И.Б.Бурдонов, А.С.Косачев.
Тестирование конформности на основе соответствия состояний.
Труды Института системного программирования РАН, N 18, 2010, стр.183-220.
38 стр.

Тестирование конформности на основе соответствия состояний

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

Аннотация. Статья посвящена тестированию соответствия (конформности) реализации требованиям спецификации. Идея безопасного тестирования, предложенная авторами для конформности, основанной на трассах наблюдений, распространяется на случай (слабой) симуляции – конформности, основанной на соответствии состояний реализации и спецификации. Строится теория безопасной симуляции и ее тестирования. Предлагаются общий алгоритм полного тестирования и его модификация для практического применения, опирающаяся на некоторые ограничения на реализацию и спецификацию.

1. Введение

Тестирование конформности – 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.

Если при тестировании мы можем наблюдать только (внешние) действия, выполняемые реализацией в ответ на тестовые воздействия или отсутствие таких действий (отказ), то конформность определяется на основе трасс наблюдений, то есть последовательностей действий и отказов. Спецификация в этом случае описывает те трассы, которые допускаются в реализации.

Тестирование, при котором гарантировано конечное время ожидания наблюдения после тестового воздействия, называется безопасным. Возможны две причины бесконечного ожидания: дивергенция и ненаблюдаемые отказы. Дивергенция – это бесконечное выполнение реализацией внутренних (ненаблюдаемых) действий. Ненаблюдаемый отказ – это отсутствие выполняемых реализацией внешних действий, которое тест не может

определить за конечное время[1]. В обоих случаях тест не может ни продолжить тестирование, ни закончить его, так как неизвестно, нужно ли ждать наблюдения или никакого наблюдения не будет. Тестирование, при котором не возникает дивергенции и ненаблюдаемых отказов, называется безопасным.

Кроме этого возможно специальное, не регулируемое тестовыми воздействиями, действие реализации, которое называется *разрушением*. Оно моделирует любое нежелательное поведение системы, в том числе и ее реальное разрушение. Семантика разрушения предполагает, что оно также не должно возникать при безопасном тестировании.

Спецификация описывает те ситуации, при которых тестовое воздействие должно быть безопасно в реализации. Соответственно, конформность основана только на безопасном поведении реализации. Конформность, учитывающая безопасность, введена авторами этой статьи в [1,2,3,4].

Кроме конформности, основанной только на трассах наблюдений, в литературе рассматриваются разные виды конформностей, основанных на соответствии состояний реализации и спецификации (обзор см. в [5]). Такие конформности называются симуляциями. Симуляция требует, чтобы правильным было не только наблюдаемое внешнее поведение реализации, но и изменение ее состояний. Все рассматриваемые в литературе симуляции либо не учитывают безопасности тестирования, предполагая отсутствие дивергенции и ненаблюдаемых отказов, либо предполагают возможность прямого наблюдения дивергенции и всех отказов. Также они не учитывают возможность разрушения.

В данной статье вводится симуляция, учитывающая безопасность, называемая безопасной симуляцией. Спецификация описывает не только класс конформных ей реализацией, но и гипотезу о безопасности, определяющую более широкий класс реализаций, которые можно безопасно тестировать для проверки конформности.

Выбор симуляции в качестве конформности наиболее естественен, когда состояния реализации доступны для их наблюдения. Тестирование, при котором в любой момент времени можно опросить текущее состояние

[1] Например, интервал времени между тестовым воздействием и ответным внешним действием ограничен некоторым тайм-аутом. Превышение тайм-аута при ожидании внешних действий означает наблюдение отказа. При отсутствии такого рода ограничений отказ ненаблюдаем.

реализации, называется тестированием с открытым состоянием. Задача тестирования – обнаружение ошибок в реализации, понимаемое как несоответствие ее поведения спецификационным требованиям. Тестирование полное, если обнаруживается любая ошибка и не фиксируются «ложные» ошибки. В данной статье рассматривается полное тестирование с открытой состоянием безопасной симуляции.

Это рассмотрение проводится как в общетеоретическом, так и в практическом планах. Теоретическое полное тестирование должно обнаруживать любую ошибку за конечное время, но при отсутствии ошибок может продолжаться бесконечно. Причины бесконечного тестирования – это бесконечность реализации и/или спецификации, а также неограниченный недетерминизм поведения реализации. При некоторых на ограничениях возможно построение полных тестов, в любом случае завершающих свою работу за конечное время. Такие тесты уже можно использовать на практике.

2-ой раздел статьи содержит основные положения теории конформности: семантика взаимодействия и безопасное тестирование, математическая модель реализации и спецификации, определение симуляции, гипотеза о безопасности и определение безопасной симуляции. В 3-ем разделе рассматривается связь безопасной симуляции с трассовой конформностью. 4-ый раздел посвящен теоретическому тестированию: определяется полнота тестирования, описывается общий алгоритм тестирования и вводится достаточное условие его полноты. 5-ый раздел посвящен практическому тестированию: определяются ограничения на реализацию и спецификацию, позволяющие так модифицировать общий алгоритм тестирования, чтобы он стал конечным и полным, а также предлагается более практический алгоритм, состоящий из алгоритма обхода реализации и последующей верификации симуляции, приводится пример верификации симуляции.

2. Теория конформности

2.1. Семантика взаимодействия и безопасное тестирование

Данная работа развивает теорию тестирования конформности, изложенную в [1,2,3,4]. Тестирование понимается как 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 задается 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.

При нажатии кнопки $Q \in \mathbf{Q}$, вообще говоря, неизвестно, нужно ли ждать наблюдения или никакого наблюдения не будет, поскольку возник ненаблюдаемый отказ. Тем самым, нельзя ни продолжить тестирование, ни закончить его. Поэтому при правильном взаимодействии с реализацией, в том числе при тестировании, должна быть уверенность, что при нажатии кнопки $Q \in \mathbf{Q}$ не возникает отказа.

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. Дивергенция сама по себе не опасна, но при попытке выхода из нее, когда нажимается любая кнопка $P \in \mathbf{R} \cup \mathbf{Q}$, неизвестно, нужно ли ждать наблюдения или бесконечно долго будут выполняться только внутренние действия. Эта ситуация аналогична возникновению ненаблюдаемого отказа при нажатии кнопки $Q \in \mathbf{Q}$: нельзя ни продолжить тестирование, ни закончить его. Поэтому при правильном взаимодействии с реализацией, в том числе при тестировании, следует избегать тестовых воздействий, если в реализации возникла дивергенция.

Кроме этого вводится специальное, также не регулируемое кнопками, действие, которое называется *разрушением* и обозначается γ. Оно моделирует любое нежелательное поведение системы, в том числе и ее реальное разрушение. Семантика разрушения предполагает, что оно не должно возникать при правильном взаимодействии с реализацией, в том числе при тестировании.

Взаимодействие с реализацией, в том числе тестирование, при котором не возникает ненаблюдаемых отказов, попыток выхода из дивергенции и разрушения, называется безопасным.

2.2. LTS-модель

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 сводится к выполнению того или иного перехода, определенного в текущем состоянии (начиная с начального состояния) и разрешаемого нажатой кнопкой (τ- и γ-переходы всегда разрешены)[2].

Состояние s *дивергентно* (обозначается $s\uparrow$), если в нем начинается бесконечная цепочка τ-переходов (в частности, τ-цикл); в противном случае состояние *конвергентно* ($s\downarrow$). Состояние s *стабильно*, если из него не выходят τ- и γ-переходы: $s \text{—}\tau\not\to$ & $s \text{—}\gamma\not\to$. Отказ $P \in \mathbf{R}$ порождается стабильным состоянием, из которого нет переходов по действиям из P: $\forall z \in P \cup \{\tau,\gamma\}\ s \text{—}z\not\to$.

Для определения трасс LTS $\mathbf{S}$ добавим в каждом ее стабильном состоянии виртуальные петли $s \text{—}P\to s$, помеченные порождаемыми отказами, и Δ-переходы из дивергентных состояний $s \text{—}\Delta\to$. В полученной LTS рассмотрим маршруты, не продолжающиеся после Δ- и γ-переходов. Трассой назовем последовательность σ пометок на переходах такого маршрута с пропуском символов τ. Будем обозначать для $s, s' \in V_S$, $u \in \mathbf{L} \cup \mathbf{R} \cup \mathbf{Q} \cup \{\gamma,\Delta\}$, $\sigma = \langle u_1, \ldots, u_n \rangle \in (\mathbf{L} \cup \mathbf{R} \cup \mathbf{Q} \cup \{\gamma,\Delta\})^*$:

[2] При параллельной композиции двух LTS, моделирующей взаимодействие, нажатой кнопке для одной LTS соответствует состояние другой LTS, в котором определены переходы по всем действиям, разрешаемым этой кнопкой (при композиции в CCS это противоположные им действия), и только они.

