Пополнение спецификации для *ioco*

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

Аннотация. Статья посвящена проблемам отношения ***ioco***, определяющего соответствие (конформность) реализации спецификации. Рассматриваются проблемы нерефлексивности отношения ***ioco***, наличие в спецификации неконформных (отсутствующих в любой конформной реализации) трасс, несохранение ***ioco*** при композиции (композиция реализаций, конформных своим спецификациям, может быть неконформна композиции этих спецификаций) и проблема «ложных» ошибок при тестировании в контексте. Для решения этих проблем в статье предлагается алгоритм пополнения спецификации, по отношению к которому ***ioco*** инвариантно (сохраняется класс конформных реализаций). На классе пополненных спецификаций указанные проблемы отсутствуют.

1. Введение

Правильность исследуемой системы понимается как её соответствие (конформность) заданным требованиям. Когда говорят о верификации конформности на основе формальных моделей, то предполагают, что система отображается в реализационную модель (реализацию), требования – в спецификационную модель (спецификацию), а их соответствие – в бинарное отношение конформности. Если требования к системе выражаются в терминах ее взаимодействия с внешним окружением, то соответствующую конформность можно проверять с помощью тестирования, то есть в процессе тестовых экспериментов над исследуемой системой.

Одной из наиболее известных конформностей является отношение ***ioco*** (**I**nput-**O**utput **CO**nformance), предложенное Яном Тритмансом [12]. Взаимодействие понимается как обмен дискретными порциями информации между реализацией и окружением. То, что передается из окружения в реализацию, называется *стимулом* (input), а то, что реализация передает в окружение, – *реакцией* (output). Предполагается, что в реализации не может возникнуть *отказ* от приема стимула (*блокировка* стимула), однако реализация может отказаться выдавать реакции, причем такой *отказ* составляет особый тип наблюдения за поведением реализации, обозначаемый символом δ. Результатом тестового эксперимента является трасса наблюдений – последовательность стимулов, реакций и отказов δ.

Отношение ***ioco*** определяется на моделях, которыми являются системы помеченных переходов (LTS – Labelled Transition System). LTS основана на

понятиях *состояния* и *перехода* между состояниями трех видов: переход по стимулу, переход по реакции и внутренний, ненаблюдаемый τ-переход.

Отношение ***ioco*** предполагает, что реализация всегда готова принять любой стимул (всюду определена по стимулам), тогда как на спецификацию такого ограничения не налагается. Это различие приводит к проблеме нерефлексивности отношения ***ioco***. Как следствие, в спецификации могут быть *неконформные трассы*, которых не может быть ни в какой конформной реализации.

Кроме этого, в ***ioco***-реализации и ***ioco***-спецификации не должно быть бесконечных цепочек τ-переходов (*дивергенции*). Это приводит к некоторым проблемам при композиции таких моделей, поскольку в результате композиции дивергенция может возникнуть.

С композицией связана и основная проблема отношения ***ioco*** – нарушение монотонности: несохранение конформности при композиции, когда композиция реализаций, конформных своим спецификациям, оказывается неконформной композиции этих спецификаций. Особая разновидность проблемы монотонности проявляется при тестировании в контексте, когда тест взаимодействует с реализацией не напрямую, а через некоторый контекст (например, очереди стимулов и/или реакций). Композиция конформной реализации с контекстом может оказаться неконформна композиции спецификации с этим контекстом. В результате тест ловит «ложные» ошибки, которых не бывает при непосредственном взаимодействии теста с реализацией.

Для решения указанных проблем обычно предлагается то или иное преобразование спецификации, называемое *пополнением* [3,6,7,8,10,13,14]. Пополненные спецификации всюду определены по стимулам так же, как реализации, и указанные выше проблемы отсутствуют или значительно ослабляются на классе таких спецификаций. Однако, существенный недостаток предлагавшихся до сих пор преобразований заключается в том, что по отношению к ним не инвариантно само отношение ***ioco***: в результате пополнения конформность может ослабляться (некоторые неконформные реализации становятся конформными) и/или усиливаться (некоторые конформные реализации становятся неконформными) [1, раздел 2.5.3].

Наилучшим можно считать, так называемое, *демоническое пополнение* (demonic completion) [13,14], при котором блокировка стимулов стимула заменяется на, так называемое, *хаотическое* поведение [4], допускающее любые трассы наблюдений. Это пополнение не усиливает конформность, но может ее ослабить. В [7] предлагается пополнение, которое лишено указанных недостатков, но, во-первых, оно определяется не для отношения ***ioco***, а для более простого отношения ***ioconf***, и, во-вторых, преобразуется не LTS, а

множество ее трасс, то есть не строится LTS с пополненным множеством трасс, что не дает возможности использовать такое пополнение при композиции LTS-моделей.

Предлагавшиеся ранее пополнения не решают проблему «ложных» ошибок при тестировании в контексте общего вида, в котором допускаются блокировки стимулов. Заметим, что такие контексты встречаются на практике. Например, ограниченные очереди стимулов и реакций можно трактовать как LTS с блокировкой стимулов (когда очередь полна).

В данной статье предлагаются два преобразования пополнения LTS-спецификаций, которые лишены указанных недостатков и применяются для ограниченных, но достаточно широких, классов спецификаций. Первое преобразование применяется тогда, когда не предполагается тестирование в контексте, который не всюду определен по стимулам. Второе, более сложное, преобразование налагает на спецификации более сильные ограничения, но может применяться во всех случаях.

Во 2-ом разделе статьи вводятся основные понятия и обозначения LTS, определяются отношение ***ioco***, композиция LTS и генерация тестов для ***ioco***. В 3-ем разделе более подробно рассматриваются проблематика отношения ***ioco***, приводятся примеры спецификаций и реализаций, на которых демонстрируются рассматриваемые проблемы. В разделе 4 определяется первое из предлагаемых пополнений LTS-спецификаций, которое в 5-ом разделе демонстрируется на примерах из раздела 3. В 6-ом разделе определяется второе преобразование для тестирования в контексте общего вида.

2. LTS-модель и отношение ioco

2.1. Система помеченных переходов (LTS)

В качестве модели реализации и спецификации используется LTS (Labelled Transition System), определяемая как совокупность S=LTS(V_S,L,E_S,s_0), где V_S – непустое множество состояний, L – алфавит внешних действий, $E_S \subseteq V_S \times (L \cup \{\tau\}) \times V_S$ – множество переходов, $s_0 \in V_S$ – начальное состояние. Класс всех LTS обозначим $\boldsymbol{LTS}$. Для $\boldsymbol{LTS}_\mathbf{i} \subseteq \boldsymbol{LTS}$ класс всех LTS в алфавите L, принадлежащих $\boldsymbol{LTS}_\mathbf{i}$, будем обозначать $\boldsymbol{LTS}_\mathbf{i}$ (L).

Одинарными стрелками обозначаются наличие/отсутствие переходов:

$s \xrightarrow{u} s' \triangleq (s,u,s') \in E_S$,

$s \xrightarrow{u} \not\rightarrow \triangleq \neg \exists s' \; s \xrightarrow{u} s'$.

Последовательность внешних действий называется трассой. Для данного состояния s говорят о трассах, начинающихся в этом состоянии, и о состояниях, в которых такие трассы заканчиваются. Формально для $z\in L$ и $\sigma=\langle z_1,\ldots,z_n\rangle\in L^*$

$s\Rightarrow s` \quad \triangleq s=s` \vee \exists s_1,\ldots,s_n\ s=s_1\text{—}\tau\rightarrow s_2\text{—}\tau\rightarrow\ldots\text{—}\tau\rightarrow s_n=s`,$

$s=\langle z\rangle\Rightarrow s` \triangleq \exists s_1,s_2\ s\Rightarrow s_1\text{—}z\rightarrow s_2\Rightarrow s`,$

$s=\sigma\Rightarrow s` \triangleq \exists s_0,\ldots,s_n\ s=s_0=\langle z_1\rangle\Rightarrow s_1\ldots s_{n-1}=\langle z_n\rangle\Rightarrow s_n=s`,$

$s=\sigma\Rightarrow \triangleq \exists s`\ s=\sigma\Rightarrow s`,$

$s=\sigma\not\Rightarrow \triangleq \neg(s=\sigma\Rightarrow),$

$s\ \textbf{\textit{after}}\ \sigma \triangleq \{s`|s=\sigma\Rightarrow s`\}.$

Состояние s *достижимо*, если в него можно попасть из начального состояния по некоторой трассе. Множество достижимых состояний LTS S будем обозначать $\textbf{\textit{der}}(S)\triangleq\{s\in V_S|\exists\sigma\ s_0=\sigma\Rightarrow s\}$. Состояние s *терминальное*, если из него не выходят никакие переходы: $\forall u\in L\cup\{\tau\}\ \ s\text{—}u\not\rightarrow$. Состояние s *стабильное*, если из него не выходят τ-переходы: $s\text{—}\tau\not\rightarrow$. Состояние s *дивергентное* (обозначается $s\uparrow$), если в нем начинается бесконечная цепочка τ-переходов: $s\uparrow\triangleq\exists s_1,s_2,\ldots\ s\text{—}\tau\rightarrow s_1\text{—}\tau\rightarrow s_2\text{—}\tau\rightarrow\ldots$. В противном случае состояние s *конвергентно*, что обозначается как $s\downarrow\triangleq\neg s\uparrow$. LTS *строго конвергентна*, если в ней все достижимые состояния конвергентны: $\forall s\in\textbf{\textit{der}}(S)\ s\downarrow$. Класс всех строго конвергентных LTS обозначим $\boldsymbol{LTS_1}$.

2.2. ioco-семантика взаимодействия

В семантике отношения ***ioco*** предполагается, что универсум внешних действий **L** разбит на универсум стимулов (input) **X** и универсум реакций (output) **Y**: $\mathbf{L}=\mathbf{X}\cup\mathbf{Y}$ & $\mathbf{X}\cap\mathbf{Y}=\varnothing$. Для алфавита LTS $L\subseteq\mathbf{L}$ обозначим $X=L\cap\mathbf{X}$ и $Y=L\cap\mathbf{Y}$. Тогда $L=X\cup Y$ & $X\cap Y=\varnothing$. Взаимодействие реализации в алфавите $L\subseteq\mathbf{L}$ с окружением сводится к последовательности актов взаимодействия двух типов: 1) посылка одного указанного стимула $x\in X$ из окружения в реализацию и 2) прием любой реакции $y\in Y$ из реализации в окружение. Предполагается, что в реализации не может возникнуть *отказ* (refusal) от приема стимула, называемый *блокировкой* стимула. В то же время реализация может отказаться выдавать реакции, причем этот *отказ* наблюдаем, и такое наблюдение обозначается символом δ.

