mathbf{S}$ ***after*** $\sigma)$. **R**-трасса безопасна, если 1) LTS не разрушается с самого начала, то есть в ней нет трассы $\langle\gamma\rangle$, 2) каждый символ трассы безопасен после непосредственно предшествующего ему префикса трассы. Множества безопасных трасс LTS **S** обозначим $\boldsymbol{Safe}(\mathbf{S})$.

Требование безопасности тестирования выделяет класс *безопасных* реализаций, которые могут быть безопасно протестированы для проверки их конформности заданной спецификации. Этот класс определяется *гипотезой о безопасности*: реализация **I** *безопасна* для спецификации **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*** $\mathbf{S} \triangleq (\langle\gamma\rangle \notin \boldsymbol{T}(\mathbf{S}) \Rightarrow \langle\gamma\rangle \notin \boldsymbol{T}(\mathbf{I}))$

& $\forall\sigma\in$***Safe***(**S**)$\cap$***T***(**I**) $\forall$P$\in$**R** (P ***safe*** **S** ***after*** σ $\Rightarrow$ P ***safe*** **I** ***after*** σ).

Гипотеза о безопасности не проверяема при тестировании и является его предусловием. Отношение *конформности* формально определяется так: реализация **I** *конформна* спецификации **S**, если она безопасна и выполнено *проверяемое условие*: любое наблюдение, возможное в реализации в ответ на нажатие безопасной (в спецификации) кнопки, разрешается спецификацией:

I ***saco*** **S** $\triangleq$ **I** ***safe for*** **S** & $\forall\sigma\in$***Safe***(**S**)$\cap$***T***(**I**) $\forall$P$\in$**R**

(P ***safe*** **S** ***after*** σ $\Rightarrow$ ***obs***(**I** ***after*** σ,P)$\subseteq$***obs***(**S** ***after*** σ,P)),

где ***obs***(M,P) $\triangleq$ {u|$\exists$m$\in$M u$\in$P & m—u$\rightarrow$ $\vee$ u=P & $\forall$z$\in$P m—z$\not\rightarrow$} – множество наблюдений, которые возможны в состояниях из множества M при нажатии кнопки P.

В [3,5] показано, что достаточно ограничиться семантиками, в которых все отказы наблюдаемы: **Q**=$\varnothing$. Любая **R**/**Q**-семантика эквивалентна **R**$\cup$**Q**/$\varnothing$-семантике: для любой спецификации в **R**/**Q**-семантике существует (и может быть построена при некоторых, практически приемлемых, ограничениях) спецификация в **R**$\cup$**Q**/$\varnothing$-семантике, определяющая тот же класс конформных реализаций и не меньший класс реализаций, поддающихся тестированию. Поэтому мы будем рассматривать только **R**-семантики, то есть **Q**=$\varnothing$.

2.4. Генерация тестов

Тест – это инструкция, состоящая из терминальных и нетерминальных пунктов. В каждом пункте указывается кнопка, которую нужно нажимать, и для каждого наблюдения, возможного после нажатия этой кнопки, – пункт инструкции, который должен выполняться следующим, или вердикт (***pass*** или ***fail***), если тестирование нужно закончить. В [3,4,5] такая инструкция соответствует формальному определению *управляемого* LTS-теста, который однозначно определяет тестирование (без лишнего недетерминизма).

Реализация *проходит* тест, если её тестирование 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. Задача заключается в генерации полного набора тестов по спецификации.

Полный набор тестов всегда существует, в частности, им является набор всех *примитивных* тестов. Такой тест строится по одной выделенной безопасной **R**-

трассе спецификации. Для этого в трассу вставляются кнопки, которые нужно нажимать: перед каждым отказом `R` вставляется кнопка `R`, а перед каждым действием `z` – какая-нибудь безопасная (после префикса трассы) кнопка `P`, разрешающая действие `z`. Безопасность трассы гарантирует безопасность кнопки `R` и наличие такой безопасной кнопки `P`. Выбор кнопки `P` может быть неоднозначным: по одной безопасной трассе спецификации можно сгенерировать, вообще говоря, несколько разных примитивных тестов. Если наблюдение, полученное после нажатия кнопки, продолжает трассу, тест продолжается. Последнее в трассе наблюдение и любое наблюдение, «ведущее в сторону», всегда заканчивают тестирование. Вердикт ***pass*** выносится, если полученная **R**-трасса есть в спецификации, а вердикт ***fail*** – если нет. Такие вердикты соответствуют *строгим* тестам, которые, во-первых, значимые (не ловят ложных ошибок) и, во-вторых, не пропускают обнаруженных ошибок. Любой строгий тест сводится к некоторому множеству примитивных тестов, которые обнаруживают те же самые ошибки.