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

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

$s=\sigma\Rightarrow s` \triangleq \exists s_1,\dots,s_{n+1}\ s=s_1=\langle u_1\rangle\Rightarrow s_2\dots s_n=\langle u_n\rangle\Rightarrow s_{n+1}=s`,$

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

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

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

Запись $s=\sigma\Rightarrow s`$ (или $s\Rightarrow s`$) понимается как наличие маршрута, начинающегося в состоянии s (*пресостоянии*), заканчивающегося в состоянии $s`$ (*постсостоянии*) и имеющего трассу σ (или пустую трассу $\langle\rangle$). **R**-трассой будем называть трассу, не содержащую отказов из **Q**, а **R**-маршрутом – маршрут с **R**-трассой. Множество **R**-трасс, начинающихся в состоянии s, обозначим $\boldsymbol{T}(s)$. По умолчанию, будем считать, что трасса начинается в начальном состоянии LTS: будем обозначать $\mathbf{S}\ \textbf{\textit{after}}\ \sigma = s_0\ \textbf{\textit{after}}\ \sigma$ и $\boldsymbol{T}(\mathbf{S})=\boldsymbol{T}(s_0)$.

2.3. Слабая симуляция

В наших предыдущих работах [1,2,3,4] рассматривалась конформность, основанная только на трассах наблюдений и не учитывающая состояний реализации и спецификации. В то же время в литературе рассматриваются разные виды конформностей, основанные на соответствии R состояний реализации и спецификации[3]. Такие конформности называются симуляциями (обзор см. в [5]). Симуляция требует, чтобы каждое наблюдение u, возможное в реализационном состоянии i с постсостоянием $i`$, было возможно в каждом соответствующем ему спецификационном состоянии $s$, и в спецификации для $s$ и $u$ нашлось бы постсостояние $s`$, соответствующее $i`$. Разные симуляции отличаются друг от друга, главным образом, отношением к наблюдаемости внутренних действий (τ). В данной статье мы исходим из основного допущения о принципиальной ненаблюдаемости τ-действий: при тестировании мы не можем различать наличие или отсутствие τ-действий как до, так и после внешнего действия (при наблюдении отказа τ-действия возможны, очевидно, только до отказа). Этому соответствует слабая симуляция (weak simulation), называемая также наблюдаемой симуляцией

[3] Мнемоника: R – **R**elation.

(observation simulation). Мы дадим три эквивалентных определения слабой симуляции (Рис.1).

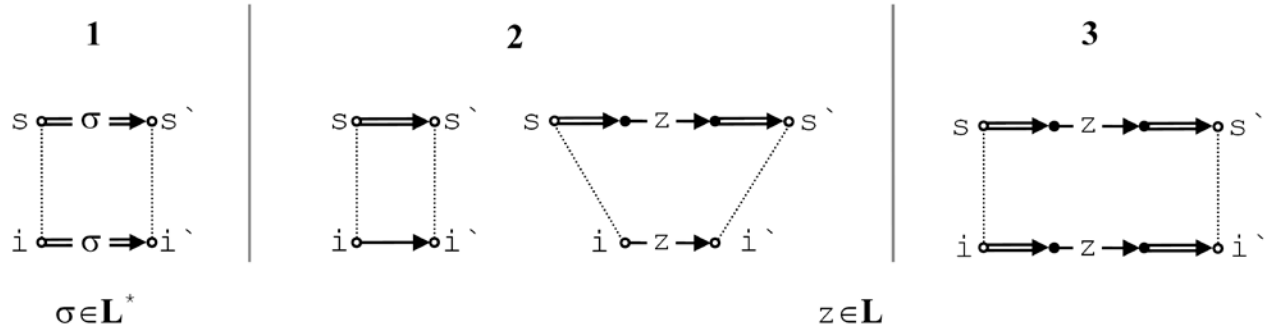


Рис.1. Три определения слабой симуляции

$\mathbf{I} \le^1_{ws} \mathbf{S} \triangleq \exists R \subseteq V_\mathbf{I} \times V_\mathbf{S}\ (i_0,s_0)\in R\ \&\ \forall (i,s)\in R\ \forall \sigma\in \mathbf{L}^*\ \forall i`$
$(i=\sigma\Rightarrow i` \Rightarrow \exists s`\ s=\sigma\Rightarrow s`\ \&\ (i`,s`)\in R).$
$\mathbf{I} \le^2_{ws} \mathbf{S} \triangleq \exists R \subseteq V_\mathbf{I} \times V_\mathbf{S}\ (i_0,s_0)\in R\ \&\ \forall (i,s)\in R\ \forall u\in \mathbf{L}\ \forall i`$
$(i\text{—}\tau\rightarrow i` \Rightarrow \exists s`\ s\Rightarrow s`\ \&\ (i`,s`)\in R)\ \&$
$(i\text{—}u\rightarrow i` \Rightarrow \exists s`\ s=\langle u\rangle\Rightarrow s`\ \&\ (i`,s`)\in R).$
$\mathbf{I} \le^3_{ws} \mathbf{S} \triangleq \exists R \subseteq V_\mathbf{I} \times V_\mathbf{S}\ (i_0,s_0)\in R\ \&\ \forall (i,s)\in R\ \forall u\in \mathbf{L}\ \forall i`$
$(i=\langle u\rangle\Rightarrow i` \Rightarrow \exists s`\ s=\langle u\rangle\Rightarrow s`\ \&\ (i`,s`)\in R).$

Соответствие R, для которого выполнены условия слабой симуляции $\le^k_{ws}$, назовем $\le^k_{ws}$-соответствием или просто *конформным соответствием*.

Первые два определения, принадлежащие Милнеру [8,9], эквивалентны.

Лемма 1: $\le^1_{ws} = \le^2_{ws}$.

Доказательство. Легко показать, что эти определения эквивалентны даже в более сильном смысле: если R является $\le^1_{ws}$-соответствием, то оно же является $\le^2_{ws}$-соответствием, и обратно. □

Лемма 2: $\le^1_{ws} = \le^3_{ws}$.

Доказательство. Поскольку трасса $\langle u\rangle\in\mathbf{L}^*$, любое $\le^1_{ws}$-соответствие является $\le^3_{ws}$-соответствием. Обратное, однако, не верно; зато верно следующее утверждение: из существования $\le^3_{ws}$-соответствия следует существование, быть

может, другого соответствия, которое является как $\leq^3_{ws}$-соответствием, так и $\leq^1_{ws}$-соответствием.

Пусть R – некоторое $\leq^3_{ws}$-соответствие. Пополним его каждой такой парой $(i`,s)$, для которой $(i,s)\in R$ и $i\Rightarrow i`$, и покажем, что полученное соответствие $R`$ также является $\leq^3_{ws}$-соответствием. Нам надо показать, что для добавленной пары $(i`,s)$ и любого маршрута $i`=\langle u\rangle\Rightarrow i``$, где $u\in\mathbf{L}$, существует $s``$ такое, что $s=\langle u\rangle\Rightarrow s``$ & $(i``,s``)\in R`$. Действительно, если $i\Rightarrow i`$ и $i`=\langle u\rangle\Rightarrow i``$, то $i=\langle u\rangle\Rightarrow i``$, а, поскольку $R$ является $\leq^3_{ws}$-соответствием для $\mathbf{I}$ и $\mathbf{S}$, то для $(i,s)\in R$ существует $s``$ такое, что $s=\langle u\rangle\Rightarrow s``$ & $(i``,s``)\in R\subseteq R`$, что и требуется.

Теперь докажем основное утверждение индукцией по трассе σ. Для пустой трассы из $(i,s)\in R`$ и $i\Rightarrow i`$ по построению имеем $(i`,s)\in R`$ и, очевидно, $s\Rightarrow s$. Пусть утверждение верно для трассы σ и докажем его для трассы $\sigma\cdot\langle u\rangle$, где $u\in\mathbf{L}$. Пусть $(i,s)\in R$ и $i=\sigma\cdot\langle u\rangle\Rightarrow i``$. Тогда существует $i`$ такое, что $i=\sigma\Rightarrow i`$ и $i`=\langle u\rangle\Rightarrow i``$. По предположению шага индукции, $\exists s`\ s=\sigma\Rightarrow s`$ & $(i`,s`)\in R`$. Поскольку $R`$ является $\leq^3_{ws}$-соответствием для $\mathbf{I}$ и $\mathbf{S}$, то для $(i`,s`)\in R`$ существует $s``$ такое, что $s`=\langle u\rangle\Rightarrow s``$ & $(i``,s``)\in R`$. Наконец, $s=\sigma\Rightarrow s`$ и $s`=\langle u\rangle\Rightarrow s``$ влечет $s=\sigma\cdot\langle u\rangle\Rightarrow s``$, что и требуется: соответствие $R`$ является $\leq^1_{ws}$-соответствием для $\mathbf{I}$ и $\mathbf{S}$. □

2.4. Отказы

В данной статье под наблюдениями понимаются не только внешние действия из $\mathbf{L}$, но и наблюдаемые отказы из $\mathbf{R}$. Модификация слабой симуляции с отказами выглядит так (изменения по сравнению с $\leqslant^3_{ws}$ подчеркнуты волнистой линией):

$\mathbf{I}\leq^4_{ws}\mathbf{S} \triangleq \exists R\subseteq V_{\mathbf{I}}\times V_{\mathbf{S}}\ (i_0,s_0)\in R$ & $\forall(i,s)\in R\ \forall u\in\mathbf{L}\underset{\sim}{\cup}\mathbf{R}\ \forall i`$
$(i=\langle u\rangle\Rightarrow i` \Rightarrow \exists s`\ s=\langle u\rangle\Rightarrow s`$ & $(i`,s`)\in R)$.