Состояние s будем называть *стационарным* (*quiescent*) и обозначать $\delta(s)$, если оно стабильно и в нем не определены переходы по реакциям: $\delta(s)\triangleq\forall u\in Y\cup\{\tau\}\ s\text{—}u\not\rightarrow$. Отказ δ наблюдается в стационарных состояниях

реализации. Добавим в стационарных состояниях LTS S переходы-петли по символу δ: $s \text{—}\delta\rightarrow s \triangleq \delta(s)$. Тем самым, мы получим новую LTS $S\text{`}=\text{LTS}(V_S, L_\delta, E_{S\text{`}}, s_0)$, где $L_\delta = L \cup \{\delta\}$ и $E_{S\text{`}} = E_S \cup \{(s,\delta,s) | s \in V_S \,\&\, \delta(s)\}$. Будем называть S-трассой (*suspension trace*) LTS $S$ трассу LTS $S\text{`}$, и распространим на S-трассы обозначения с двойной стрелкой. Обозначим множество S-трасс LTS S: $\boldsymbol{Straces}(S) \triangleq \{\delta \in L_\delta^* | s_0 = \sigma\Rightarrow\}$.

LTS в алфавите $X \cup Y$ называется *всюду определенной по стимулам*, если в каждом ее достижимом стабильном состоянии определены переходы по всем стимулам: $\forall s \in \boldsymbol{der}(S)\ (s \text{—}\tau\not\rightarrow \Rightarrow \forall x \in \underline{X}\ s \text{—}x\rightarrow)$.

Спецификацией для ***ioco*** будем считать строго конвергентную LTS в алфавите $L \subseteq \mathbf{L}$. *Реализацией* для ***ioco*** будем считать строго конвергентную LTS в алфавите $L \subseteq \mathbf{L}$, которая всюду определена по стимулам. Класс таких LTS обозначим $\boldsymbol{LTS_2}$. Для строго конвергентной LTS всюду определенность по стимулам эквивалентна наличию в каждом достижимом состоянии трассы по каждому стимулу: $\forall s \in \boldsymbol{der}(S)\ \forall x \in X\ s = \langle x \rangle \Rightarrow$. Требование строгой конвергентности и всюду определенности по стимулам реализации не проверяется при тестировании, являясь его предусловием, то есть *гипотезой о реализации*.

2.3. Определение отношения ioco

Обозначим множество наблюдений, которые можно получить при приеме реакций от LTS S в алфавите $X \cup Y$, находящейся в состоянии $a \in V_S$ или в состоянии из множества $A \subseteq V_S$:

$\boldsymbol{out}(a) \triangleq \{z \in Y_\delta | a = \langle z \rangle \Rightarrow\}$, где $Y_\delta = Y \cup \{\delta\}$,

$\boldsymbol{out}(A) \triangleq \cup\{\boldsymbol{out}(a) | a \in A\}$.

Отношение ***ioco*** требует, чтобы при приеме реакций от реализации I могли быть получены только такие наблюдения, которые возможны в спецификации S в той же самой ситуации, то есть после наблюдения той же самой S-трассы. Формально: $\forall L \subseteq \mathbf{L}\ \forall I \in \boldsymbol{LTS_2}(L)\ \forall S \in \boldsymbol{LTS_1}(L)$

I ***ioco*** $S \triangleq \forall \sigma \in \boldsymbol{Straces}(S)\ \boldsymbol{out}(i_0\ \boldsymbol{after}\ \sigma) \subseteq \boldsymbol{out}(s_0\ \boldsymbol{after}\ \sigma)$.

2.4. Параллельная композиция LTS

Определим на универсуме внешних действий **L** инволюцию "_", ставящую в соответствие каждому внешнему действию отличное от него *противоположное* действие: стимулу – реакцию, реакции – стимул:

$\forall x\in\mathbf{X}\ \underline{x}\in\mathbf{Y}\ \&\ \forall y\in\mathbf{Y}\ \underline{y}\in\mathbf{X}\ \&\ \forall z\in\mathbf{L}\ \underline{z}=z$. Распространим инволюцию "_" на множество A внешних действий: $\underline{A}\triangleq\{\underline{a}|a\in A\}$.

Параллельное выполнение двух взаимодействующих LTS, заданных в алфавитах $A\subseteq\mathbf{L}$ и $B\subseteq\mathbf{L}$, понимается так, что переходы по противоположным действиям z и $\underline{z}$, где $z\in A$ и $\underline{z}\in B$, выполняются синхронно, то есть, в обеих LTS одновременно, причём в композиции это становится τ-переходом. Остальные переходы выполняются асинхронно, то есть, в одной из LTS при сохранении состояния другой LTS. Это соответствует определению параллельной композиции в CCS – Calculus of Communicating Systems [9].

Для алфавитов $A\subseteq\mathbf{L}$ и $B\subseteq\mathbf{L}$ определим композицию $|\!|$ алфавитов следующим образом: $A|\!|B\triangleq(A\setminus\underline{B})\cup(B\setminus\underline{A})$. Композиция двух LTS I=LTS(V_I,A,E_I,i_0) и T=LTS(V_T,B,E_T,t_0) определяется как LTS S=LTS($V_I\times V_T$,$A|\!|B$,E_S,i_0t_0), где E_S – наименьшее множество, порождаемое следующими правилами вывода:
$\forall z\ \forall i\in V_I\ \forall t\in V_T$

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

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

3) $z\in A\cap\underline{B}\quad\&\ i\text{—}z\to i`\ \&\ t\text{—}\underline{z}\to t`\quad\vdash it\text{—}\tau\to i`t`$.

Заметим, что при параллельной композиции в CSP – Communicating Sequential Processes [5] синхронный переход соответствует паре переходов в LTS-операндах по одному и тому же символу. Поэтому после композиции требуется дополнительное применение оператора ***Hide*** [13], который превращает такие композиционные переходы в τ-переходы.

2.5. Тестирование и генерация тестов для *ioco*

Тестирование понимается как параллельное выполнение LTS-реализации в алфавите $L\subseteq\mathbf{L}$ и LTS-теста. Тест подменяет собой окружение реализации, поэтому он определяется в противоположном алфавите $\underline{L}$. Композиция реализации и теста содержит только τ-переходы, и параллельное выполнение реализации и теста – это выполнение цепочки τ-переходов в их композиции. Для наблюдения тестом отказа δ, возникающего в реализации, в LTS-тесте вводится специальный θ-переход, который предназначен для разрешения тупиков: в композиции реализации и теста он выполняется как τ-переход тогда и только тогда, когда другие переходы невозможны. Символ θ добавляется в алфавит теста. В определении композиции LTS-реализации I=LTS(V_I,L,E_I,i_0) и LTS-теста T=LTS(V_T,$\underline{L}\cup\{\theta\}$,$E_T$,$t_0$) добавляется четвертое правило: $\forall i\in V_I\ \forall t\in V_T$

4) $i\text{—}\tau\not\to\ \&\ t\text{—}\tau\not\to\ \&\ (\forall z\in L\ i\text{—}z\not\to\ \vee\ t\text{—}\underline{z}\not\to)\ \&\ t\text{—}\theta\to t`\quad\vdash it\text{—}\tau\to it`$.

В LTS-тесте должны быть только два терминальных состояния, соответствующие вердиктам ***pass*** и ***fail***. Тестирование заканчивается тогда, когда композиция реализации и теста попадает в терминальное состояние *it*. Если состояние *t* теста терминальное (выносится вердикт), то композиционное состояние *it* тоже терминальное. Для того, чтобы было верно и обратное, то есть чтобы тестирование всегда заканчивалось вынесением вердикта, нужно, чтобы для каждого терминального состояния *it* композиции состояние *t* теста было терминальным. При композиции тупик не возникает, если состояние теста нестабильно или в нем определен хотя бы один переход по посылке стимула, то есть по символу $\underline{x}$, где $x \in X$, поскольку реализация всюду определена по стимулам. В ***ioco***-семантике, кроме внешних действий, может наблюдаться единственный отказ δ. Поэтому для того, чтобы тест мог применяться к любой реализации в алфавите *L*, мы должны потребовать, чтобы в любом достижимом нетерминальном стабильном состоянии *t* теста, в котором нет переходов по посылке стимулов, были определены переходы по приему от реализации всех реакций и θ-переход:

$\forall t \in \boldsymbol{der}(T)\ \forall u \in \underline{L} \cup \{\tau,\theta\}\ t \text{—} u \not\rightarrow \vee\ \exists x \in \underline{X} \cup \{\tau\}\ t \text{—} x \rightarrow \vee\ \forall y \in \underline{Y} \cup \{\theta\}\ t \text{—} y \rightarrow$.

Для того, чтобы тестирование всегда заканчивалось за конечное время, нужно, чтобы в композиции реализации и теста не было бесконечных цепочек переходов. Для того, чтобы это выполнялось для любой реализации в алфавите *L*, в тесте все цепочки переходов должны быть конечны (в частности, тест должен быть строго конвергентным). Иными словами граф LTS-теста является конечным ациклическим графом, стоками которого являются состояния, соответствующие вердиктам ***pass*** и ***fail***.

При указанных ограничениях на тест, реализация *проходит* тест, если в композиции реализации и теста для любого достижимого терминального состояния *it* состоянию *t* приписан вердикт *pass*. Реализация проходит набор тестов, если она проходит каждый тест из набора. Набор тестов *значимый* (*sound*), если каждая конформная реализация его проходит. Набор тестов *исчерпывающий* (*exhaustive*), если каждая неконформная реализация его не проходит. Набор тестов полный, если он значимый и исчерпывающий.

Обычно на тесты дополнительно налагается ограничение отсутствия «лишнего недетерминизма»: в каждом достижимом нетерминальном состоянии *t* теста либо посылается один стимул: $\exists x \in \underline{X}\ t \text{—} x \rightarrow\ \&\ \forall u \neq x\ t \text{—} u \not\rightarrow$, либо принимаются все реакции и обнаруживается отказ δ: $\forall y \in \underline{Y} \cup \{\theta\}\ t \text{—} y \rightarrow\ \&\ \forall u \in \underline{X} \cup \{\tau\}\ t \text{—} u \not\rightarrow$. Эти ограничения не уменьшает мощности тестирования, то есть при этих ограничениях существует полный набор тестов.