3. Преобразование кнопок и наблюдений

3.1. Медиатор семантик

В теории конформности предполагается, что реализация и спецификация заданы в одной **R**-семантике. Однако на практике обычно требуется некоторое преобразование спецификационных тестовых воздействий в реализационные тестовые воздействия и обратное преобразование реализационных наблюдений в спецификационные наблюдения. Программа, осуществляющая эти преобразования, называется медиатором. По сути, это означает, что реализация может быть задана в другой тестовой семантике – **R`**-семантике, и медиатор осуществляет преобразование семантик.

В простейшем случае различие семантик только в разных способах представления одних и тех же действий в реализации и спецификации. Достаточно взаимно-однозначного преобразования алфавитов реализации и спецификации. В общем случае такого преобразования недостаточно, поскольку спецификация, как правило, определяется на более высоком уровне абстракции. Рассмотрим несколько примеров.

1. Передача сообщений. В спецификации может быть определена операция (кнопка) ***send***`(m)` передачи сообщения `m`, а в реализации – ***send***`(m,i)` с указанием дополнительного параметра `i` – номера одного из нескольких выходных портов. На уровне спецификации тест принимает любое сообщение, и отказ означает отсутствие сообщений. На уровне реализации тест может принимать сообщение только из одного порта, и для наблюдения

спецификационного отказа нужно выполнить прием по каждому порту и убедиться, что ни по одному из них сообщений нет. В этом примере одной спецификационной кнопке ***send***`(m)` соответствует множество реализационных кнопок ***send***`(m,i)`, каждую из которых нужно нажать в том же самом состоянии реализации для того, чтобы при полном тестировании осуществить требуемое спецификационное тестовое воздействие для всех возможных ситуаций. Поскольку отказ не меняет состояния, последовательность отказов по всем портам гарантированно означает спецификационный отказ для того же самого состояния реализации. Однако, если из некоторого порта сообщение принимается, то состояние реализации меняется и прием из другого порта можно будет сделать только тогда, когда реализация снова окажется в том же состоянии, а тест объявит прием сообщений. Таким образом медиаторное преобразование, во-первых, адаптивно, то есть реализационные тестовые воздействия зависят от полученных реализационных наблюдений (сообщение или отказ), и, во-вторых, зависит от состояния реализации, в котором оно начинает выполняться.

2. Пул элементов с операциями ***alloc*** и ***dealloc***. В спецификации пула может быть определена операция ***dealloc*** без параметров, понимаемая как освобождение любого ранее занятого элемента пула, в то время как в реализации эта операция требует в качестве параметра указать конкретный освобождаемый элемент ***dealloc***`(i)`. Будем предполагать, что реализация удовлетворяет следующей реализационной гипотезе: все состояния пула, соответствующие одному и тому же множеству занятых элементов, эквивалентны. Здесь аналогично предыдущему примеру одной спецификационной кнопке ***dealloc*** соответствует множество (по числу занятых элементов) реализационных кнопок ***dealloc***`(i)`, каждую из которых нужно нажать хотя бы в одном состоянии реализации, соответствующем данному множеству занятых атомов. Здесь есть два отличия от предыдущего примера. 1) Медиаторное преобразование не адаптивно, точнее, от наблюдений (выполнение ***dealloc***`(i)` или отказ) зависит только соответствующее спецификационное наблюдение, возвращаемое в тест, но не последующие реализационные тестовые воздействия: для каждого нажатия кнопки ***dealloc*** выполняется только одно нажатие одной кнопки ***dealloc***`(i)`. 2) Преобразование зависит от множества занятых элементов, вычисляемое по предыстории взаимодействия. По реализационной гипотезе, зависимость от реализационного состояния можно не учитывать, поскольку состояния с одним множеством занятых элементов эквивалентны.

3. Стационарное тестирование. Спецификация описывает конечный автомат, каждый переход которого помечен парой (стимул $x \in X$, реакция $y \in Y$);