Лемма 3: $\leq^4_{ws}\subset\leq^3_{ws}$.

Доказательство. Отношение $\leq^4_{ws}$ предъявляет больше требований к реализации: требуется верификация не только наблюдаемых действий, но и наблюдаемых отказов. □

На классе реализаций, не имеющих наблюдаемых отказов, эти соответствия совпадают: $\leq^4_{ws} = \leq^3_{ws}$.

Заметим, что после отказа $u \in \mathbf{R}$ не может быть τ-действий, то есть маршрут с трассой $\langle u \rangle$ не может заканчиваться τ-переходом, а только виртуальной петлей по отказу $i` \text{—} u \rightarrow i`$ и $s` \text{—} u \rightarrow s`$.

Лемма 4: Отношения $\leq^k_{ws}$, где $k=1 \div 4$, являются предпорядками (рефлексивны и транзитивны).

Доказательство. Для доказательства рефлексивности отношений достаточно взять тождественное соответствие $R=\{(s,s)|s \in V_s\}$. Для доказательства транзитивности отношений достаточно взять композицию соответствий $R=R_1 \circ R_2=\{(a,c)|\exists b\ (a,b) \in R_1\ \&\ (b,c) \in R_2\}$. Если $(a,c) \in R$, то существует b такое, что $(a,b) \in R_1$ и $(b,c) \in R_2$. Если $a=\langle u \rangle \Rightarrow a`$, то, поскольку $(a,b) \in R_1$, существует $b`$ такое, что $b=\langle u \rangle \Rightarrow b`$ & $(a`,b`) \in R_1$. Тогда, поскольку $(b,c) \in R_2$, существует $c`$ такое, что $c=\langle u \rangle \Rightarrow c`$ & $(b`,c`) \in R_2$. Тогда $(a`,c`) \in R$, что и требуется. □

2.5. Безопасность

Мы будем рассматривать проверку конформности типа слабой симуляции только при безопасном взаимодействии с реализацией.

Состояние s называется *безопасным*, если в этом состоянии не начинается γ-трасса: $s=\langle \gamma \rangle \not\Rightarrow$. При безопасном взаимодействии проходятся только безопасные состояния реализации.

Кнопка $P \in \mathbf{R} \cup \mathbf{Q}$ называется *безопасной в состоянии* s, если ее можно нажимать при безопасном взаимодействии: состояние s безопасно, конвергентно и нажатие кнопки P в состоянии s не вызывает ненаблюдаемого отказа или разрушения после действия, разрешаемого кнопкой:

P ***safe*** $s \triangleq s=\langle\gamma\rangle\not\Rightarrow$ & $s\!\downarrow$ & $(P\in\mathbf{Q} \Rightarrow s=\langle P\rangle\not\Rightarrow)$ & $\forall z\in P\ s=\langle z,\gamma\rangle\not\Rightarrow$.

Будем называть наблюдение u *безопасным в состоянии* s, если оно разрешается некоторой кнопкой P ($u\in P\cup\{P\}$), безопасной в этом состоянии P ***safe*** s; в частности должно быть безопасно состояние s. Внутреннее действие τ безопасно в состоянии, если это состояние безопасно. Переход $s\text{—}u\rightarrow s'$ *безопасен*, если действие u (внешнее или внутреннее) безопасно в пресостоянии s этого перехода. Пустой маршрут *безопасен*, если безопасно его пресостояние, а непустой маршрут *безопасен*, если каждый его переход безопасен. Состояние *безопасно достижимо*, если оно является концом безопасного маршрута, начинающегося в начальном состоянии.

Теперь мы можем определить модификацию слабой симуляции с отказами и безопасностью. Прежде всего, заметим, что, если начальное состояние спецификации не безопасно, то есть $s_0=\langle\gamma\rangle\Rightarrow$, то это разрешает любое поведение реализации, в том числе разрушение с самого начала $i_0=\langle\gamma\rangle\Rightarrow$. В этом случае любая реализация конформна спецификации, но некоторые реализации нельзя не только тестировать, но даже запускать на выполнение, поскольку они могут разрушиться с самого начала. Если же $s_0=\langle\gamma\rangle\not\Rightarrow$, то начальные состояния должны соответствовать друг другу $(i_0,s_0)\in R$.

Далее, при безопасном взаимодействии с реализацией может нажиматься только такая кнопка P, которая безопасна в текущем состоянии реализации i. Если кнопка P безопасна в некотором соответствующем i спецификационном состоянии s, то в этом состоянии s спецификация описывает те наблюдения $u\in P\cup\{P\}$, которые допустимы в конформной реализации после нажатия кнопки P в состоянии i, и постсостояния s' спецификации, хотя бы одно из которых должно соответствовать постсостоянию i' реализации. Иными словами, любое наблюдение в реализации после нажатия кнопки P в состоянии i должно быть и в спецификации после нажатия той же кнопки P в соответствующем состоянии s при условии, что эта кнопка безопасна в s, и в этом случае постсостоянию i' реализации должно соответствовать хотя бы одно постсостояние s'.

Если же кнопка P опасна в некотором соответствующем i спецификационном состоянии s_1, то такое состояние s_1 никак не регламентирует поведение реализации после нажатия кнопки P. Это, однако, не означает, что в состоянии i могут быть любые наблюдения $u\in P\cup\{P\}$ с

любыми постсостояниями $i`$, поскольку наблюдение u может разрешаться той же кнопкой P в другом спецификационном состоянии s, в котором кнопка P безопасна, или другой кнопкой, безопасной в том же или другом спецификационном состоянии, соответствующем состоянию i. Для проверки конформности сравниваются наблюдения и постсостояния в реализации и спецификации только после нажатия кнопок, безопасных в них обеих. Модификация слабой симуляции с отказами и безопасностью выглядит так (изменения по сравнению с $\leqslant^4_{ws}$ подчеркнуты волнистой линией):

$$\mathbf{I}\leq^5_{ws}\mathbf{S} \triangleq \exists R\subseteq V_{\mathbf{I}}\times V_{\mathbf{S}}\ (\underline{s_0=\langle\gamma\rangle\not\Rightarrow}\Rightarrow (i_0,s_0)\in R)$$
$$\&\ \forall (i,s)\in R\ \underline{\forall P\ \textit{safe}\ i}\ \forall u\in\underline{P\cup\{P\}}\ \forall i`$$
$$(\underline{P\ \textit{safe}\ s}\ \&\ i=\langle u\rangle\Rightarrow i`\ \Rightarrow\ \exists s`\ s=\langle u\rangle\Rightarrow s`\ \&\ (i`,s`)\in R).$$

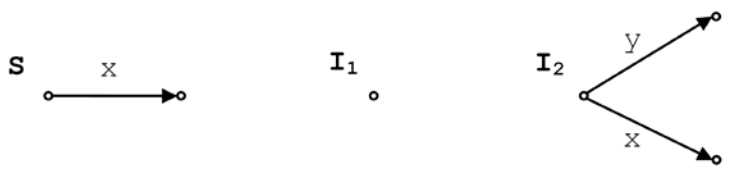
Лемма 5: $\leq^4_{ws}\ \subset\ \leq^5_{ws}$.

Доказательство. Отношение $\leq^4_{ws}$ предъявляет больше требований к реализации: требуется верификация наблюдений, разрешаемых не только безопасными кнопками. □

На классе реализаций и спецификаций, в которых нет ненаблюдаемых отказов, дивергенции и разрушения (то есть все кнопки безопасны), эти соответствия совпадают: $\leq^4_{ws} = \leq^5_{ws}$.

Лемма 6: $\leq^3_{ws}\ \not\subseteq\ \leq^5_{ws}$ и $\leq^3_{ws}\ \not\supseteq\ \leq^5_{ws}$.

Доказательство. Смотри пример на Рис.2. □



$\mathbf{R}=\{\{x\}\}$, $\mathbf{Q}=\{\{y\}\}$

$\mathbf{I}_1 \leq^3_{ws} \mathbf{S}$	$\mathbf{I}_1 \not\leq^5_{ws} \mathbf{S}$
$\mathbf{I}_2 \not\leq^3_{ws} \mathbf{S}$	$\mathbf{I}_2 \leq^5_{ws} \mathbf{S}$

Рис.2. Соотношение симуляций $\leq^3_{ws}$ и $\leq^5_{ws}$

Лемма 7: Симуляция $\leq^5_{ws}$ рефлексивна, но не транзитивна.