В частности, полным является набор всех *примитивных* тестов. Такой тест строится по одной непустой S-трассе σ спецификации *S*. Состояниями LTS теста являются префиксы S-трассы σ, отличные от нее самой, и некоторые их продолжения одним символом, начальное состояние – пустая S-трасса, а переходы определяются следующим правилами вывода:

$\forall i$=1..*n*, где *n* – длина S-трассы σ, $\forall z \in Y_\delta$

$i<n$ $\quad\vdash \sigma[1..i\text{-}1] \text{—} \underline{\sigma(i)} \rightarrow \sigma[1..i]$,

$\sigma(i) \in Y_\delta \;\&\; \sigma[1..i]\cdot\langle z\rangle \in \textbf{\textit{Straces}}(S) \;\&\; (z \neq \sigma(i) \vee i=n) \quad\vdash \sigma[1..i\text{-}1] \text{—} \underline{z} \rightarrow \textbf{\textit{pass}}$,

$\sigma(i) \in Y_\delta \;\&\; \sigma[1..i]\cdot\langle z\rangle \notin \textbf{\textit{Straces}}(S) \quad\vdash \sigma[1..i\text{-}1] \text{—} \underline{z} \rightarrow \textbf{\textit{fail}}$,

где σ[1..*j*] - префикс S-трассы σ, содержащий первые *j* символов, а $\underline{\delta} \triangleq \theta$.

3. Проблемы отношения *ioco*

С отношением ***ioco*** связаны четыре основные проблемы: 1) нерефлексивность конформности, 2) неконформные S-трассы спецификации, 3) немонотонность конформности, под которой понимается несохранение конформности при композиции, и 4) несохранение конформности при тестировании в контексте.

3.1. Нерефлексивность *ioco*

Поскольку реализация должна быть всюду определенной по стимулам, а спецификация в некоторых состояниях может блокировать (не принимать) некоторые стимулы, такая спецификация по определению неконформна сама себе по отношению ***ioco***. Нерефлексивность означает, что спецификация не может быть понята как конформная ей самой реализация. Мы считаем это недостатком, поскольку это противоречит интуитивному пониманию спецификации как набору требований, которые можно реализовать «как есть». Такую спецификацию нельзя использовать как прототип реализации: если реализацию написать как эксплицитное выражение спецификации, то мы получим неконформную реализацию.

3.2. Неконформные S-трассы спецификации

Конформной S-трассой (для спецификации *S*) будем называть S-трассу, которая имеется хотя бы в одной конформной реализации. В проблеме нерефлексивности выделяется особый случай, когда сама спецификация *S* содержит неконформные S-трассы.

Рассмотрим пример двух спецификаций S_1 и S_2 на Рис.1. В спецификации S_1 стимул блокируется в состоянии 4 после S-трассы $\langle\delta,x,\delta\rangle$. Поскольку реализации всюду определены по стимулам, в любой реализации, в которой есть S-трасса $\langle\delta,x,\delta\rangle$, эта S-трасса должна продолжаться стимулом *x*. Если в

реализации есть S-трасса ⟨δ,*x*,δ,*x*⟩, то в ней есть и S-трассы ⟨δ,*x*,*x*⟩, ⟨*x*,δ,*x*⟩ и ⟨*x*,*x*⟩. Эти три S-трассы есть в спецификации, и каждая из них продолжается в спецификации только реакцией *a*. Более того, все эти три S-трассы во множестве S-трасс спецификации продолжаются одними и теми же S-трассами (в состоянии 7). Для того, чтобы получить LTS-пополнение S_1`, которое эквивалентно и конформно спецификации S_1, достаточно добавить в спецификацию переход 4—*x*→7.

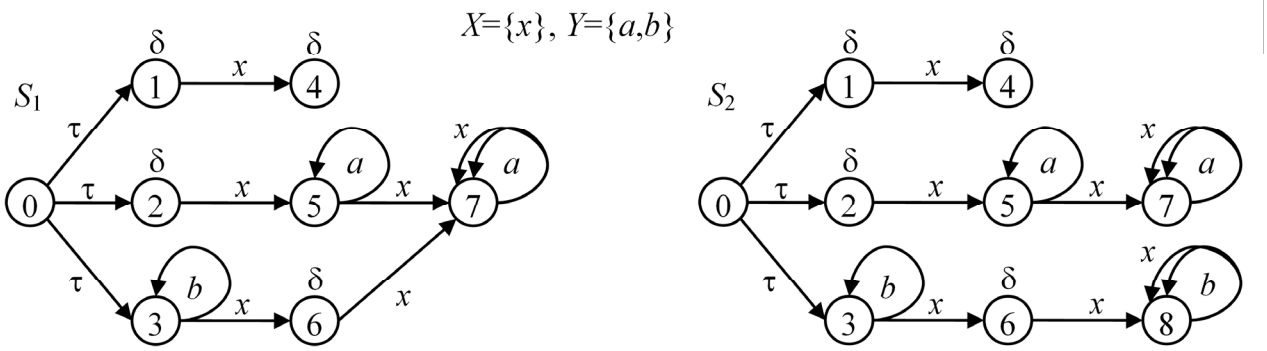


Рис.1. S-трасса ⟨δ,*x*,δ⟩ конформна слева и неконформна справа

Аналогично, для спецификации S_2 реализация, в которой есть S-трасса ⟨δ,*x*,δ⟩, содержит и S-трассы ⟨δ,*x*,δ,*x*⟩, ⟨δ,*x*,*x*⟩, ⟨*x*,δ,*x*⟩ и ⟨*x*,*x*⟩. Последние три S-трассы также имеются в спецификации S_2 и продолжаются в ней реакциями, соответственно, только *a*, только *b*, или обеими реакциями *a* и *b*. Тем самым, в отличие от спецификации S_1 после этих трех S-трасс нет общего наблюдения при приеме реакций тестом (общей реакции или δ). Следовательно, S-трасса ⟨δ,*x*,δ⟩ не может встречаться в конформной реализации. Для того, чтобы получить LTS-пополнение, которое эквивалентно и конформно спецификации S_2, нам нужно, в отличие от спецификации S_1, не добавлять S-трассы, а удалить неконформную S-трассу ⟨δ,*x*,δ⟩, например, удалив переход 0—τ→1.

Наличие «лишних» неконформных S-трасс в спецификации неприятно тем, что они порождают лишние тесты (в частности, по ним генерируются примитивные тесты). Рассмотрим спецификации S_3 и S_4 на Рис.2, которые являются модификацией спецификаций S_1 и S_2 на Рис.1: для того, чтобы в состоянии 5 не было отказа δ, вместо перехода по действию *a* используется τ-переход. В спецификации S_3 как и в спецификации S_1 все S-трассы конформны и для пополнения спецификации достаточно добавить переход 4—*x*→7. В спецификации S_4 как и в спецификации S_2 неконформна S-трасса ⟨δ,*x*,δ⟩. Но в отличие от S_2 конформность требует, чтобы после S-трассы ⟨δ,*x*⟩ не было также

реакций a и b. Следовательно, в конформной реализации не может быть S-трассы $\langle\delta,x\rangle$. Если бы в конформной реализации была S-трасса $\langle\delta\rangle$, то, поскольку реализация всюду определена по стимулам, в ней была бы и неконформная S-трасса $\langle\delta,x\rangle$. Следовательно, в конформной реализации нет S-трассы $\langle\delta\rangle$. Для пополнения спецификации S_4 нужно удалить неконформную S-трассу $\langle\delta\rangle$, например, удалив переходы $0\text{—}\tau\rightarrow 1$ и $0\text{—}\tau\rightarrow 2$.

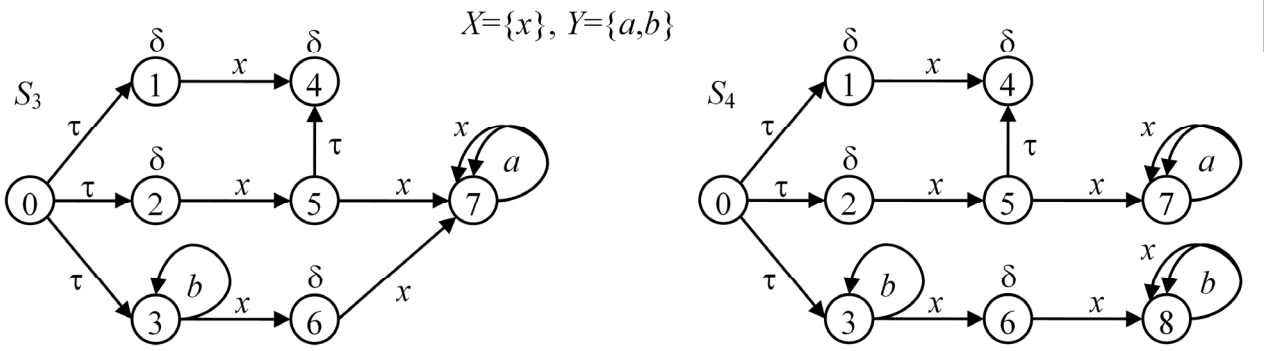


Рис.2. S-трасса $\langle\delta\rangle$ и ее продолжения конформны слева и неконформны справа

Еще более удивителен пример на Рис.3. Спецификации S_5 и S_6 являются дальнейшей модификацией спецификаций S_3 и S_4 на Рис.2: для того, чтобы в состоянии 3 не было отказа δ, вместо перехода по действию b используются τ-переход. В спецификации S_5 как и в спецификации S_3 все S-трассы конформны и для пополнения спецификации достаточно добавить переход $4\text{—}x\rightarrow 7$. В спецификации S_6 как и в спецификации S_4 неконформна S-трасса $\langle\delta\rangle$. Но в отличие от S_4 конформность требует, чтобы после пустой S-трассы не было также реакций a и b. Следовательно, в конформной реализации не может быть пустой S-трассы. Поскольку пустая S-трасса есть в любой реализации, для спецификации S_6 нет конформных реализаций. Все S-трассы спецификации S_6 неконформны.