спецификационная кнопка $P(x)=\{(x,y)|y\in Y\}$ означает посылку стимула x и прием любой реакции. Реализация представляет собой LTS, каждый внешний переход которой помечен либо стимулом $x\in X$, либо реакцией $y`\in Y`$; реализационные кнопки: $P`(x)=\{x\}$, означающая посылку стимула $x$, и $P`=Y`$, означающая прием любой реализационной реакции. Медиатор посылает стимул $x$ и либо возвращает отказ, если наблюдается отказ, либо принимает реакции до тех пор, пока они поступают от реализации, а затем для полученной (быть может, пустой) последовательности $y`_1,\ldots,y`_n$ реализационных реакций выдает спецификационную реакцию $y=f(y`_1,\ldots,y`_n)$. Функция f как раз и определяет медиаторное преобразование. Тестирование через такой медиатор иногда называют стационарным тестированием, поскольку стимулы подаются только в стационарных состояниях реализации, то есть в состояниях, в которых нет реакций [2,7]. Здесь спецификационной кнопке соответствует последовательность реализационных тестовых воздействий, в процессе выполнения которой состояние реализации меняется, но, в отличие от первого примера, нас интересует состояние только в конце последовательности, соответствующее полному выполнению спецификационного тестового воздействия. Преобразование адаптивно, но не зависит от реализационных состояний.

В общем случае медиатор может оказаться довольно сложной программой, осуществляющей медиативные функции между реализацией в **R`**-семантике и тестом, генерируемым по спецификации в **R**-семантике. Медиатор компонуется с тестом; результат такой композиции рассматривается как тест в **R`**-семантике, который взаимодействует напрямую с реализацией.

Тестирование через медиатор похоже на тестирование в контексте (асинхронное тестирование) [4,5,8,10,13]. Там тоже тест взаимодействует с реализацией не напрямую, а через промежуточную среду взаимодействия. Отличие в том, что при асинхронном тестировании спецификация задаётся в той же семантике, что и реализация. Проблема заключается в том, что тесты, генерируемые по исходной спецификации, при асинхронном тестировании могут обнаруживать ложные ошибки (не обнаруживаемые при синхронном тестировании, когда тест и реализация взаимодействуют напрямую). Для решения этой проблемы спецификация монотонно преобразуется (с сохранением класса конформных и не сужением класса безопасных реализаций) и компонуется с известной средой передачи. По этой композиции и генерируются тесты. В отличие от этого при тестировании через медиатор тест генерируется по исходной спецификации, семантика которой отлична от

семантики реализации, и компонуется с медиатором, в результате чего получается тест в семантике реализации.

Тестирование через медиатор также похоже на тестирование по фактор-спецификации [1]. Идея факторизации заключается в том, что на исходной LTS-спецификации, заданной в реализационной семантике, вводятся отношения эквивалентности состояний и переходов, которые индуцируют фактор-LTS: фактор-состояние – это класс эквивалентности состояний, а фактор-переход – это множество эквивалентных переходов, имеющих эквивалентные начальные состояния и эквивалентные конечные состояния. Действие, которым помечен фактор-переход – это класс эквивалентности переходов. Для фактор-спецификации определяется фактор-семантика, которая может быть любой, но должна быть согласована с реализационной семантикой и эквивалентностью переходов. В исходной спецификации для данного состояния фактор-кнопка определяет множество переходов, начинающихся в этом состоянии и отображаемых в фактор-действия, принадлежащие этой фактор-кнопке. Требуется, чтобы это множество переходов совпадало с множество переходов из этого состояния, разрешаемых некоторым множеством реализационных кнопок. Нажатие фактор-кнопки транслируется в нажатие одной из этих реализационных кнопок. Это согласование является частным случаем соотношения спецификационных и реализационных кнопок при тестировании через медиатор, как в первых двух примерах, приведенных выше. Существенное различие в цели тестирования. Факторизация преследует цель сократить время тестирования, основываясь на следующей реализационной гипотезе: если реализация ведет себя правильно при нажатии данной реализационной кнопки в данном состоянии, то она ведет себя правильно при нажатии любой реализационной кнопки, соответствующей той же фактор-кнопке, в любом эквивалентном состоянии. По этой причине нам достаточно хотя бы в одном состоянии из класса эквивалентности нажать хотя бы одну реализационную кнопку, соответствующую данной фактор-кнопке. Тестирование через медиатор не использует эту гипотезу, и для его полноты требуется в каждом состоянии нажать, вообще говоря, каждую реализационную кнопку, соответствующую данной спецификационной кнопке (в случае, когда спецификационной кнопке соответствует именно множество реализационных кнопок, как в первых двух примерах).

3.2. Устройство теста и медиатора

При взаимодействии теста непосредственно с реализацией нажимаются реализационные кнопки и получаются реализационные наблюдения. При тестировании через медиатор с реализацией аналогичным образом взаимодействует медиатор, а интерфейс медиатора и теста будет уже другой.