Доказательство. Для доказательства рефлексивности симуляции достаточно взять тождественное соответствие $R=\{(s,s)\mid s\in V_S\}$. Пример нетранзитивности симуляции на Рис.3. □

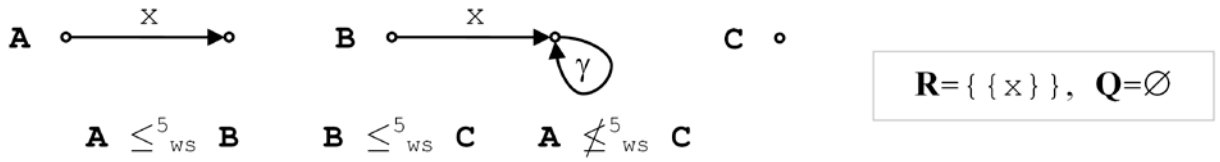


Рис.3. Нетранзитивность отношения $\leq^5_{ws}$

2.6. Гипотеза о безопасности

Поскольку спецификация задана, мы можем проверять по ней условие P ***safe*** s. Но при тестировании (в отличие от аналитической верификации) реализация неизвестна, и судить о безопасности кнопок в состояниях реализации (P ***safe*** i) мы можем только на основании некоторой *гипотезы о безопасности*. В данной работе такая гипотеза основана на некотором соответствии $H\subseteq V_I\times V_S$ состояний реализации и спецификации[4]. Мы будем называть ее H-*гипотезой* о безопасности. Она предполагает 1) безопасность начального состояния i_0 реализации, если безопасно начальное состояние s_0 спецификации, 2) безопасность кнопки в состоянии реализации, если она безопасна хотя бы в одном в соответствующем по H состоянии спецификации.

Определим соответствие H рекурсивно. Начальные состояния i_0 и s_0 соответствуют друг другу, если они оба безопасны. Также соответствуют друг другу любые два состояния, достижимые из безопасных начальных состояний по пустой трассе. Два состояния $i`$ и $s`$ соответствуют друг другу, если они достижимы из соответствующих друг другу состояний i и s по одному и тому же наблюдению u, разрешаемому кнопкой P, которая безопасна в обоих состояниях i и s. Формально соответствие H определяется как минимальное соответствие, порождаемое следующими правилами вывода:

$\forall i,i`\in V_I\ \forall s,s`\in V_S\ \forall P\in \mathbf{R}\cup\mathbf{Q}\ \forall u\in P\cup\{P\}$

$s_0=\langle\gamma\rangle\not\Rightarrow\ \&\ i_0=\langle\gamma\rangle\not\Rightarrow\ \&\ i_0\Rightarrow i\ \&\ s_0\Rightarrow s \quad\vdash (i,s)\in H,$

$(i,s)\in H\ \&\ P$ ***safe*** $i\ \&\ P$ ***safe*** $s\ \&\ i=\langle u\rangle\Rightarrow i`\ \&\ s=\langle u\rangle\Rightarrow s` \quad\vdash (i`,s`)\in H.$

[4] Мнемоника: H – **H**ypothesis.

Соответствие H похоже на соответствие R слабой симуляции $\leq^5_{ws}$, но имеет три существенных отличия. 1) Начальные состояния i_0 и s_0 соответствуют друг другу по R, если безопасно состояние s_0, а соответствие H требует безопасности обоих состояний i_0 и s_0. 2) Постсостоянию i` после наблюдения u, разрешаемому кнопкой P, которая безопасна в обоих соответствующих состояниях i и s, по R должно соответствовать хотя бы одно постсостояние s`, а по H – каждое постсостояние. 3) Соответствие H минимальное, то есть состояния соответствуют друг другу только в том случае, когда они достижимы из безопасных начальных состояний нажатием одних и тех же безопасных кнопок; соответствие R может содержать пары состояний, которые недостижимы таким образом (даже вообще недостижимы из начальных состояний).

Кнопку P будем называть H-*безопасной* в реализационном состоянии i, если она безопасна хотя бы в одном соответствующем i спецификационном состоянии s:

P H-***safe*** $i \triangleq \exists s\ (i,s)\in H\ \&\ P$ ***safe*** s.

Теперь дадим формальное определение H-гипотезы:

$\mathbf{I}$ H-***safe*** $\mathbf{S} \triangleq (s_0{=}\langle\gamma\rangle{\not\Rightarrow} \Rightarrow i_0{=}\langle\gamma\rangle{\not\Rightarrow})$

$\&\ \forall P\in\mathbf{R}\cup\mathbf{Q}\ (P$ H-***safe*** $i \Rightarrow P$ ***safe*** $i)$.

Реализацию, удовлетворяющую H-гипотезе, будем называть H-*безопасной*.

Лемма 8: Отношение безопасности H-***safe*** не рефлексивно и не транзитивно.

Доказательство. Пример нерефлексивности на Рис.4. Пример нетранзитивности на Рис.5. □

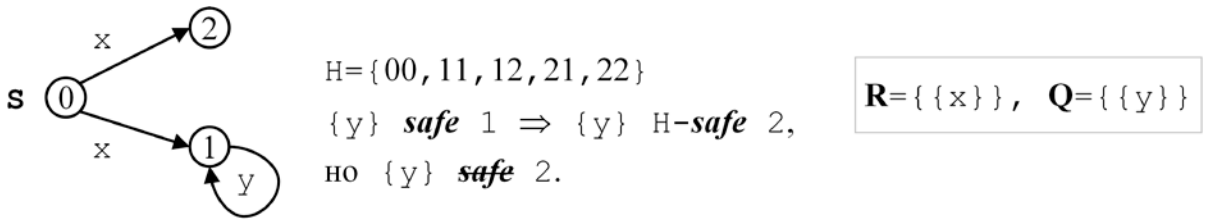


Рис.4. Нерефлексивность отношения безопасности

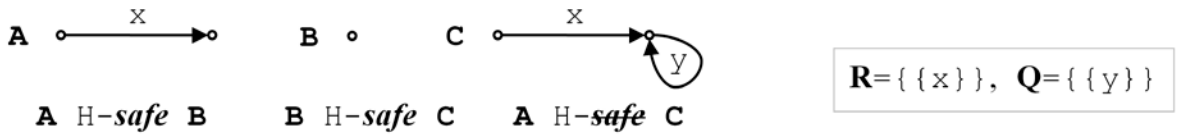


Рис.5. Нетранзитивность отношения безопасности

Если H-гипотеза выполнена, то либо $H=\varnothing$, если $s_0=\langle\gamma\rangle\Rightarrow$, либо $(i_0,s_0)\in H$ в противном случае.

При условии выполнения H-гипотезы определим маршруты реализации, которые могут быть пройдены при безопасном тестировании (H-безопасные маршруты), и состояния реализации, которые при этом могут достигаться (H-достижимые состояния). Маршрут реализации H-*безопасен*, если начальное состояние спецификации безопасно (тем самым по H-гипотезе начальное состояние реализации тоже безопасно) и каждый переход $i—u→i`$ маршрута, кроме τ-переходов, помечен наблюдением u, разрешаемым кнопкой, которая H-безопасна в пресостоянии i этого перехода (то есть безопасна в некотором состоянии s спецификации, соответствующем по H состоянию i, и тем самым по H-гипотезе безопасна в состоянии i). Состояние реализации H-*достижимо*, если оно достижимо по H-безопасному маршруту, то есть является его постсостоянием.

Заметим, что пресостоянию i каждого перехода $i—u→i`$ H-безопасного маршрута соответствует по $H$ хотя бы одно состояние спецификации, а для постсостояния $i`$ это не обязательно верно, если это не пресостояние следующего перехода (постсостояние маршрута). Соответствие H естественным образом индуцирует соответствие $H`$ маршрутов реализации и спецификации. Во-первых, каждому переходу по наблюдению одного маршрута соответствует переход по тому же наблюдению другого маршрута, причем это наблюдение разрешается кнопкой, которая безопасна в пресостояниях этих переходов как в спецификации, так и в реализации. Во-вторых, каждому состоянию одного маршрута соответствует по H некоторое состояние другого маршрута таким образом, что, если пресостояния переходов маршрутов соответствуют друг другу, то постсостояния этих переходов тоже соответствуют друг другу. Все реализационные маршруты, имеющие по $H`$ соответствующие спецификационные маршруты, являются H-безопасными. Продолжения таких реализационных маршрутов, получаемые нажатием кнопок, которые H-безопасны в постсостояниях маршрутов, могут не иметь по

H` соответствующих спецификационных маршрутов, но также являются H-безопасными[5].