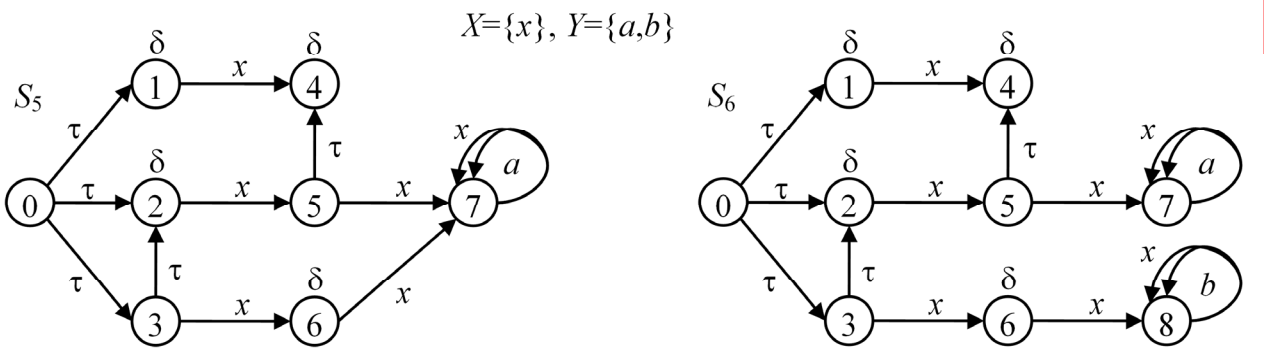


Рис.3. Все S-трассы конформны слева и все S-трассы неконформны справа

Эти примеры показывают, насколько полезным может оказаться анализ неконформных S-трасс. Полный набор примитивных тестов для любой из спецификаций $S_1 \div S_6$ бесконечен, поскольку бесконечно число S-трасс (за счет цикла в состоянии 7). Тем самым полное тестирование, вообще говоря, бесконечно. Для спецификации S_4 после получения S-трассы $\langle\delta\rangle$ можно сразу закончить тестирование с вердиктом ***fail***. Тем самым, не нужно генерировать тесты по всем имеющимся в спецификации продолжениям S-трассы $\langle\delta\rangle$, число которых бесконечно, то есть мы удаляем из полного набора тестов бесконечное число тестов. Для спецификации S_6 тестирование вообще излишне.

3.3. Немонотонность *ioco*

Будем говорить, что конформность *монотонна* относительно композиции LTS, если композиция реализаций, конформных своим спецификациям, конформна композиции этих спецификаций. Монотонность – важное свойство: если конформность монотонна, для проверки «правильности» составной, композиционной системы нам достаточно верифицировать конформность компонентов системы их спецификациям. Тогда система оказывается конформной композиции спецификаций компонентов, которую можно рассматривать как спецификацию системы. Если же спецификация системы задана, то мы можем решать задачу верификации декомпозиции системных требований, то есть проверять согласованность спецификации системы и спецификаций ее компонентов. Для этого достаточно проверить, что композиция спецификаций компонентов конформна имеющейся спецификации системы. Композиция спецификаций компонентов оказывается самой сильной (то есть предъявляющей максимальные требования) спецификацией системы среди всех спецификаций, согласованных со спецификациями компонентов [1,2].

К сожалению, отношение ***ioco*** не является монотонным [1,2,13,14]. Пример приведен на Рис.4. Здесь в композиции конформных реализаций $I_A \rceil\lfloor I_B$ с самого начала (после пустой S-трассы) есть наблюдение δ, а в композиции спецификаций $S_A \rceil\lfloor S_B$ это не так.

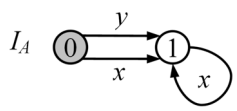
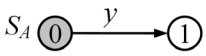
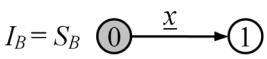
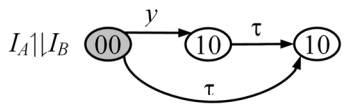
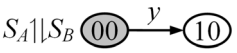
алфавиты	реализации	спецификации	конформность
$A=X_A\cup Y_A$ $X_A=\{x\}$ $Y_A=\{y\}$			I_A ***ioco*** S_A
$B=X_B\cup Y_B$ $X_B=\varnothing$ $Y_B=\{\underline{x}\}$			I_B ***ioco*** S_B
$X_{A\|\|B}=\varnothing$ $Y_{A\|\|B}=\{y\}$			$I_A\|\|I_B$ ~~***ioco***~~ $S_A\|\|S_B$

Рис.4. Немонотонность отношения ***ioco***

3.4. Несохранение *ioco* при тестировании в контексте

Особым случаем проблемы несохранения отношения ***ioco*** при композиции является тестирование в контексте [11,13,14], когда взаимодействие теста и реализации происходит не непосредственно, а через ту или иную промежуточную среду взаимодействия (*контекст*). Тестирование в контексте можно рассматривать как тестирование системы из двух компонентов, один из которых – реализация, а другой – фиксированный контекст. В отличие от общего случая композиции предполагается, что контекст известен, в нем нет ошибок и его не нужно тестировать. Проблема несохранения конформности заключается в том, что при тестировании в контексте тесты могут ловиться «ложные» ошибки, то есть те, которые не обнаруживаются, когда тест взаимодействует с реализацией непосредственно, без контекста.

Пример приведен на Рис.5 для реализации I и спецификации S (в алфавите A), и контекста Q_x (в алфавите B), представляющего собой неограниченную очередь стимулов. Поскольку очередь буферизует стимулы, посылаемые в реализацию, появляется возможность при тестировании по спецификации S послать два стимула x подряд (композиция спецификации с контекстом всюду определена по стимулам). Если после этого принимать реакции, то, согласно спецификации S, должны последовательно наблюдаться две реакции y. В то же время, хотя реализация I конформна, при тестировании в контексте будет обнаружена «ложная» ошибка: реализация может сначала принять два стимула x, а потом выдать только одну реакцию y, после которой наблюдается δ.

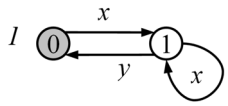
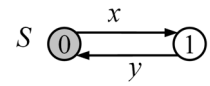
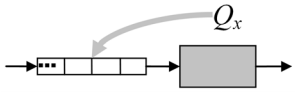
алфавиты	реализации	спецификации	конформность
$A=X_A\cup Y_A$, $X_A=\{x\}$, $Y_A=\{y\}$			I ***ioco*** S
$B=X_B\cup Y_B$, $X_B=\{x\}$, $Y_B=\{\underline{x}\}$ $X_{A\|B}=\{x\}$, $Y_{A\|B}=\{y\}$			$I\|\downarrow Q_x$ ~~***ioco***~~ $S\|\downarrow Q_x$

Рис.5. Несохранение ***ioco*** при тестировании в контексте

4. Пополнение спецификации

Указанные в предыдущем разделе четыре проблемы отношения ***ioco*** можно искать, опираясь на эквивалентность спецификаций. Две спецификации S и $S`$ *эквивалентны*, если они определяют один и тот же класс конформных реализаций: $S \sim_{\boldsymbol{ioco}} S` \triangleq \{I | I \;\boldsymbol{ioco}\; S\} = \{I | I \;\boldsymbol{ioco}\; S`\}$.

Пополнением будем называть такое преобразование $\boldsymbol{C}$, которое для каждой спецификации S строит эквивалентную ей спецификацию $\boldsymbol{C}(S)$ для которой эти четыре проблемы отношения ***ioco*** не возникают.

Такие преобразования были предложены авторами статьи для отношения конформности $\boldsymbol{ioco}_{\beta\gamma\delta}$ в [1] и более широкого класса конформностей ***saco*** в [2]. Здесь мы оптимизируем эти преобразования для отношения ***ioco***, в частности для случая, когда спецификация конечна (конечно число переходов) и преобразование можно выполнить алгоритмически за конечное время. Отношение $\boldsymbol{ioco}_{\beta\gamma\delta}$ допускает наблюдение не только отсутствия реакций δ, но и блокировок стимулов. Поэтому домен $\boldsymbol{ioco}_{\beta\gamma\delta}$ шире домена ***ioco***. Однако, на классе $\boldsymbol{LTS_2}$ всюду определенных по стимулам спецификаций эти отношения совпадают [1, 2.5.3, Определение отношения ***ioco***]. Поэтому результаты, полученные в [1], распространяются на отношение ***ioco***, если спецификации ограничиваются классом $\boldsymbol{LTS_2}$.

Проблемы нерефлексивности ***ioco*** и неконформных S-трасс спецификации. На классе $\boldsymbol{LTS_2}$ отношение $\boldsymbol{ioco}_{\beta\gamma\delta}$ эквивалентно вложенности S-трасс: $\forall L \subseteq \mathbf{L}$ $\forall I,S \in \boldsymbol{LTS_2}(L)$ $I \;\boldsymbol{ioco}_{\beta\gamma\delta}\; S = \boldsymbol{Straces}(I) \subseteq \boldsymbol{Straces}(S)$ [1, 2.2.7, Утв. 24]. Тем самым, на классе спецификаций $\boldsymbol{LTS_2}$ отношение ***ioco*** также эквивалентно вложенности S-трасс [см. также 14, Lemma A.4]. Отсюда непосредственно следует, что на классе спецификаций $\boldsymbol{LTS_2}$ отношение ***ioco*** рефлексивно. Как следствие, все S-трассы таких спецификаций конформны. Поэтому имеет смысл искать пополнение спецификации в классе $\boldsymbol{LTS_2}$.

Проблема немонотонности ***ioco***. Отношение ***ioco*** предполагает строгую конвергентность спецификации (класс $\boldsymbol{LTS_1}$) и для реализаций дополнительно всюду определенность по стимулам (класс $\boldsymbol{LTS_2}$).

Композиция всюду определенных по стимулам LTS также является всюду определенной по стимулам. Действительно, рассмотрим композицию $I|\!\rfloor T$, где $I \in \boldsymbol{LTS_2}(A)$ и $T \in \boldsymbol{LTS_2}(B)$. По правилам композиции стабильность композиционного состояния it влечет стабильность состояний i и t в LTS-операндах I и T, соответственно. Поэтому для каждого стимула $x \in X_A \backslash \underline{Y_B}$ всюду определенность по стимулам LTS I влечет $i \text{—} x \rightarrow$, что влечет $it \text{—} x \rightarrow$. Аналогично для каждого стимула $x \in X_B \backslash \underline{Y_A}$ всюду определенность по стимулам LTS T влечет $t \text{—} x \rightarrow$, что влечет $it \text{—} x \rightarrow$.