Работа теста, генерируемого по спецификации в **R**-семантике, – это последовательность пар «тестовое воздействие (кнопка $P \in \mathbf{R}$), получение наблюдения $a \in P$ или $a=P$». В медиаторе этой паре соответствует тройка «получение кнопки $P \in \mathbf{R}$, взаимодействие с реализацией в **R`**-семантике, выдача наблюдения $a$». Взаимодействие медиатора с реализацией аналогично взаимодействию теста и реализации в одной **R`**-семантике. Но есть два отличия. 1) Вместо вынесения вердикта выдается спецификационное наблюдение. 2) Такое взаимодействие начинается после приема той или иной спецификационной кнопки.

В медиаторе выделяются состояния трех типов. В состоянии 1-го типа принимаются спецификационные кнопки, причем медиатор должен быть готов принять любую такую кнопку. После приема кнопки медиатор осуществляет взаимодействие с реализацией в состояниях 2-го типа, которое за конечное число шагов должно заканчиваться переходом в состояние 3-го типа. Там выдается спецификационное наблюдение с переходом в состояние 1-го типа.

Мы будем записывать $\sigma`/\mathcal{M}(P)=a/\lambda$, если после прохождения в процессе тестирования реализационной трассы $\sigma`$, тест обращается к медиатору с кнопкой P, реализация взаимодействует с медиатором, проходя трассу λ, после чего медиатор возвращает тесту наблюдение a. Медиатор формально можно задать в виде отображения $\mathbf{M}:\mathbf{R}\times\mathbf{N}`^*\to\mathcal{P}(\mathbf{N}`^*\times\mathbf{N})$, где $\mathbf{N}=\mathbf{L}\cup\mathbf{R}$ множество наблюдений в **R**-семантике, с тем ограничением, что после нажатия кнопки может наблюдаться только разрешаемое ею действие или отказ: $\forall P\in\mathbf{R}\ \forall\sigma`\in\mathbf{N}`^*\ \forall(\lambda,a)\in\mathbf{M}(P,\sigma`)\ a\in P\cup\{P\}$. Кроме этого медиатор (как и реализационный тест) должен быть готов получить любое наблюдение от реализации, которое может быть после нажатия реализационной кнопки. Это означает, что замыкание по взятию префикса множества трасс λ, генерируемое отображением $\mathbf{M}(P,\sigma`)$, ветвится в каждой точке по всем наблюдениям, соответствующим некоторой реализационной кнопке.

В общем случае медиаторное преобразование может зависеть от реализационного состояния или функции от него. Если такой зависимости нет, достаточно иметь одно состояние 1-го типа. Зависимость от состояния выражается в наличии нескольких состояний 1-го типа. При этом одному состоянию 1-го типа может соответствовать, вообще говоря, множество эквивалентных состояний.

Если медиатор зависит от реализационного состояния, то, прежде всего, должна быть возможность опросить это состояние. Тестирование с такой

возможностью называется тестированием с открытым состоянием [2,6,11]. Предполагается, что опрос состояния достоверен, то есть в каждом состоянии возвращает именно это состояние. Это можно понимать как добавление кнопки опроса состояния $\mathtt{P_0`}$ (множества возможных состояний реализации) к $\mathbf{R`}$. Алфавит реализации расширяется её состояниями. В каждом состоянии $\mathtt{i}$ реализации добавляется переход-петля по опросу состояния $\mathtt{i—i→i}$. Тогда зависимость медиаторного преобразования от состояния сводится к адаптивности, то есть зависимости от наблюдений.

При тестировании с открытым состоянием при некоторых дополнительных ограничениях на реализацию появляется возможность полного тестирования за конечное время. Для этого нужно, чтобы тест также мог опросить состояние реализации, используя для этой цели обращение к медиатору с опросом состояния, которому мы поставим в соответствие спецификационную кнопку «опрос реализационного состояния», также обозначаемую как $\mathtt{P_0`}$.

В некоторых случаях (пример с пулом) вместо реализационного состояния медиатор сообщает тесту некоторое обобщенное состояние как функцию от реализационного состояния, но вычисляемую по предыстории взаимодействия (множество занятых элементов). Тестирование также может быть полным, если выполнена реализационная гипотеза об эквивалентности реализационных состояний, соответствующих одному обобщенному. Если эта гипотеза справедлива, опрос состояния реализации не требуется.

3.3. Гипотеза о безопасности и конформность

Медиаторное отображение $\mathbf{M}$ генерирует отображение трасс $\mathbf{M_T}:\mathbf{N}^*\to\mathcal{P}(\mathbf{N`}^*)$, дающее то множество реализационных трасс, которые могут быть получены при прохождении данной спецификационной трассы. Оно определяется следующими правилами:

1) $\mathbf{M_T}(\epsilon)=\{\epsilon\}$,

2) $\forall\sigma\in\mathbf{N}^*\ \forall\mathtt{a}\in\mathbf{N}\ \mathbf{M_T}(\sigma\cdot\langle\mathtt{a}\rangle)=\{\sigma`\cdot\lambda\mid\sigma`\in\mathbf{M_T}(\sigma)\ \&\ \exists\mathtt{P}\in\mathbf{R}\ (\lambda,\mathtt{a})\in\mathbf{M}(\mathtt{P},\sigma`)\}$.