2.7. Безопасная симуляция

Теперь, соединяя H-гипотезу о безопасности и слабую симуляцию, мы получаем окончательный вариант слабой симуляции с отказами и безопасностью (изменения по сравнению с $\leqslant^5_{ws}$ подчеркнуты волнистой линией), которую будем называть *безопасной симуляцией* и обозначать ***ss***:[6]

I ***ss*** **S** $\triangleq$ **I** H-***safe*** **S** & $\exists R \subseteq V_I \times V_S$ $(s_0 = \langle\gamma\rangle \not\Rightarrow \Rightarrow (i_0, s_0) \in R)$
& $\forall (i,s) \in R$ $\forall P$ H-***safe*** i $\forall u \in P \cup \{P\}$ $\forall i`$
(P ***safe*** s & $i = \langle u \rangle \Rightarrow i` \Rightarrow \exists s`$ $s = \langle u \rangle \Rightarrow s`$ & $(i`, s`) \in R$).

Если реализация задана явно, то можно аналитически проверять как H-гипотезу, так и безопасную симуляцию. Когда реализация неизвестна, верификация симуляции выполняется с помощью тестирования. В этом случае H-гипотеза является предусловием безопасного тестирования.

Если $s_0 = \langle\gamma\rangle \Rightarrow$, то $H = \varnothing$, безопасное тестирование невозможно, но и не нужно, так как любая реализация конформна (при любом R). Если $s_0 = \langle\gamma\rangle \not\Rightarrow$, то тестирование заключается в проверке *тестируемого условия* (нижние две строки определения ***ss***). Если существует кнопка P H-***safe*** i, то состояние i H-достижимо. Для каждого H-достижимого состояния i нажимается каждая H-безопасная в нем кнопка P, и полученные наблюдение u и постсостояние i` верифицируются по спецификации: наблюдение u должно быть в каждом соответствующем ему по R состоянии s, в котором кнопка P безопасна, а среди постсостояний s` хотя бы одно должно соответствовать i` по R.

Лемма 9: $H\text{-}\mathit{safe} \cap \leq^5_{ws} \subset \mathit{ss}$.

Доказательство. Отношение $\leq^5_{ws}$, во-первых, не требует выполнения H-гипотезы о безопасности, и, во-вторых, требует верификации наблюдений, разрешаемых не только H-безопасными кнопками, а любыми кнопками,

[5] Эти продолжения аналогичны безопасным продолжениям безопасных трасс (то есть тестовым трассам) в теории трассовой конформности [5,6].
[6] Мнемоника: ***ss*** – **S**afe **S**imulation.

безопасными как в реализационном, так и в соответствующем ему по R спецификационном состояниях. □

Для класса спецификаций без ненаблюдаемых отказов, дивергенции и разрушения (когда все кнопки безопасны в спецификации), имеем: H-***safe*** $\cap$ $\leq^5_{ws}$ = ***ss***, а на поддомене безопасных реализаций $\leq^5_{ws}$ = ***ss***.

Лемма 10: Отношение ***ss***, вообще говоря, нерефлексивно, но на классе спецификаций, удовлетворяющих собственной H-гипотезе, рефлексивно: $\forall$**S S** H-***safe*** **S** $\Rightarrow$ **S** ***ss*** **S**.

Доказательство. Нерефлексивность ***ss*** в общем случае следует из нерефлексивности отношения H-***safe*** (лемма 8). Рефлексивность ***ss*** для спецификаций, удовлетворяющих собственной гипотезе о безопасности, следует из леммы 9 и рефлексивности отношения $\leq^5_{ws}$. □

Лемма 11: Для конформного по ***ss*** соответствия R соответствие R$\cap$H тоже конформно.

Доказательство. Если $s_0=\langle\gamma\rangle\Rightarrow$, то H=$\varnothing$ и любая реализация конформна при любом соответствии R, в частности при R$\cap\varnothing=\varnothing$. Если $s_0=\langle\gamma\rangle\not\Rightarrow$, то, выполнение H-гипотезы влечет $(i_0,s_0)\in H$, а конформность соответствия R влечет $(i_0,s_0)\in R$, следовательно, $(i_0,s_0)\in R\cap H$. Пусть для некоторых состояний, кнопки и наблюдения выполняется: $(i,s)\in R\cap H$ & P H-***safe*** i & $u\in P\cup\{P\}$ & P ***safe*** s & $i=\langle u\rangle\Rightarrow i`$. Тогда по H-гипотезе P H-***safe*** i влечет P ***safe*** i, а по конформности соответствия R, существует такое состояние s`, что $s=\langle u\rangle\Rightarrow s`$ & $(i`,s`)\in R$. Тем самым выполнены условия второго правила вывода в определении соответствия H. Следовательно, $(i`,s`)\in H$. В результате имеем $(i`,s`)\in R\cap H$, что и требовалось доказать. □

Эта лемма позволяет переформулировать определение безопасной симуляции следующим образом:

I ***ss*** **S** $\triangleq$ **I** H-***safe*** **S** & $\exists R\subseteq H$ $(s_0=\langle\gamma\rangle\not\Rightarrow \Rightarrow (i_0,s_0)\in R)$
& $\forall(i,s)\in R$ $\forall$P ***safe*** s $\forall u\in P\cup\{P\}$ $\forall i`$
$(i=\langle u\rangle\Rightarrow i` \Rightarrow \exists s`\ s=\langle u\rangle\Rightarrow s` \ \&\ (i`,s`)\in R)$.

Лемма 12: Объединение конформных по ***ss*** соответствий конформно.

Доказательство. Тривиально.

Из последних двух лемм следует, что можно ограничиться только такими соответствиями R, которые вложены в H. Мы имеем два естественных конформных соответствия: R_1 – объединение всех конформных соответствий, и $R_2=R_1\cap H$. Ниже в разделе, посвященном практическому тестированию конформности ***ss***, мы покажем, как можно при некоторых ограничениях в процессе тестирования строить такие конформные соответствия или доказывать их несуществование для заданной спецификации и любой фиксированной, но неизвестной реализации, удовлетворяющей H-гипотезе.

Лемма 13: Отношение ***ss*** транзитивно.

Доказательство. Пусть **A** ***ss*** **B** и **B** ***ss*** **C**.
Обозначим соответствующие отношения H и конформные соответствия через H_{AB}, H_{BC}, H_{AC}, R_{AB}, R_{BC}. По лемме 11 будем считать, что $R_{AB}\subseteq H_{AB}$, $R_{BC}\subseteq H_{BC}$.
Обозначим $R_{AC}=R_{AB}\circ R_{BC}=\{(a,c)\mid\exists b\ (a,b)\in R_{AB}\ \&\ (b,c)\in R_{BC}\}$.

Если $c_0=\langle\gamma\rangle\Rightarrow$, то любая реализация конформна спецификации **C**, в частности **A** ***ss*** **C**. Далее будем считать, что $c_0=\langle\gamma\rangle\not\Rightarrow$.

Сначала покажем, что **A** H-***safe*** **C**. Так как $c_0=\langle\gamma\rangle\not\Rightarrow$, то по конформности соответствия R_{BC} имеем $b_0=\langle\gamma\rangle\not\Rightarrow$, а тогда по конформности соответствия R_{AB} имеем $a_0=\langle\gamma\rangle\not\Rightarrow$.

Теперь нам нужно доказать, что, если $(a,c)\in H_{AC}$ & P ***safe*** c, то P ***safe*** a. Мы докажем более сильное утверждение, включающее дополнительное требование: $\exists b\ (a,b)\in H_{AB}\ \&\ (b,c)\in H_{BC}$.

Сначала покажем, что утверждение верно для пары (a,c), полученной по 1-ому правилу вывода для H_{AC}. Действительно, если $c_0=\langle\gamma\rangle\not\Rightarrow$, $a_0=\langle\gamma\rangle\not\Rightarrow$, $a_0\Rightarrow a$ и $c_0\Rightarrow c$, то по H_{BC}-безопасности B имеем $b_0=\langle\gamma\rangle\not\Rightarrow$. 1-ое правило вывода применимо к паре (a,b_0), то есть $(a,b_0)\in H_{AB}$. Также 1-ое правило

вывода применимо к паре (b_0,c), то есть $(b_0,c)\in H_{BC}$. Тогда по H_{BC}-безопасности B имеем P ***safe*** b_0, а отсюда по H_{AB}-безопасности A имеем P ***safe*** a.

Теперь предположим, что утверждение верно для пары (a,c) и выполнены условия 2-го правила вывода: $(a,c)\in H_{AC}$ & P ***safe*** a & P ***safe*** c & $a=\langle u\rangle\Rightarrow a'$ & $c=\langle u\rangle\Rightarrow c'$. Докажем утверждение для пары $(a',c')\in H_{AC}$ & P' ***safe*** c'. По предположению $\exists b$ $(a,b)\in H_{AB}$ & $(b,c)\in H_{BC}$. Тогда по H_{BC}-безопасности B имеем P ***safe*** b, а тогда по R_{AB}-конформности существует b' такое, что $b=\langle u\rangle\Rightarrow b'$ & $(a',b')\in R_{AB}\subseteq H_{AB}$. Тогда по 2-ому правилу вывода для H_{BC} имеем $(b',c')\in H_{BC}$. Тогда по H_{BC}-безопасности B имеем P' ***safe*** b', а тогда по H_{AB}-безопасности A имеем P' ***safe*** a'.