Однако композиция строго конвергентных LTS может не быть строго конвергентной. Например, пусть $I \in \boldsymbol{LTS_2}(A)$, $T \in \boldsymbol{LTS_2}(B)$ и для $z \in X_A \cap \underline{Y_B}$ имеются переходы-петли $i \text{—} z \rightarrow i$ в I и $t \text{—} \underline{z} \rightarrow t$ в T. Тогда в композиции $I|\!\rfloor T$ будет τ-переход-петля $it \text{—} \tau \rightarrow it$, то есть состояние it дивергентно, и, если оно достижимо, композиция не строго конвергентна.

Поэтому монотонность конформности приходится понимать в условном смысле: если композиция спецификаций $S_1|\!\rfloor S_2$ и композиция ***ioco***-конформных им реализаций $I_1|\!\rfloor I_2$ строго конвергентны, то должно быть $I_1|\!\rfloor I_2$ ***ioco*** $S_1|\!\rfloor S_2$. Если это свойство выполнено на некотором классе спецификаций, то это свойство будем называть *условной монотонностью* отношения ***ioco*** на этом классе спецификаций. В [14, Theorem 9] доказана такая условная монотонность ***ioco*** на классе спецификаций $\boldsymbol{LTS_2}$.[1]

Однако, если строгую конвергентность композиции спецификаций можно проверить, поскольку спецификации явно заданы, то LTS-модели реализаций могут быть неизвестны, и проверка строгой конвергентности композиции реализаций является проблемой. Поэтому важной является проблема определения таких дополнительных ограничений на спецификации, при выполнении которых строгая конвергентность композиции спецификаций влечет строгую конвергентность композиции реализаций; эти ограничения определяют некоторый подкласс класса $\boldsymbol{LTS_2}$. В [14] эта проблема не решается. *Безусловной монотонностью* отношения ***ioco*** на таком подклассе спецификаций будем называть следующее свойство: если композиция

[1] В [14] используется не CCS-композиция LTS, а CSP-композиция с оператором ***Hide***.

спецификаций $S_1 \uparrow\downarrow S_2$ из данного подкласса строго конвергентна, то композиция ***ioco***-конформных им реализаций $I_1 \uparrow\downarrow I_2$ также строго конвергентна и $I_1 \uparrow\downarrow I_2$ ***ioco*** $S_1 \uparrow\downarrow S_2$.

Мы покажем, что в качестве такого подкласса спецификаций достаточно взять подкласс класса $\boldsymbol{LTS_2}$, состоящий из локально конечно ветвящихся LTS [1,2]: это такие LTS, в каждом достижимом состоянии которых определено конечное число переходов по каждому символу, включая τ. Класс таких LTS обозначим $\boldsymbol{LTS_3}$. В [1, 4.3.1, Утв. 116] доказано, что на классе спецификаций $\boldsymbol{LTS_3}$ строгая конвергентность композиции спецификаций $S_1 \uparrow\downarrow S_2$ влечет строгую конвергентность композиции $\boldsymbol{ioco}_{\beta\gamma\delta}$-конформных им реализаций $I_1 \uparrow\downarrow I_2$, и монотонность отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$: $I_1 \uparrow\downarrow I_2$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S_1 \uparrow\downarrow S_2$. Поскольку $\boldsymbol{LTS_3 \subseteq LTS_2}$, это утверждение верно и для отношения ***ioco***.

Проблема несохранения ***ioco*** при тестировании в контексте. Аналогичные результаты получаются, когда контекст Q относится к классу реализаций $\boldsymbol{LTS_2}$.

Условное сохранение ***ioco***. На классе спецификаций $\boldsymbol{LTS_2}$ при условии строгой конвергентности как композиции спецификации с контекстом, так и композиции ***ioco***-конформной реализации с контекстом, отношение ***ioco*** сохраняется при тестировании в контексте. Действительно, на классе спецификаций $\boldsymbol{LTS_2}$ по [14, Lemma A.4] ***ioco*** эквивалентно вложенности S-трасс и, следовательно, рефлексивно: Q ***ioco*** Q. А тогда по [14, Theorem 9] $I \uparrow\downarrow Q$ ***ioco*** $S \uparrow\downarrow Q$.

Безусловное сохранение ***ioco***. В [1, 4.3.1, Утв. 116] доказано, что на классе спецификаций $\boldsymbol{LTS_3}$ строгая конвергентность композиции спецификации с контекстом $S \uparrow\downarrow Q$ влечет строгую конвергентность композиции $\boldsymbol{ioco}_{\beta\gamma\delta}$-конформной реализации с контекстом $I \uparrow\downarrow Q$, и сохранение отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ при тестировании в контексте: $I \uparrow\downarrow Q$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S \uparrow\downarrow Q$. Поскольку $\boldsymbol{LTS_3 \subseteq LTS_2}$, это утверждение верно и для отношения ***ioco***.

Решение проблемы несохранения ***ioco*** при тестировании в контексте общего вида (из класса ***LTS***, а не из $\boldsymbol{LTS_2}$) мы рассмотрим в разделе **Ошибка! Источник ссылки не найден.**.

В следующих подразделах мы определим преобразование $\boldsymbol{C}$:$\boldsymbol{LTS_1 \rightarrow LTS_3}$ и покажем, что для конечной спецификации пополнение может быть выполнено алгоритмически за конечное время.

При пополнении мы будем прежде всего избавляться от блокировок стимулов в спецификации. Если в S имеются S-трассы, которые не продолжаются некоторыми стимулами, то в $\boldsymbol{C}(S)$ такого быть не должно. Однако не все такие S-трассы сохраняются при пополнении (с добавлением после них недостающих стимулов), то есть не обязательно $\boldsymbol{Straces}(S) \subseteq \boldsymbol{Straces}(\boldsymbol{C}(S))$. Если спецификация S содержит неконформные S-трассы, они удаляются, естественно, со всеми своими продолжениями. Мы определим пополнение $\boldsymbol{C}$ как последовательное выполнение двух преобразований. Сначала избавляемся от блокировок стимулов (преобразование $\boldsymbol{B}$), а потом – от неконформных S-трасс (преобразование $\boldsymbol{D}$).

4.1. Преобразование *B*

Чтобы избавиться от блокировок стимулов, нам придется добавить «новые» S-трассы как продолжения «старых» (имевшихся в исходной спецификации) S-трасс теми стимулами, которыми они не продолжались в исходной спецификации. Эти новые трассы также нужно продолжить стимулами, и так далее. Одновременно каждая добавляемая трасса должна продолжаться реакциями и/или стационарностью, но так, чтобы это не приводило к изменению конформности. Если это сделать невозможно, трасса объявляется неконформной.

Если добавляется такая S-трасса σ, из которой можно получить «старые» S-трассы с помощью удаления некоторых вхождений отказа δ, то именно по этим «старым» трассам определяются требования, предъявляемые спецификацией к реализации. Если таких «старых» трасс нет, спецификация не предъявляет к реализации никаких требований после трассы σ.

Определим формально множество $\boldsymbol{d}(\sigma)$ S-трасс, получаемых из σ удалением символов δ, как наименьшее множество S-трасс, порождаемых следующими правилами вывода: $\forall \sigma,\mu,\lambda$

$$\vdash \sigma \in \boldsymbol{d}(\sigma),$$

$$\mu\cdot\langle\delta\rangle\cdot\lambda \in \boldsymbol{d}(\sigma) \ \vdash \mu\cdot\lambda \in \boldsymbol{d}(\sigma).$$

В [1] пополненная LTS-спецификация $\boldsymbol{B}(S)$ определяется таким образом, что ее состояния – это S-трассы (как «старые», так и «новые»), а переходы имеют вид $\sigma \text{—} z \rightarrow \sigma\cdot\langle z\rangle$, где $z \in L$, и $\sigma \text{—} \tau \rightarrow \sigma\cdot\langle\delta\rangle$. Однако число таких S-трасс, вообще говоря, бесконечно, даже если мы ограничимся (из практических соображений) конечными LTS-спецификациями (с конечным числом переходов). С другой стороны, при конструировании состояний LTS $\boldsymbol{B}(S)$ мы можем не различать S-трассы, заканчивающиеся в S в одном множестве состояний, поскольку они имеют одинаковые продолжения. Поэтому состояние LTS $\boldsymbol{B}(S)$ может

соответствовать не S-трассе σ, а множеству s_0 ***after*** σ состояний LTS *S*. Однако так можно делать только для «старых» S-трасс σ∈***Straces***(*S*). Мы используем другой подход, который годится как для «старых», так и для «новых» S-трасс, определяя в качестве состояния LTS ***B***(*S*) семейство ***P***(σ) множеств состояний LTS *S* после всех «старых» трасс из ***d***(σ). Состояние ***P***(σ) будет одним из состояний ***B***(*S*), в которых заканчивается S-трасса σ. Формально: $\boldsymbol{P}(\sigma) \triangleq \{s_0 \textbf{ after } \sigma` \neq \varnothing \mid \sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\}$.

Заметим, что все множества состояний, входящие в семейство ***P***(σ), не пусты (как множества состояний после S-трасс, имеющихся в LTS): $\forall p \in \boldsymbol{P}(\sigma)\ p \neq \varnothing$, в частности, $\boldsymbol{P}(\sigma) \neq \{\varnothing\}$. Само семейство ***P***(σ) не пусто тогда и только тогда, когда $\boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S) \neq \varnothing$. Пустое семейство ***P***(σ) соответствует всем таким «новым» S-трассам σ, после которых спецификация не предъявляет к реализации никаких требований.

Сначала рассмотрим случай $\boldsymbol{P}(\sigma) \neq \varnothing$, то есть $\boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S) \neq \varnothing$.