На основе отображения трасс переопределим гипотезу о безопасности и конформность для тестирования через медиатор.

$\mathbf{I}$ ***safe for*** $\mathbf{S}\ \triangleq\ (\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{S})\Rightarrow\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{I}))\ \&\ \forall\sigma\in\boldsymbol{Safe}(\mathbf{S})\ \forall\sigma`\in\mathbf{M_T}(\sigma)\cap\boldsymbol{T}(\mathbf{I})$
$\forall\mathtt{P}\in\mathbf{R}\ \forall(\lambda,\mathtt{a})\in\mathbf{M}(\mathtt{P},\sigma`)\ (\mathtt{P}$ ***safe*** $\mathbf{S}$ ***after*** $\sigma\ \Rightarrow\ \sigma`\cdot\lambda\in\boldsymbol{Safe}(\mathbf{I}))$.

$$\mathbf{I}\ \textit{saco}\ \mathbf{S} \triangleq \mathbf{I}\ \textit{safe for}\ \mathbf{S}\ \&\ \forall\sigma\in\textit{Safe}(\mathbf{S})\ \forall\sigma`\in\mathbf{M}_\tau(\sigma)\cap\mathbf{T}(\mathbf{I})$$
$$\forall P\in\mathbf{R}\ \forall(\lambda,a)\in\mathbf{M}(P,\sigma`)\ (P\ \textit{safe}\ \mathbf{S}\ \textit{after}\ \sigma\ \Rightarrow\ a\in\textit{obs}(\mathbf{S}\ \textit{after}\ \sigma,P)).$$

4. Преобразование состояний

Кроме кнопок и действий, медиатор может преобразовывать также и состояния: из реализационного в спецификационное. Это означает, что на множестве состояний реализации предполагается заданным отношение эквивалентности, и состояние спецификации соответствует классу эквивалентных состояний реализации. В отличие от факторизации это соответствие является не предусловием тестирования (реализационной гипотезой), а частью проверяемого условия. Правда, эта проверка выполняется косвенным образом: если состояние реализации на самом деле не соответствует состоянию спецификации, в которое оно преобразуется медиатором, то это проявится после нажатия в этом состоянии реализации некоторой спецификационной кнопки как спецификационное наблюдение, отсутствующее в спецификации.

Преобразование состояния осуществляется медиатором в ответ на дополнительную спецификационную кнопку «опрос спецификационного состояния» P_0, добавляемую к семейству **R**. Алфавит спецификации расширяется её состояниями. В каждом состоянии s спецификации добавляется переход-петля по опросу состояния $s\text{—}s\to s$. Медиаторное отображение, гипотеза о безопасности и конформность в измененной семантике определяются как обычно. Опишем, как это выглядит в терминах исходной семантики с учетом того, что трассы теперь содержат состояния, как особые наблюдения, а в спецификации и реализации добавлены переходы-петли по состояниям.

Обращение к медиатору для опроса спецификационного состояния будем записывать как $\mathcal{M}_0()=s$. Медиатор задает отображение $\mathbf{M}_0:V_\mathbf{I}\to V_\mathbf{S}$, где $V_\mathbf{I}$ и $V_\mathbf{S}$ множества состояний реализации **I** и спецификации **S**, соответственно[1].

При генерации трасс учитывается опрос состояния $\mathbf{M}_\tau:\mathbf{K}^*\to\mathcal{P}(\mathbf{K}`^*)$, где $\mathbf{K}=\mathbf{N}\times V_\mathbf{S}$, $\mathbf{K}`=\mathbf{N}`\times V_\mathbf{I}$:

1) $\mathbf{M}_\tau(\epsilon)=\{\epsilon\}$,

[1] В случае обобщенного состояния вместо $V_\mathbf{I}$ используется множество обобщенных состояний.

2) $\forall\sigma\in\mathbf{K}^*\ \forall a\in\mathbf{N}\ \mathbf{M}_T(\sigma\cdot\langle a\rangle)=\{\sigma`\cdot\lambda\,|\,\sigma`\in\mathbf{M}_T(\sigma)\ \&\ \exists P\in\mathbf{R}\ (\lambda,a)\in\mathbf{M}(P,\sigma`)\}$,

3) $\forall\sigma\in\mathbf{K}^*\ \forall s\in V_{\mathbf{S}}\ \mathbf{M}_T(\sigma\cdot\langle s\rangle)=\{\sigma`\cdot\langle i\rangle\,|\,\sigma`\in\mathbf{M}_T(\sigma)\ \&\ s=\mathbf{M}_0(i)\}$.