Утверждение о том, что **A** H-***safe*** **C**, доказано. Теперь докажем, что для **A** и **C** выполнено тестируемое условие конформности ***ss*** для соответствия R_{AC}. Действительно, пусть $(a,c)\in R_{AC}$, P ***safe*** c, $u\in P\cup\{P\}$ и $a=\langle u\rangle\Rightarrow a'$. Тогда существует b такое, что $(a,b)\in R_{AB}$ и $(b,c)\in R_{BC}$. По H_{BC}-безопасности B имеем P ***safe*** b, а тогда по R_{AB}-конформности A существует b' такое, что $b=\langle u\rangle\Rightarrow b'$ и $(a',b')\in R_{AB}$. Но по R_{BC}-конформности B существует c' такое, что $c=\langle u\rangle\Rightarrow c'$ и $(b',c')\in R_{BC}$. Тем самым, $(a',c')\in R_{AC}$. □

Суммируя леммы 1÷13, получаем следующую теорему, описывающую соотношение различных симуляций:

Теорема 1:
1) $\leq^4_{ws}\subset\leq^3_{ws}=\leq^2_{ws}=\leq^1_{ws}$.
2) $\leq^4_{ws}\subset\leq^5_{ws}$.
3) $\leq^3_{ws}\not\subseteq\leq^5_{ws}$ и $\leq^3_{ws}\not\supseteq\leq^5_{ws}$.
4) Симуляции $\leq^k_{ws}$, где $k=1\div4$, являются предпорядками.
5) Симуляция $\leq^5_{ws}$ рефлексивна, но не транзитивна.
6) Отношение H-***safe*** нерефлексивно и нетранзитивно.
7) H-***safe*** $\cap\leq^5_{ws}\subset$ ***ss***.
8) Отношение ***ss***, вообще говоря, нерефлексивно, но на классе спецификаций, удовлетворяющих собственной H-гипотезе, рефлексивно.

9) Отношение ***ss*** транзитивно и на классе спецификаций, удовлетворяющих собственной H-гипотезе, является предпорядком.

3. Связь симуляции с трассовой конформностью

3.1. Трассовые гипотеза о безопасности и конформность

В трассовой теории конформности гипотеза о безопасности основывалась на трассах реализации и спецификации [1,2,3,4] и не требовала соответствия состояний реализации и спецификации. Напомним основные определения этой теории.

Для реализации $\mathbf{I}$ определяется отношение ***safe in*** безопасности кнопки $P \in \mathbf{R} \cup \mathbf{Q}$ после трассы $\sigma \in \boldsymbol{T}(\mathbf{I})$:

P ***safe in*** $\mathbf{I}$ ***after*** $\sigma \triangleq (P \in \mathbf{R} \vee \sigma \cdot \langle P \rangle \notin \boldsymbol{T}(\mathbf{I}))$

$\& \ \forall z \in P \ \sigma \cdot \langle z, \gamma \rangle \notin \boldsymbol{T}(\mathbf{I}) \ \& \ \sigma \cdot \langle \Delta \rangle \notin \boldsymbol{T}(\mathbf{I})$.

Очевидно, что если кнопка безопасна по ***safe in*** после трассы (P ***safe in*** $\mathbf{I}$ ***after*** σ), то она безопасна в каждом состоянии после этой трассы: $\forall i \in (\mathbf{I}$ ***after*** $\sigma)$ P ***safe*** i.

Для спецификации отношение ***safe by*** безопасности кнопок после трасс определяется неоднозначно: это любое отношение, удовлетворяющее трем требованиям: $\forall \sigma \in \boldsymbol{T}(\mathbf{S}) \ \forall R \in \mathbf{R} \ \forall z \in \mathbf{L} \ \forall Q \in \mathbf{Q}$

1) R ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Leftrightarrow \forall u \in R \ \sigma \cdot \langle u, \gamma \rangle \notin \boldsymbol{T}(\mathbf{S}) \ \& \ \sigma \cdot \langle \Delta \rangle \notin \boldsymbol{T}(\mathbf{S})$,

2) $\sigma \cdot \langle z \rangle \in \boldsymbol{T}(\mathbf{S}) \ \& \ \exists T \in \mathbf{R} \cup \mathbf{Q} \ z \in T \ \& \ \forall u \in T \ \sigma \cdot \langle u, \gamma \rangle \notin \boldsymbol{T}(\mathbf{S}) \ \& \ \sigma \cdot \langle \Delta \rangle \notin \boldsymbol{T}(\mathbf{S})$
$\Rightarrow \exists P \in \mathbf{R} \cup \mathbf{Q} \ z \in P \ \& \ P$ ***safe by*** $\mathbf{S}$ ***after*** σ,

3) Q ***safe by*** $\mathbf{S}$ ***after*** σ
$\Rightarrow \exists v \in Q \ \sigma \cdot \langle v \rangle \in \boldsymbol{T}(\mathbf{S}) \ \& \ \forall u \in Q \ \sigma \cdot \langle u, \gamma \rangle \notin \boldsymbol{T}(\mathbf{S}) \ \& \ \sigma \cdot \langle \Delta \rangle \notin \boldsymbol{T}(\mathbf{S})$.

Будем считать, что вместе со спецификацией задано отношение ***safe by***, удовлетворяющее этим трем требованиям.

R-трасса σ спецификации $\mathbf{S}$ называется безопасной, если спецификация не содержит трассу $\langle \gamma \rangle$, а трасса σ не заканчивается на дивергенцию и разрушение, и каждый встречающийся в ней символ u (внешнее действие или **R**-отказ) безопасен после непосредственно предшествующего ему префикса трассы:

$$\mathit{SafeBy}(\mathbf{S}) \triangleq \{\sigma\in\mathit{T}(\mathbf{S}) \mid \langle\gamma\rangle\notin\mathit{T}(\mathbf{S}) \;\&\; \forall\mu,\lambda,u\ (\sigma=\mu\cdot\langle u\rangle\cdot\lambda \Rightarrow u \;\mathit{safe\ by}\; \Sigma \;\mathit{after}\; \mu)\}.$$

Для кнопок $R\in\mathbf{R}$ отношения ***safe by*** и ***safe in*** совпадают. Поэтому, если кнопка $R\in\mathbf{R}$ безопасна по ***safe by*** после трассы (R ***safe by*** $\mathbf{S}$ ***after*** σ), то она безопасна в каждом состоянии после этой трассы: $\forall s\in(\mathbf{S}\ \mathit{after}\ \sigma)$ R ***safe*** s. Однако кнопка $Q\in\mathbf{Q}$, которая безопасна по ***safe by*** после трассы, может быть опасна в некоторых (но не всех) состояниях $s\in(\mathbf{S}\ \mathit{after}\ \sigma)$. Тем не менее, верна следующая лемма:

Лемма 14: Если трасса безопасна по отношению ***safe by*** (в спецификации) или ***safe in*** (в реализации), то все маршруты с этой трассой безопасны (соответственно, в спецификации или в реализации). Обратное, вообще говоря, не верно.

Доказательство. Доказательство будем вести индукцией по трассе. Пустая трасса безопасна, если нет трассы $\langle\gamma\rangle$, а это означает безопасность всех состояний, достижимых по пустой трассе и, следовательно, безопасность всех маршрутов с пустой трассой. Пусть утверждение верно для трассы σ, и докажем его для трассы $\sigma\cdot\langle u\rangle$, где наблюдение u разрешается кнопкой P, которая безопасна после σ по отношению ***safe by*** или ***safe in***. Нам нужно показать, что в каждом состоянии после трассы σ, в котором имеется переход по u (включая виртуальные переходы по отказам из $\mathbf{R}$), кнопка P безопасна.

Действительно, если кнопка $P\in\mathbf{R}$, то отношения ***safe by*** и ***safe in*** для такой кнопки совпадают, и ее безопасность по этим отношениям означает ее безопасность в каждом состоянии после трассы σ. Если кнопка $P\in\mathbf{Q}$, то ее безопасность по ***safe in*** также означает ее безопасность в каждом состоянии после трассы σ. А безопасность кнопки $P\in\mathbf{Q}$ по отношению ***safe by*** означает ее безопасность в каждом состоянии после трассы σ, в котором нет отказа $\{P\}$. Поскольку для $P\in\mathbf{Q}$ должно быть $u\neq\{P\}$, то есть наблюдение u – это внешнее действие (не отказ), кнопка P безопасна в каждом состоянии после трассы σ, в котором есть переход по u.