S-трасса σ продолжается в ***B***(*S*) реакцией $y \in Y$ только тогда, когда каждая «старая» S-трасса $\sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)$ также продолжается этой реакцией в ***Straces***(*S*). Если бы это было не так в исходной спецификации *S*, то есть если бы $\exists \sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma` \cdot \langle y \rangle \notin \boldsymbol{Straces}(S)$, мы ослабили бы конформность, добавив S-трассу $\sigma \cdot \langle y \rangle$ и, следовательно, добавив также S-трассу $\sigma` \cdot \langle y \rangle$: после S-трассы σ` реакция *y* была запрещена, а теперь она разрешена. Поэтому переход $\boldsymbol{P}(\sigma) \text{—} y \rightarrow \boldsymbol{P}(\sigma \cdot \langle y \rangle)$ проводится в том и только том случае, когда $\forall \sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma` \cdot \langle y \rangle \in \boldsymbol{Straces}(S)$. Заметим, что в этом случае $\boldsymbol{P}(\sigma \cdot \langle y \rangle) \neq \varnothing$.

Аналогично, S-трасса σ продолжается в ***B***(*S*) отказом δ только тогда, когда каждая «старая» S-трасса $\sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)$ также продолжается δ в ***Straces***(*S*). Если это было не так в исходной спецификации *S*, то есть если бы $\exists \sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma` \cdot \langle \delta \rangle \notin \boldsymbol{Straces}(S)$, то мы ослабили бы конформность, добавив S-трассу $\sigma \cdot \langle \delta \rangle$ и, следовательно, добавив также S-трассу $\sigma` \cdot \langle \delta \rangle$: после S-трассы σ` наблюдение δ было запрещено, а теперь оно разрешено. Поскольку переход по δ в LTS является виртуальным, вместо перехода $\boldsymbol{P}(\sigma) \text{—} \delta \rightarrow \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$ мы проведем переход $\boldsymbol{P}(\sigma) \text{—} \tau \rightarrow \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$ в том и только том случае, когда $\forall \sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)\ \sigma` \cdot \langle \delta \rangle \in \boldsymbol{Straces}(S)$ и $\boldsymbol{P}(\sigma) \neq \boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$. В состоянии $\boldsymbol{P}(\sigma \cdot \langle \delta \rangle)$ не будут проводиться переходы по реакциям и τ-переходы, то есть оно будет стационарным. Заметим, что в этом случае $\boldsymbol{P}(\sigma \cdot \langle \delta \rangle) \neq \varnothing$.

Поскольку $\boldsymbol{B}(S)$ должна быть всюду определена по стимулам, в каждом состоянии $\boldsymbol{P}(\sigma)$ для каждого стимула $x \in X$ проводится переход $\boldsymbol{P}(\sigma) \text{—} x \rightarrow \boldsymbol{P}(\sigma \cdot \langle x \rangle)$.

Теперь уже может оказаться, что $\boldsymbol{P}(\sigma \cdot \langle x \rangle) = \varnothing$, то есть каждая S-трасса $\sigma` \in \boldsymbol{d}(\sigma) \cap \boldsymbol{Straces}(S)$ не продолжается в $S$ стимулом $x$. Это означает, что после каждой такой S-трассы $\sigma`$ спецификация S не налагает на реализацию никаких ограничений, допуская хаотическое поведение, то есть все возможные S-трассы. Тогда и в $\boldsymbol{B}(S)$ после S-трассы σ, то есть в состоянии $\varnothing$, мы должны разрешить хаотическое поведение. Такое состояние s, в котором определено хаотическое поведение, назовем *демоническим*: $\forall \sigma \in L_\delta^*\ s = \sigma \Rightarrow$. Демоническое состояние $\varnothing$ может быть реализовано так, как показано на Рис.6.

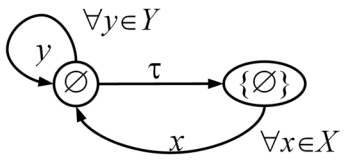


Рис.6. Демоническое состояние $\varnothing$

Проблема возникает тогда, когда в процессе преобразования $\boldsymbol{B}$ получается такая «новая» S-трасса σ (как продолжение «старой» неконформной S-трассы), которую нельзя продолжить ни какой-либо реакцией, ни отказом δ без изменения конформности. Такой S-трассе будет соответствовать стабильное состояние $\boldsymbol{P}(\sigma)$, в котором мы не должны определять (без изменения конформности) переходы по реакциям, но которое в то же время не должно быть стационарным. Такое состояние будем называть *неконформным*. Чтобы формально выразить неконформность состояния, определим в нем переход-петлю по специальной фиктивной реакции $error \notin Y$. Эта реакция добавляется в алфавит LTS $\boldsymbol{B}(S)$.

Определим преобразование $\boldsymbol{B}$ формально. Пусть задан алфавит стимулов и реакций $L = X \cup Y$, где $X \subseteq \mathbf{X}$ и $Y \subseteq \mathbf{Y}$, и спецификация S=LTS (V, L, E, s_0).

Определим для семейства P множеств состояний оператор ***after*** таким образом, чтобы для любой S-трассы σ такой, что $\boldsymbol{P}(\sigma) = P$, и любого наблюдения u было выполнено равенство $\boldsymbol{P}(\sigma \cdot \langle u \rangle) = P$ ***after*** u. Для $z \in L$, $p \in \mathcal{P}(V)$, $P \in \mathcal{P}(\mathcal{P}(V))$:

p ***after*** $z \triangleq \cup \{s$ ***after*** $\langle z \rangle \mid s \in p\}$, $\quad P$ ***after*** $z \triangleq \{p$ ***after*** $z \neq \varnothing \mid p \in P\}$,

p ***after*** $\delta \triangleq \cup \{s$ ***after*** $\langle \delta \rangle \mid s \in p\}$, $\quad P$ ***after*** $\delta \triangleq P \cup \{p$ ***after*** $\langle \delta \rangle \neq \varnothing \mid p \in P\}$.

Заметим, что всегда (P ***after*** δ) ***after*** $\delta = P$ ***after*** δ, $\varnothing$ ***after*** $z = \varnothing$ ***after*** $\delta = \varnothing$.

Тогда $\boldsymbol{B}(S) \triangleq \mathrm{LTS}(V_1,L_1,E_1,s_1)$, где $V_1=\mathcal{P}(\mathcal{P}(V))$, $L_1=L\cup\{error\}$ и $error\in\mathbf{Y}\backslash Y$, $s_1=\{s_0\ \boldsymbol{after}\ \epsilon\}$, а E_1 – наименьшее множество, порождаемое следующими правилами вывода:

$\forall P\in V_1\ \forall x\in X\ \forall y\in Y$

1) $P\neq\varnothing$ & $\forall p\in P\ p\ \boldsymbol{after}\ y\neq\varnothing$ $\vdash P\text{—}y\rightarrow P\ \boldsymbol{after}\ y$,
2) $P\neq\varnothing$ & $\forall p\in P\ p\ \boldsymbol{after}\ \delta\neq\varnothing$ & $P\neq P\ \boldsymbol{after}\ \delta$ $\vdash P\text{—}\tau\rightarrow P\ \boldsymbol{after}\ \delta$,
3) $P\neq\varnothing$ & $P\ \boldsymbol{conf}$ $\vdash P\text{—}x\rightarrow P\ \boldsymbol{after}\ x$,
4) $P\neq\varnothing$ & $\neg(P\ \boldsymbol{conf})$ $\vdash P\text{—}error\rightarrow P$,
5) $\vdash \varnothing\text{—}y\rightarrow\varnothing$,
6) $\vdash \varnothing\text{—}\tau\rightarrow\{\varnothing\}$,
7) $\vdash \{\varnothing\}\text{—}x\rightarrow\varnothing$,

где $P\ \boldsymbol{conf} \triangleq \exists u\in Y_\delta\ \forall p\in P\ p\ \boldsymbol{after}\ u\neq\varnothing$.

4.2. Преобразование *D*

На втором этапе мы избавимся от неконформных S-трасс в LTS $\boldsymbol{B}(S)$. Прежде всего, неконформны все S-трассы, заканчивающиеся в неконформном состоянии P, в котором по 4-ому правилу вывода для $\boldsymbol{B}$ был определен переход $P\text{—}error\rightarrow P$.

Далее объявим неконформным каждое состояние P, для которого выполнено хотя бы одно из следующих трех условий:

1) Состояние P стабильно и в нем определен переход по стимулу $P\text{—}x\rightarrow P`$, ведущий в неконформное состояние $P`$. Поскольку в LTS $\boldsymbol{B}(S)$ определено не более одного перехода по стимулу в каждом состоянии, после удаления перехода $P\text{—}x\rightarrow P`$ возникает блокировка стимула x в состоянии P.
2) Состояние P стабильно и в нем определен хотя бы один переход по реакции, но все переходы по реакциям из состояния P ведут в неконформные состояния. После удаления всех переходов по реакциям из состояния P в этом состоянии возникает отказ δ, которого раньше не было (несохранение конформности).
3) В состоянии P не определены переходы по реакциям и определен τ-переход $P\text{—}\tau\rightarrow P`$, ведущий в неконформное состояние $P`$. Поскольку в LTS $\boldsymbol{B}(S)$ определено не более одного τ-перехода в каждом состоянии, после удаления перехода $P\text{—}\tau\rightarrow P`$ в состоянии P возникает отказ δ, поэтому S-трассы, заканчивающиеся в состоянии P, по-прежнему, неконформно продолжаются отказом δ.

Будем повторять эту процедуру объявления неконформности состояний до тех пор, пока это возможно. Формально, множество $W(\boldsymbol{B}(S))$ неконформных состояний LTS $\boldsymbol{B}(S)$=LTS(V_1,L_1,E_1,s_1) – это минимальное множество, порождаемое следующими правилами вывода: $\forall P \in V_{S1}$

$P \text{—} error \rightarrow P \quad \vdash P \in W(\boldsymbol{B}(S))$,

$P \text{—} \tau \nrightarrow \;\&\; \exists x \in X \, \exists P` \; P \text{—} x \rightarrow P` \;\&\; P` \in W(\boldsymbol{B}(S)) \quad \vdash P \in W(\boldsymbol{B}(S))$,

$P \text{—} \tau \nrightarrow \;\&\; \exists y \in Y \, P \text{—} y \rightarrow \;\&\; \forall y \in Y \, \forall P` \, (P \text{—} y \rightarrow P` \Rightarrow P` \in W(\boldsymbol{B}(S))) \quad \vdash P \in W(\boldsymbol{B}(S))$,

$\forall y \in Y \, P \text{—} y \nrightarrow \;\&\; \exists P` \; P \text{—} \tau \rightarrow P` \;\&\; P` \in W(\boldsymbol{B}(S)) \quad \vdash P \in W(\boldsymbol{B}(S))$.