Опрос состояния безопасен в состоянии, которое не дивергентно и из которого не достижимы по τ-переходам состояния с γ-переходами. Соответственно, гипотеза о безопасности определяется так:

$\mathbf{I}$ ***safe for*** $\mathbf{S}\ \triangleq\ (\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{S})\Rightarrow\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{I}))\ \&\ \forall\sigma\in\boldsymbol{Safe}(\mathbf{S})\ \forall\sigma`\in\mathbf{M}_T(\sigma)\cap\boldsymbol{T}(\mathbf{I})$
$\forall P\in\mathbf{R}\ \forall(\lambda,a)\in\mathbf{M}(P,\sigma`)\ (P$ ***safe*** $\mathbf{S}$ ***after*** $\sigma\ \Rightarrow\ \sigma`\cdot\lambda\in\boldsymbol{Safe}(\mathbf{I}))$
$\&\ (P_0$ ***safe*** $\mathbf{S}$ ***after*** $\sigma\ \Rightarrow\ P_0`$ ***safe*** $\mathbf{I}$ ***after*** $\sigma`)$.

Если после безопасной трассы спецификации опрос состояния P_0 безопасен, то состояние, возвращаемое медиатором, должно быть одним из состояний спецификации после этой трассы. Соответственно, конформность модифицируется так:

$\mathbf{I}$ ***saco*** $\mathbf{S}\ \triangleq\ \mathbf{I}$ ***safe for*** $\mathbf{S}\ \&\ \forall\sigma\in\boldsymbol{Safe}(\mathbf{S})\ \forall\sigma`\in\mathbf{M}_T(\sigma)\cap\boldsymbol{T}(\mathbf{I})$
$\forall P\in\mathbf{R}\ \forall(\lambda,a)\in\mathbf{M}(P,\sigma`)\ (P$ ***safe*** $\mathbf{S}$ ***after*** $\sigma\ \Rightarrow\ a\in\boldsymbol{obs}(\mathbf{S}$ ***after*** $\sigma,P))$
$\&\ \forall i\in(\mathbf{I}$ ***after*** $\sigma`)\ (P_0$ ***safe*** $\mathbf{S}$ ***after*** $\sigma\ \Rightarrow\ \mathbf{M}_0(i)\in(\mathbf{S}$ ***after*** $\sigma))$.

Такая конформность похожа на слабую симуляцию [9,12], которая определяется как соответствие $\boldsymbol{r}\subseteq\boldsymbol{der}(I)\times\boldsymbol{der}(S)$ достижимых состояний реализации и спецификации[2], удовлетворяющее требованию: $(i_0,s_0)\in\boldsymbol{r}\ \&$ $\forall(i,s)\in\boldsymbol{r}\ \forall\sigma\in\mathbf{L}^*\ \forall i`\ (i=\sigma\Rightarrow i`\ \Rightarrow\ \exists s`\ s=\sigma\Rightarrow s`\ \&\ (i`,s`)\in\boldsymbol{r})$, где $i=\sigma\Rightarrow i`$ означает, что состояние $i`$ достижимо из состояния i по трассе σ: существует маршрут, начинающийся в i, заканчивающийся в $i`$ и имеющий трассу σ.

Наша конформность имеет три существенных отличия от слабой симуляции. 1) *Отказы*: в нашем случае рассматриваются трассы с отказами, то есть не в алфавите $\mathbf{L}^*$, а в расширенном алфавите $\mathbf{N}^*$. Это предъявляет более жесткие требования к реализации. 2) *Безопасность*: в нашем случае рассматриваются не любые трассы, а только те, которые безопасны в спецификации, и не любые достижимые состояния, а только те, которые достижимы по безопасным трассам спецификации. Это позволяет не учитывать состояния и трассы реализации, которые опасны в спецификации. 3) *Медиатор*: трассы в реализации и спецификации не обязательно совпадают, хотя и связаны

[2] $\boldsymbol{der}(I)\triangleq\{i\,|\,\exists\mu\ i\in(\mathbf{I}$ ***after*** $\mu)\}$.

медиаторным отображением. Это позволяет рассматривать реализацию и спецификацию в разных семантиках (в том числе, в разных алфавитах) при наличии соответствующего медиатора. Кроме того, в нашем случае соответствие состояний – это всегда отображение $V_I \to V_S$, то есть одному реализационному состоянию соответствует только одно спецификационное состояние.