Из безопасности всех маршрутов с данной трассой, вообще говоря, не следует безопасность этой трассы, что показывается примером на Рис.6. □

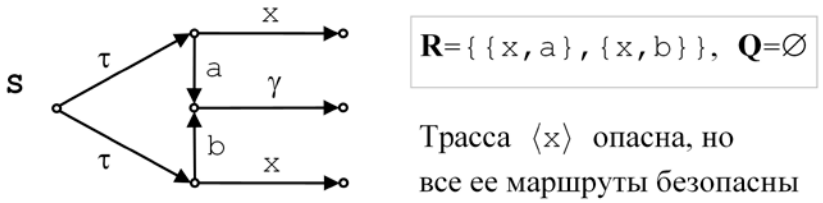


Рис.6. Пример опасной трассы, все маршруты которой безопасны

Трассовая гипотеза о безопасности определялась следующим образом:
$\mathbf{I}$ ***safe for*** $\mathbf{S} \triangleq (\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{S}) \Rightarrow \langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{I}))\ \&\ \forall\sigma\in\boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{T}(\mathbf{I})\ \forall P\in\mathbf{R}\cup\mathbf{Q}$
$(P$ ***safe by*** $\mathbf{S}$ ***after*** $\sigma \Rightarrow P$ ***safe in*** $\mathbf{I}$ ***after*** $\sigma)$.

Трассовая конформность определялась так:
$\mathbf{I}$ ***saco*** $\mathbf{S} \triangleq \mathbf{I}$ ***safe for*** $\mathbf{S}\ \&\ \forall\sigma\in\boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{T}(\mathbf{I})\ \forall P$ ***safe by*** $\mathbf{S}$ ***after*** σ
$\forall i\in(\mathbf{I}$ ***after*** $\sigma)\ \forall u\in P\cup\{P\}\ (i=\langle u\rangle\Rightarrow\ \Rightarrow\ \exists s\in(\mathbf{S}$ ***after*** $\sigma)\ s=\langle u\rangle\Rightarrow)$.

3.2. Связь H-гипотезы с трассовой гипотезой о безопасности

Лемма 15: H-гипотеза предъявляет к реализации более сильные требования, чем трассовая гипотеза о безопасности: H-***safe*** $\subseteq$ ***safe for***.

Доказательство. Пусть $\mathbf{I}$ H-***safe*** $\mathbf{S}$. Покажем, что $\mathbf{I}$ ***safe for*** $\mathbf{S}$.

Сначала покажем, что $\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{S}) \Rightarrow \langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{I})$. Условие $\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{S})$ эквивалентно условию $s_0=\langle\gamma\rangle\not\Rightarrow$, а условие $\langle\gamma\rangle\notin\boldsymbol{T}(\mathbf{I})$ эквивалентно условию $i_0=\langle\gamma\rangle\not\Rightarrow$. По H-гипотезе имеем $s_0=\langle\gamma\rangle\not\Rightarrow\ \Rightarrow\ i_0=\langle\gamma\rangle\not\Rightarrow$.

Теперь нам нужно показать, что если $\sigma\in\boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{T}(\mathbf{I})$ и кнопка P ***safe by*** $\mathbf{S}$ ***after*** σ, то P ***safe in*** $\mathbf{I}$ ***after*** σ. Условие P ***safe in*** $\mathbf{I}$ ***after*** σ эквивалентно условию $\forall i\in(\mathbf{I}$ ***after*** $\sigma)$ P ***safe*** i. Из условия P ***safe by*** $\mathbf{S}$ ***after*** σ следует, что кнопка P ***safe*** s хотя бы для одного состояния $s\in(\mathbf{S}$ ***after*** $\sigma)$. Если $(i,s)\in H$, то по H-гипотезе P ***safe*** i. Поэтому нам достаточно показать, что $\forall\sigma\in\boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{T}(\mathbf{I})$ $\forall s\in(\mathbf{S}$ ***after*** $\sigma)\ \forall i\in(\mathbf{I}$ ***after*** $\sigma)\ (i,s)\in H$.

Будем вести доказательство индукцией по трассе $\sigma\in\boldsymbol{SafeBy}(\mathbf{S})\cap\boldsymbol{T}(\mathbf{I})$. Пустая трасса $\sigma=\langle\rangle$ всегда есть в реализации и безопасна в спецификации, если

$\langle\gamma\rangle \notin \boldsymbol{T}(\mathbf{S})$, то есть $s_0=\langle\gamma\rangle\not\Rightarrow$, а тогда по доказанному $\langle\gamma\rangle \notin \boldsymbol{T}(\mathbf{I})$, то есть $i_0=\langle\gamma\rangle\not\Rightarrow$. А тогда по 1-ому правилу вывода для соответствия H имеем $\forall s\in(\mathbf{S}\ \textit{after}\ \langle\rangle)\ \ \forall i\in(\mathbf{I}\ \textit{after}\ \langle\rangle)\ \ (i,s)\in H$. Пусть утверждение верно для трассы σ и докажем его для трассы $\sigma\cdot\langle u\rangle$, где наблюдение u разрешается кнопкой P ***safe by*** $\mathbf{S}$ ***after*** σ. Пусть $s`\in(\mathbf{S}\ \textit{after}\ \sigma\cdot\langle u\rangle)$ и $i`\in(\mathbf{I}\ \textit{after}\ \sigma\cdot\langle u\rangle)$. Тогда $\exists s\in(\mathbf{S}\ \textit{after}\ \sigma)\ \ P\ \textit{safe}\ s\ \ \&\ \ s=\langle u\rangle\Rightarrow s`$. Также $\exists i\in(\mathbf{I}\ \textit{after}\ \sigma)\ \ i=\langle u\rangle\Rightarrow i`$. По предположению шага индукции $(i,s)\in H$, следовательно, по H-гипотезе $P\ \textit{safe}\ i$. А тогда выполнены условия 2-го правила вывода для соответствия H, и мы имеем $(i`,s`)\in H$, что и требовалось доказать. □

Можно рассматривать симуляцию с трассовой гипотезой о безопасности. Такая симуляция определяется следующим образом (изменения по сравнению с ***ss*** подчеркнуты волнистой линией):

$\mathbf{I}\ \textit{sst}\ \mathbf{S} \triangleq \mathbf{I}\ \underline{\textit{safe for}}\ \mathbf{S}\ \&\ \exists R\subseteq V_{\mathbf{I}}\times V_{\mathbf{S}}\ (s_0=\langle\gamma\rangle\not\Rightarrow\ \Rightarrow\ (i_0,s_0)\in R)$
$\&\ \forall(i,s)\in R\ \ \underline{\forall\sigma\in\textit{SafeBy}(\mathbf{S})\cap\mathbf{I}\ \ \forall P\ \textit{safe by}\ \mathbf{S}\ \textit{after}\ \sigma}\ \ \forall u\in P\cup\{P\}\ \ \forall i`$
$(P\ \textit{safe}\ s\ \&\ \underline{i\in(\mathbf{I}\ \textit{after}\ \sigma)\ \&}\ i=\langle u\rangle\Rightarrow i`\ \Rightarrow\ \exists s`\ s=\langle u\rangle\Rightarrow s`\ \&\ (i`,s`)\in R)$.

Симуляция с H-гипотезой о безопасности предъявляет более сильные требования к реализации, чем симуляция с трассовой гипотезой о безопасности.

Лемма 16: Из симуляции с H-гипотезой о безопасности следует симуляция с трассовой гипотезой о безопасности, но не наоборот: ***ss*** $\subset$ ***sst***.

Доказательство. Из леммы 15 непосредственно следует, что симуляция с H-гипотезой влечет симуляцию с трассовой гипотезой о безопасности: ***ss*** $\subseteq$ ***sst***. Это объясняется тем, что множество кнопок, безопасных в состоянии реализации по ***safe for***, вложено во множество кнопок, H-безопасных в этом состоянии. Иными словами, для ***sst*** нужно выполнять меньше проверок, чем для ***ss***. Но из ***sst***, вообще говоря, не следует ***ss***. Пример на Рис.7. □

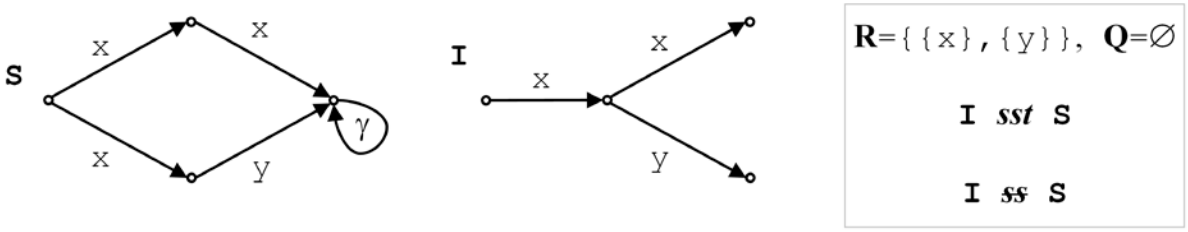


Рис.7. Пример симуляции с трассовой гипотезой о безопасности при отсутствии симуляции с H-гипотезой о безопасности

3.3. Симуляция с трассовой гипотезой о безопасности

Симуляция с трассовой гипотезой о безопасности, в свою очередь, предъявляет более сильные требования к реализации, чем трассовая конформность, основанная на той же гипотезе о безопасности.