Если оказалось, что $s_1 \in W(\boldsymbol{B}(S))$, то все S-трассы LTS $\boldsymbol{B}(S)$ неконформны; у такой спецификации нет конформных реализаций.

В противном случае удалим из LTS $\boldsymbol{B}(S)$ все неконформные состояния и переходы, начинающиеся или заканчивающиеся в таких состояниях.

4.3. Преобразование *C*

Определим итоговое пополнение:
$\boldsymbol{C}(S) \triangleq \boldsymbol{D}(\boldsymbol{B}(S)) \triangleq \text{LTS}(V_2,L_2,E_2,s_2)$, где $V_2=V_1 \backslash W(\boldsymbol{B}(S))$, $L_2=L_1 \backslash \{error\}=L$, $s_2=s_1=\{s_0 \; \boldsymbol{after} \; \epsilon\}$, $E_2=\{(P,u,P`) \in E_1 | P \in V_2 \;\&\; P` \in V_2\}$.

По построению LTS $\boldsymbol{C}(S)$ строго конвергентна, всюду определена по стимулам и локально конечно ветвящаяся, то есть принадлежит классу $\boldsymbol{LTS_3}$. В [1, Утв.50] доказана эквивалентность $\boldsymbol{C}(S) \sim_{\boldsymbol{ioco}} S$. Аналогичное утверждение доказано в [2, Теорема 22]. Отсюда, как было сказано выше (по [1, 4.3.1, Утв. 116]), на классе пополненных спецификаций решаются все указанные выше четыре проблемы отношения ***ioco***. Монотонность понимается как безусловная монотонность, а для сохранения конформности при тестировании контексте предполагается, что контекст принадлежит $\boldsymbol{LTS_2}$.

Если LTS-спецификация конечна (конечно число переходов), то пополненная спецификация $\boldsymbol{C}(S)$ конечна, если не возникает необходимости в демоническом состоянии, то есть не применяются правила вывода 5÷7. В этом случае преобразование $\boldsymbol{C}$, очевидно, алгоритмически выполняется за конечное время. Реализация демонического состояния конечна тогда и только тогда, когда конечен алфавит, поскольку в состоянии $\varnothing$ проводятся переходы по всем реакциям (правило вывода 5), а в состоянии $\{\varnothing\}$ – по всем стимулам (правило вывода 7). Поэтому при наличии демонического состояния преобразование $\boldsymbol{C}$ алгоритмически выполняется за конечное время только для конечного алфавита. Для конечных LTS их композиция конечна и алгоритмически

выполняется за конечное время. Проверка строгой конвергентности конечной LTS выполняется также за конечное время (проверяется отсутствие τ-циклов).

5. Примеры пополнения спецификаций

Рассмотрим пополнения спецификаций для примеров из раздела 3.

Решение проблемы рефлексивности отношения ***ioco*** и проблемы неконформных S-трасс спецификации. На Рис.7 показаны пополнения спецификаций S_5 и S_6 (Рис.3). Неконформные состояния выделены серым фоном.

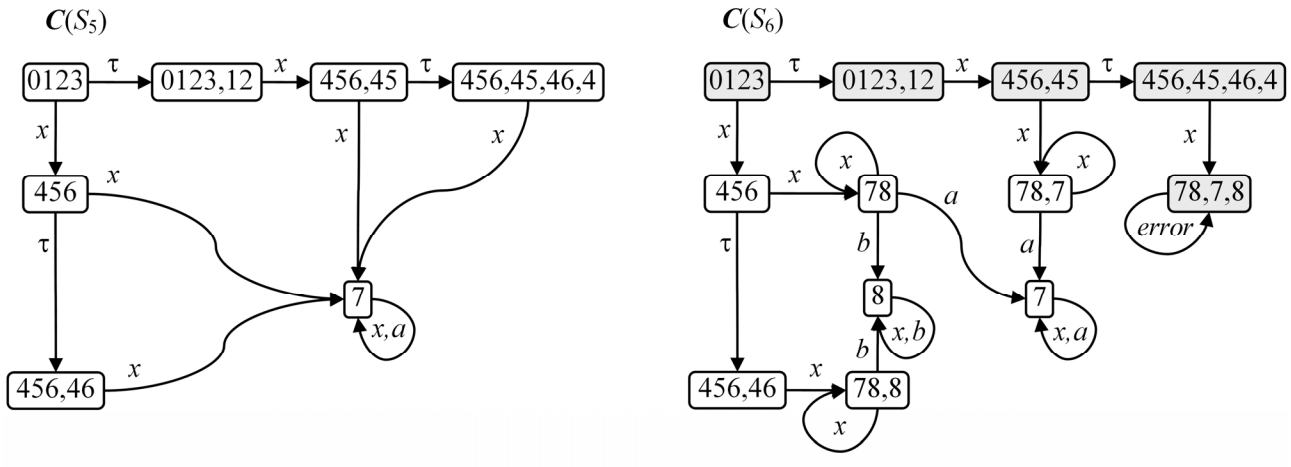


Рис.7. Пополнение спецификаций S_5 и S_6 (Рис.3)

Решение проблемы немонотонности отношения ***ioco***. На Рис.8 показаны пополнения спецификаций S_A и S_B (Рис.4), а также композиция пополненных спецификаций.

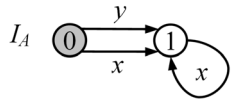
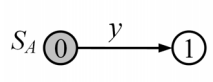
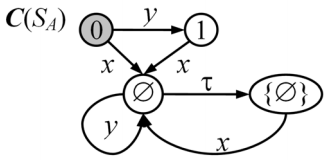
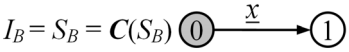
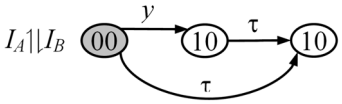
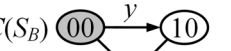
алфавиты	реализации	спецификации	конформность
$A=X_A\cup Y_A$ $X_A=\{x\}$ $Y_A=\{y\}$			I_A ***ioco*** S_A
			I_A ***ioco*** $\boldsymbol{C}(S_A)$
$B=X_B\cup Y_B$ $X_B=\varnothing$ $Y_B=\{\underline{x}\}$			I_B ***ioco*** S_B I_B ***ioco*** $\boldsymbol{C}(S_B)$
$X_{A\|\|B}=\varnothing$ $Y_{A\|\|B}=\{y\}$			$I_A\|\|I_B$ ~~***ioco***~~ $S_A\|\|S_B$
			$I_A\|\|I_B$ ***ioco*** $\boldsymbol{C}(S_A)\|\|\boldsymbol{C}(S_B)$

Рис.8. Пополнение спецификаций S_A и S_B (Рис.4)

Решение проблемы сохранения конформности при тестировании в контексте из класса $\boldsymbol{LTS_2}$. На Рис.9 показано пополнение спецификации S (Рис.5).

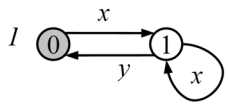
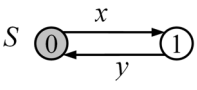
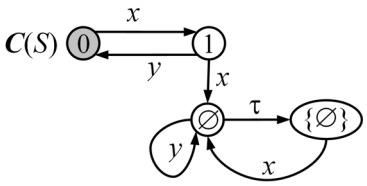
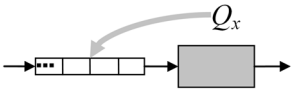
алфавиты	реализации	спецификации	конформность
$A=X_A\cup Y_A$, $X_A=\{x\}$, $Y_A=\{y\}$			I ***ioco*** S
			I ***ioco*** $\boldsymbol{C}(S)$
$B=X_B\cup Y_B$, $X_B=\{x,\underline{y}\}$, $Y_B=\{y,\underline{x}\}$ $X_{A\|\|B}=\{x\}$, $Y_{A\|\|B}=\{y\}$			$I\|\|Q_x$ ~~***ioco***~~ $S\|\|Q_x$ $I\|\|Q_x$ ***ioco*** $\boldsymbol{C}(S)\|\|Q_x$

Рис.9. Пополнение спецификации S (Рис.5)

6. Тестирование в контексте общего вида

В предыдущем разделе показано, что для спецификаций из класса $\boldsymbol{LTS_3}$ отношение ***ioco*** безусловно сохраняется при тестировании в контексте из класса

$\boldsymbol{LTS_2}$. Однако, требование всюду определенности по стимулам нарушается для некоторых практически используемых контекстов. Например, ограниченные очереди стимулов и реакций можно трактовать как LTS с блокировкой стимулов (когда очередь полна). Здесь мы рассмотрим случай безусловного сохранения ***ioco*** для контекста общего вида и спецификаций из класса $\boldsymbol{LTS_3}$ (в частности, полученных с помощью пополнения $\boldsymbol{C}$).

Для применения ***ioco*** после композиции требуется, чтобы композиция $I|\!\rfloor Q$ конформной реализации I с контекстом Q принадлежала классу $\boldsymbol{LTS_2}$, а композиция $S|\!\rfloor Q$ спецификации S с тем же контекстом Q – классу $\boldsymbol{LTS_1}$. Композиция $I|\!\rfloor Q$ для контекста Q общего вида (в отличие от контекста из класса $\boldsymbol{LTS_2}$) может содержать блокировки стимулов, тем самым не принадлежать классу $\boldsymbol{LTS_2}$. Более того, такая реализация всегда существует для спецификации $S\in\boldsymbol{LTS_2}$, если композиция $S|\!\rfloor Q$ содержит блокировки стимулов: в силу рефлексивности ***ioco*** такой реализацией является сама спецификаций S. Поэтому для спецификаций из некоторого подкласса класса $\boldsymbol{LTS_3}$ безусловное сохранение означает, что если $S|\!\rfloor Q\in\boldsymbol{LTS_2}$, то $I|\!\rfloor Q\in\boldsymbol{LTS_2}$ и $I|\!\rfloor Q$ ***ioco*** $S|\!\rfloor Q$.

В [1, глава 4.1] предложено преобразование $\boldsymbol{T}_{\beta\delta}$ для решения проблемы несохранения конформности $\boldsymbol{ioco}_{\beta\gamma\delta}$ при тестировании в контексте общего вида (класс $\boldsymbol{LTS}$). Это преобразование применяется к локально конечно ветвящимся LTS в алфавите с конечным числом реакций. Пересечение класса таких LTS с классом $\boldsymbol{LTS_3}$ обозначим $\boldsymbol{LTS_4}$.