До сих пор мы предполагали, что в тесте опрос спецификационного состояния $\mathcal{M}_0()=s$ необязателен: спецификационная трасса может как включать, так и не включать начальное состояние и состояние после того или иного наблюдения. Возможен вариант тестирования с жестким требованием опроса спецификационного состояния в начале тестирования и после каждого наблюдения. Если в безопасную трассу с необязательными состояниями добавить состояния в недостающих местах, то полученная трасса, очевидно, тоже безопасна. Отсюда легко показывается, что при переходе к тестированию с обязательным опросом состояния класс конформных реализаций сохраняется, а класс безопасных реализаций не сужается. Однако класс безопасных реализаций может расшириться, что показывается примером на Рис.1.

$\mathbf{R}=\mathbf{R}`=\{\{x\},\{y\}\}$, $\mathbf{S}$ – спецификация, $\mathbf{I}$ – реализация,
$\forall P\in\mathbf{R}\ \forall\sigma`\in\mathbf{N}^{`*}\ \mathbf{M}(P,\sigma`)=\{(\langle a\rangle,a)\mid a\in P\cup\{P\}\}$, $\forall i\in V_{\mathbf{I}}\ \mathbf{M}_0(i)=0$.

Рис.1. Опасная реализация становится безопасной при переходе к обязательному опросу спецификационного состояния.

5. Полное тестирование с открытым состоянием ограниченно недетерминированных систем

Когда семантики реализации и спецификации совпадают, полное тестирование с открытым состоянием удается закончить за конечное время, если выполнены следующие ограничения [6]. 1) *Ограничения на семантику*: число кнопок конечно и задан алгоритм разрешения кнопки относительно всех действий (в частности, все кнопки конечны). 2) *Ограничения на спецификацию*: LTS-спецификация конечна (конечно число достижимых состояний и переходов). 3) *Ограничения на реализацию*: реализация конечна, сильно-связна[3] и ограниченно

[3] Из каждого состояния, достижимого из начального состояния, достижимо начальное состояние.

недетерминирована. Ограниченность недетерминизма означает, что существует такое число t (степень недетерминизма), что в любом состоянии i реализации после t нажатий любой кнопки P будут получены все возможные пары (наблюдение $a \in P \cup \{P\}$, постсостояние $i`$). При $t=1$ реализация детерминирована.

При тестировании через медиатор алгоритм полного тестирования, описанный в [6], практически не меняется, кроме того, что тестовые воздействия и наблюдения выполняются через обращение к медиатору. Более существенные отличия касаются степени недетерминизма t.

Во-первых, спецификационной кнопке P соответствует адаптивная последовательность реализационных кнопок $P`_{i_1},\ldots,P`_{i_{n_i}}$, зависящая в общем случае как от реализационных наблюдений, так и от реализационных состояний. Поэтому, если $t`$ – это степень недетерминизма каждой реализационной кнопки $P`_{i_j}$, то степень недетерминизма t спецификационной кнопки P вычисляется как максимум из t^{n_i}. Число t, вообще говоря, зависит от кнопки P, причем эта зависимость известна лишь медиатору. Для того, чтобы использовать общую степень недетерминизма для всех спецификационных кнопок, пришлось бы брать максимум из t по всем кнопкам P.

Понятно, что при таком тестировании мы будем делать много лишних нажатий спецификационных кнопок. Вместо этого медиатор мог бы вычислять степень недетерминизма t отдельно для каждой спецификационной кнопки и каждого реализационного состояния, в котором она нажимается. Более того, для разных последовательностей $P`_{i_1},\ldots,P`_{i_{n_i}}$ число t могло бы быть разным, то есть зависящим от длины последовательности n_i. Вместо того, чтобы передавать тесту вычисленное значение числа t, медиатору достаточно просто сообщить, нужно или не нужно еще раз нажимать эту спецификационную кнопку в этом реализационном состоянии.

Во-вторых, степень недетерминизма зависит от семантики тестового взаимодействия. В примере с передачей сообщений через выходные порты реализация детерминирована в реализационной семантике, где каждому порту соответствует отдельная кнопка, если в каждом своем состоянии i она выдает не более одного сообщения в каждый порт j, а постсостояние однозначно определяется парой (i,j). Однако, если реализация выдает сообщения по нескольким портам, она недетерминирована в спецификационной семантике, где кнопка соответствует всем портам в совокупности. Чтобы получить все

возможные наблюдения, мы должны нажимать спецификационную кнопку приема сообщений n раз, если реализация выдает сообщения по $n>0$ портам, или 1 раз, если $n=0$.

Медиаторное отображение теперь имеет вид $\mathbf{M}:\mathbf{R}\times\mathbf{N}^{`*}\to\mathcal{P}(\mathbf{N}^{`*}\times\mathbf{N}\times\boldsymbol{Bool})$. Тест через медиатор опрашивает реализационное состояние $\sigma`/\mathcal{M}(P_0`)=(i,\boldsymbol{false})/\langle i\rangle$, после чего с нужной кнопкой P обращается к медиатору $\sigma`\cdot\langle i\rangle/\mathcal{M}(P)=(a,\mathrm{Press})/\lambda$ и во втором выходном параметре узнает, нужно (Press=***true***) или не нужно (Press=***false***) еще раз нажимать кнопку P в том же реализационном состоянии i.

Алгоритм тестирования в [6] строил LTS, состояниями которой являются достигнутые состояния реализации. С реализационным состоянием i связывалось семейство $\mathbf{S}(i)$ множеств вида $\mathbf{S}\ \boldsymbol{after}\ \sigma$, где $\sigma\in\boldsymbol{Safe}(\mathbf{S})$ и $i\in(\mathbf{I}\ \boldsymbol{after}\ \sigma)$. При тестировании через медиатор $i\in(\mathbf{I}\ \boldsymbol{after}\ \sigma`)$, где $\sigma`\in\mathbf{M}_T(\sigma)\cap\boldsymbol{T}(\mathbf{I})$. Эта связь используется в блоке «Распространение», который осуществляет верификацию не только наблюдений, полученные после *реальных* спецификационных трасс, пройденных при тестировании, но и возможных наблюдений после *потенциальных* трасс. Если после спецификационной трассы σ_1 мы попадаем в реализационное состояние i, в котором мы уже были после трассы σ_2, то есть $(\mathbf{S}\ \boldsymbol{after}\ \sigma_2)\in\mathbf{S}(i)$, то все наблюдения уже полученные в состоянии i после σ_2, могут быть в реализации и после трассы σ_1. Такие наблюдения достаточно верифицировать аналитически. 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.

Если медиатор осуществляет преобразование состояний $\mathbf{M}_0:V_\mathbf{I}\to V_\mathbf{S}$, то с каждым реализационным состоянием i связано ровно одно спецификационное состояние $\mathbf{M}_0(i)$. Поэтому блок «Распространение» больше не нужен. Это не означает, что вместо всех аналитических проверок нам теперь нужно выполнять столько же шагов реального тестирования. Различие в том, что конформность с преобразованием состояний предъявляет к реализации более жесткие требования: правильным должно быть не только наблюдение, но и постсостояние. Без преобразования состояний правильность постсостояний могла проверяться лишь косвенно: если постсостояние неправильное, то рано или поздно это скажется на неправильных наблюдениях в нем. Теперь мы делаем эту проверку напрямую.

Литература

1. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ. «Программирование». 2000. No. 2.
2. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Асинхронные автоматы: классификация и тестирование. Труды ИСП РАН, т. 4, 2003.
3. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, No. 5.
4. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.
5. 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
6. Бурдонов И.Б., Косачев А.С. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. Настоящий сборник.
7. Г.В.Ключников, А.С.Косачев, Н.В.Пакулин, А.К.Петренко, В.З.Шнитман. Применение формальных методов для тестирования реализации IPv6. Труды Института системного программирования РАН, N 4, 2003. стр.121-140.
8. van der Bijl M., Rensink A., Tretmans J. Compositional testing with ioco. Formal Approaches to Software Testing: Third International Workshop, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. Editors: Alexandre Petrenko, Andreas Ulrich ISBN: 3-540-20894-1. LNCS volume 2931, Springer, pp. 86-100.
9. 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.
10. Jard C., Jéron T., Tanguy L., Viho C. Remote testing can be as powerful as local testing. In Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX’ 99, Beijing, China, J. Wu, S. Chanson, Q. Gao (eds.), pp. 25-40, October 1999.
11. Lee D., Yannakakis M. Principles and Methods of Testing Finite State Machines – A Survey. Proceedings of the IEEE 84, No. 8, 1090–1123, 1996.
12. Milner R. Lectures on a calculus for communicating systems. Seminar on Concurrency, LNCS 197, Springer-Verlag, pp. 197-220.
13. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1 // ISO Interim Meeting / ITU-T on, Paris, 1995.

И.Б.Бурдонов, А.С.Косачев.
Тестирование с преобразованием семантик.
Труды Института системного программирования РАН, N 17, 2009, стр.193-208.
16 стр.