Поскольку на классе спецификаций $\boldsymbol{LTS_2}$ отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ и ***ioco*** совпадают, а $\boldsymbol{LTS_4}\subseteq\boldsymbol{LTS_3}\subseteq\boldsymbol{LTS_2}$, мы можем использовать это преобразование для отношения ***ioco*** и спецификаций из класса $\boldsymbol{LTS_4}$. Здесь мы опишем его адаптированный вариант для спецификаций из класса $\boldsymbol{LTS_4}$, в которых отсутствуют блокировки стимулов, – преобразование $\boldsymbol{T}:\boldsymbol{LTS_4}\to\boldsymbol{LTS_4}$. Для LTS $S\in\boldsymbol{LTS_4}$ состояния LTS $\boldsymbol{T}_{\beta\delta}(S)$ строятся на основе множеств состояний после трасс LTS S. В то же время, как было сказано выше, мы можем не различать трассы, заканчивающиеся в одном множестве состояний, поскольку они имеют одни и те же продолжения. Поэтому состояния LTS $\boldsymbol{T}(S)$ строятся просто на основе множеств состояний LTS S без учета того, какие трассы заканчиваются в том или ином множестве.

Определим преобразование $\boldsymbol{T}$ формально. Пусть задан алфавит стимулов и реакций $L=X\cup Y$, где $X\subseteq\mathbf{X}$ и $Y\subseteq\mathbf{Y}$, и спецификация $S=\mathrm{LTS}(V,L,E,s_0)\in\boldsymbol{LTS_4}$. Тогда $\boldsymbol{T}(S)\triangleq\mathrm{LTS}(V_T,L,E_T,s_T)$, где $V_T=\mathcal{P}(V)\cup(\mathcal{P}(V)\times Y)$, $s_T=\{s_0\ \boldsymbol{after}\ \epsilon\}$, а E_T – наименьшее множество, порождаемое следующими правилами вывода:
$\forall p\in\mathcal{P}(V)\ \forall x\in X\ \forall y\in Y$

1) p ***after*** $x \neq \varnothing$ $\quad\vdash p \text{—}x\text{→} p$ ***after*** x;

2) p ***after*** $y \neq \varnothing$ $\quad\vdash p \text{—}\tau\text{→}(p,y) \text{—}y\text{→} p$ ***after*** y;

3) p ***after*** $x \neq \varnothing$ & p ***after*** $y \neq \varnothing$ $\quad\vdash (p,y) \text{—}x\text{→} p$ ***after*** x;

4) p ***after*** $\delta \neq \varnothing$ & p ***after*** $\delta \neq$ p $\quad\vdash p \text{—}\tau\text{→} p$ ***after*** δ.

По построению, для $S \in \boldsymbol{LTS_4}$ имеет место $\boldsymbol{T}(S) \in \boldsymbol{LTS_4}$.

Сначала покажем инвариантность ***ioco*** при преобразовании $\boldsymbol{T}$:
$\forall S \in \boldsymbol{LTS_4}\ \boldsymbol{T}(S) \sim_{\boldsymbol{ioco}} S$.

Действительно, по [1, 4.1.2, Утв. 97] $\boldsymbol{T}_{\beta\delta}(S)$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ S & S $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\boldsymbol{T}_{\beta\delta}(S)$, что для $S \in \boldsymbol{LTS_4}$ эквивалентно $\boldsymbol{T}(S)$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ S & S $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\boldsymbol{T}(S)$. По транзитивности отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ [1, 2.2.7, Утв. 20] это влечет $\forall I \in \boldsymbol{LTS}$ I $\boldsymbol{ioco}_{\beta\gamma\delta}$ $S \Leftrightarrow I$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\boldsymbol{T}(S)$. Для $S \in \boldsymbol{LTS_4}$ имеем $\boldsymbol{T}(S) \in \boldsymbol{LTS_4}$. Поскольку на классе спецификаций $\boldsymbol{LTS_4} \subseteq \boldsymbol{LTS_2}$ отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ и ***ioco*** совпадают, имеем: $\forall I \in \boldsymbol{LTS}$ I ***ioco*** $S \Leftrightarrow I$ ***ioco*** $\boldsymbol{T}(S)$, что означает $\boldsymbol{T}(S) \sim_{\boldsymbol{ioco}} S$.

Теперь покажем безусловное сохранение ***ioco*** при тестировании в контексте общего вида для $\boldsymbol{T}$-преобразованных спецификаций:
$\forall I \in \boldsymbol{LTS_2}\ \forall S \in \boldsymbol{LTS_4}\ \forall Q \in \boldsymbol{LTS}\ I$ ***ioco*** S & $\boldsymbol{T}(S) \mid\!\downarrow Q \in \boldsymbol{LTS_2} \Rightarrow I \mid\!\downarrow Q$ ***ioco*** $\boldsymbol{T}(S) \mid\!\downarrow Q$.

Действительно, поскольку на классе спецификаций $\boldsymbol{LTS_4} \subseteq \boldsymbol{LTS_2}$ отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ и ***ioco*** совпадают, I ***ioco*** S влечет I $\boldsymbol{ioco}_{\beta\gamma\delta}$ S. По [1, 4.1.2, Утв. 97] это влечет $I \mid\!\downarrow Q$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\boldsymbol{T}_{\beta\delta}(S) \mid\!\downarrow Q$, что для $S \in \boldsymbol{LTS_4}$ эквивалентно $I \mid\!\downarrow Q$ $\boldsymbol{ioco}_{\beta\gamma\delta}$ $\boldsymbol{T}(S) \mid\!\downarrow Q$. Поскольку на классе спецификаций $\boldsymbol{LTS_2}$ отношения $\boldsymbol{ioco}_{\beta\gamma\delta}$ и ***ioco*** совпадают, и $\boldsymbol{T}(S) \mid\!\downarrow Q \in \boldsymbol{LTS_2}$, имеем: $I \mid\!\downarrow Q$ ***ioco*** $\boldsymbol{T}(S) \mid\!\downarrow Q$.

Окончательное преобразование пополнения для тестирования в контексте общего вида выполняется как последовательные преобразования $\boldsymbol{C}$ и $\boldsymbol{T}$ для любых строго конвергентных LTS-спецификаций в алфавите с конечным числом реакций.

Если LTS-спецификация S конечна (конечно число переходов), то спецификация $\boldsymbol{T}(S)$ конечна. В этом случае преобразование $\boldsymbol{T}$, очевидно, алгоритмически выполняется за конечное время. Композиция конечной LTS-спецификации с контекстом будет конечной и может быть выполнена алгоритмически за конечное время, если LTS контекста конечна. Для конечной композиции ее строгая конвергентность и всюду определенность по стимулам проверяются за конечное время.

7. Заключение

В статье предложено преобразование пополнения ***C*** LTS-спецификаций для конформности ***ioco***. Пополненная спецификация эквивалентна исходной спецификации в смысле сохранения класса конформных реализаций, то есть отношение ***ioco*** инвариантно к пополнению. Преобразование ***C*** решает проблему рефлексивности ***ioco*** и удаляет неконформные S-трассы исходной спецификации.

Также решается проблема монотонности конформности при композиции: если композиция пополненных спецификаций строго конвергентна, то ей конформна композиция любых реализаций, конформных этим спецификациям.

Преобразование ***C*** также решает проблему несохранения конформности при тестировании в контексте («ложные» ошибки): если контекст так же как реализация всюду определен по стимулам, а композиция пополненной спецификации с контекстом строго конвергентна, то ей конформна композиция любой конформной реализации с этим контекстом.

Для конечной LTS-спецификации в конечном алфавите спецификация ***C***(*S*) конечна, и преобразование ***C*** алгоритмически выполняется за конечное время. Композиция конечных LTS конечна и алгоритмически выполняется за конечное время. Проверка строгой конвергентности композиции также выполняется за конечное время.

Для контекста общего вида (когда допускаются блокировки стимулов) предложено дальнейшее преобразование ***T*** спецификации ***C***(*S*), которое сохраняет класс конформных реализаций и при условии конечности числа реакций в алфавите гарантирует: если композиция спецификации ***T***(***C***(*S*)) с контекстом строго конвергентна и всюду определена по стимулам, то ей конформна композиция любой конформной реализации с этим контекстом.

Для конечной спецификации ***C***(*S*) в алфавите с конечным числом реакций спецификация ***T***(***C***(*S*)) конечна, и преобразование ***T*** алгоритмически выполняется за конечное время. Композиция конечной LTS-спецификации с конечным контекстом конечна и алгоритмически выполняется за конечное время. Проверка строгой конвергентности и всюду определенности по стимулам конечной композиции алгоритмически выполняется за конечное время.

Литература

1. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.

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. http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
3. Gregor V. Bochmann , Alexandre Petrenko. Protocol testing: review of methods and relevance for software testing, Proceedings of the 1994 international symposium on Software testing and analysis, pp.109-124, August 17-19, 1994, Seattle, Washington, United States.
4. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the Association for Computing Machinery 31 (1984) 560–599.
5. C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.
6. J. L. Huo, A. Petrenko. On Testing Partially Specified IOTS through Lossless Queues. Proc. of TestCom 2004, LNCS 2978, pp. 76–94, Springer-Verlag, 2004.
7. C. Jard, T. Jéron, L. Tanguy, C. Viho. 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.
8. Lee D., Yannakakis M. Principles and Methods of Testing Finite State Machines – A Survey. Proceedings of the IEEE 84, No. 8, 1090–1123, 1996.
9. R. Milner. Communication and Concurrency. Prentice-Hall, 1989.
10. De Nicola, R., Segala, R.: A process algebraic view of Input/Output Automata. Theoretical Computer Science 138 (1995) 391–423.
11. Revised Working Draft on “Framework: Formal Methods in Conformance Testing”. JTC1/SC21/WG1/Project 54/1 // ISO Interim Meeting / ITU-T on, Paris, 1995.
12. Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3, 1996.
13. van der Bijl M., Rensink A., Tretmans J. Compositional testing with ioco. In 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.
14. van der Bijl M., Rensink A., Tretmans J. Component Based Testing with ioco. CTIT Technical Report TR-CTIT-03-34, University of Twente, 2003.