Лемма 17: Из безопасной симуляции с трассовой гипотезой о безопасности следует трассовая конформность, но не наоборот: ***sst*** $\subset$ ***saco***.

Доказательство. Пусть $\mathbf{I}$ ***sst*** $\mathbf{S}$ и R – конформное соответствие. Пополним соответствие, добавив в него все пары (i,s_0), где $i_0 \Rightarrow i$, при условии безопасности начального состояния спецификации: $s_0 = \langle\gamma\rangle \not\Rightarrow$. Пополненное соответствие $R`$ также является конформным соответствием. Действительно, нам надо показать, что если для некоторой трассы $\sigma \in \mathbf{SafeBy}(\mathbf{S})$, кнопки P ***safe by*** $\mathbf{S}$ ***after*** σ и наблюдения $u \in P \cup \{P\}$ в реализации для состояния $i \in (\mathbf{I}\ \mathbf{after}\ \sigma)$ имеет место $i = \langle u \rangle \Rightarrow i`$, то в спецификации кнопка $P$ либо опасна в $s_0$, либо найдется такое состояние $s`$, что $s_0 = \langle u \rangle \Rightarrow s`$ и $(i`,s`) \in R$. Поскольку $i_0 \Rightarrow i$ и $i = \langle u \rangle \Rightarrow i`$, имеем $i_0 = \langle u \rangle \Rightarrow i`$. Если $P$ ***safe*** $s_0$, то по конформности соответствия $R$ в спецификации найдется такое состояние $s`$, что $s_0 = \langle u \rangle \Rightarrow s`$ и $(i`,s`) \in R \subseteq R`$, что и требовалось доказать.

Теперь докажем вспомогательное утверждение: если трасса σ безопасна в спецификации, то есть $\sigma \in \mathbf{SafeBy}(\mathbf{S})$, и заканчивается в реализации в состоянии i, то есть $i \in (\mathbf{I}\ \mathbf{after}\ \sigma)$, то $(\mathbf{S}\ \mathbf{after}\ \sigma) \cap R`(i) \neq \varnothing$. Доказательство будем вести индукцией по трассе σ.

Сначала рассмотрим случай пустой трассы $\sigma=\langle\rangle$. Тогда $\langle\rangle\in$***SafeBy*****(S)** влечет $s_0=\langle\gamma\rangle\not\Rightarrow$, что влечет для пополненного соответствия $(i,s_0)\in R`$, то есть $s_0\in R`(i)$. Но также $s_0\in$(**S** ***after*** $\langle\rangle$) и, следовательно, (**S** ***after*** $\langle\rangle$)$\cap R`(i)\neq\varnothing$.

Пусть утверждение верно для трассы σ и докажем его для трассы $\sigma\cdot\langle u\rangle$. Если $i_1\in$(**I** ***after*** $\sigma\cdot\langle u\rangle$) и $\sigma\cdot\langle u\rangle\in$***SafeBy*****(S)**, то найдется такое состояние $i\in$(**I** ***after*** σ) и такая кнопка P ***safe by*** **S** ***after*** σ, что $u\in P\cup\{P\}$ и $i=\langle u\rangle\Rightarrow i_1$. Поскольку $\sigma\cdot\langle u\rangle\in$***SafeBy*****(S)** влечет $\sigma\in$***SafeBy*****(S)**, то по предположению шага индукции найдется состояние $s\in$(**S** ***after*** σ)$\cap R`(i)$. По конформности соответствия $R`$ найдется такое состояние s_1, что $s=\langle u\rangle\Rightarrow s_1$ и $(i_1,s_1)\in R`$, то есть $s_1\in R`(i_1)$. Но также $s\in$(**S** ***after*** σ) и $s=\langle u\rangle\Rightarrow s_1$ влечет $s_1\in$(**S** ***after*** $\sigma\cdot\langle u\rangle$). Тем самым, (**S** ***after*** $\sigma\cdot\langle u\rangle$)$\cap R`(i_1)\neq\varnothing$, что и требовалось доказать.

Из доказанного вспомогательного утверждения непосредственно следует, что безопасная симуляция с трассовой гипотезой о безопасности влечет трассовую конформность. Действительно, если $\sigma\in$***SafeBy*****(S)** и $i\in$(**I** ***after*** σ), то найдется состояние $s\in$(**S** ***after*** σ)$\cap R`(i)$. Тогда, если P ***safe by*** **S** ***after*** $\sigma$, $u\in P\cup\{P\}$ и $i=\langle u\rangle\Rightarrow i`$, то по конформности соответствия $R`$ имеем $s=\langle u\rangle\Rightarrow$, что и требуется для трассовой конформности.

Мы доказали, что ***sst*** $\subseteq$ ***saco***.

Для завершения доказательства леммы заметим, что из трассовой конформности, вообще говоря, не следует симуляция. Пример на Рис.8. □

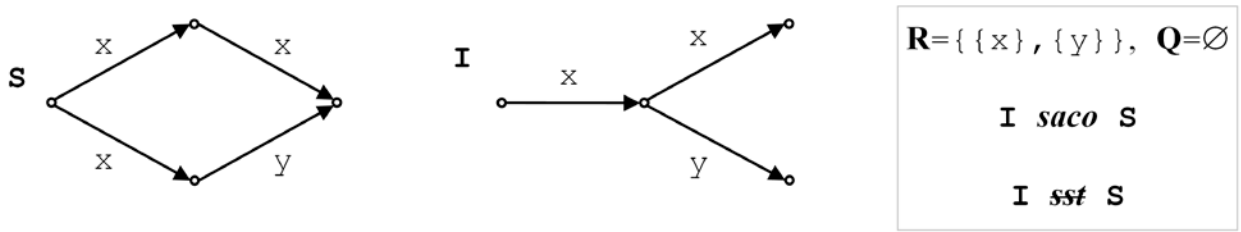


Рис.8. Пример трассовой конформности при отсутствии симуляции с трассовой гипотезой о безопасности

Как итог мы имеем следующую теорему, описывающую соотношение трассовой конформности и безопасных симуляций с различными гипотезами о безопасности.

Теорема 2: ***ss*** $\subset$ ***sst*** $\subset$ ***saco***.

Усиление требований к реализации идет сначала как введение дополнительных требований к соответствию состояний реализации и спецификации для симуляции при сохранении той же гипотезы о безопасности ***sst*** $\subset$ ***saco*** и далее как усиление гипотезы о безопасности ***ss*** $\subset$ ***sst***, что ведет к большему количеству безопасных нажатий кнопок в состояниях и, тем самым, к большему числу проверок.

4. Теоретическое тестирование

4.1. Полнота тестирования

Симуляция основана на соответствии состояний реализации и спецификации. Спецификация должна быть задана и ее состояния мы видим. При тестировании реализация неизвестна, поэтому для проверки симуляции нам нужна специальная операция опроса текущего состояния реализации (status message [7]). Тестирование с такой тестовой возможностью называется тестированием с открытым состоянием. Операция опроса состояния будет применяться в начале тестирования и после каждого наблюдения[7].

Тест – это инструкция, состоящая из пунктов, описывающих тестовые воздействия и получаемые наблюдения и постсостояния, а также указаний по выполнению следующих пунктов или вынесению вердикта (***pass*** или ***fail***). В начале опрашивается состояние реализации, и для каждого возможного состояния указывается следующий пункт. В каждом другом пункте указывается кнопка, которую нужно нажимать, и для каждого наблюдения и постсостояния, возможных после нажатия этой кнопки, – пункт инструкции, который должен выполняться следующим, или вердикт, если тестирование нужно закончить. В [1,2,3] такая инструкция соответствует формальному определению *управляемого* LTS-теста, который однозначно определяет тестирование (без лишнего недетерминизма).

[7] Отказ возникает в стабильном состоянии. Поэтому, если после наблюдения отказа (когда реализация гарантированно оказалась в стабильном состоянии) снова наблюдается отказ (быть может, другой), то состояние можно не опрашивать, поскольку оно не изменяется.

Реализация *проходит* тест, если ее тестирование всегда (то есть при любом проявлении недетерминизма реализации) не заканчивается с вердиктом ***fail***. 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. Основная задача заключается в генерации полного набора тестов по спецификации.

На практике требуется, чтобы каждый тест заканчивался за конечное время, а набор тестов был конечным. К сожалению, это возможно только при некоторых ограничениях на семантику, спецификацию, реализацию и недетерминизм ее выполнения. Иногда требуют конечности времени выполнения теста, но допускают бесконечные наборы тестов. На самом деле это не лучше и не хуже конечного набора тестов с бесконечным выполнением тестов.

Различие между тестом и набором тестов условно, в основном это вопрос удобства построения тестовой системы. Дело в том, что завершение теста и запуск другого (или того же самого) теста эквивалентно выполнению рестарта исследуемой системы в рамках одного теста. Предлагаемые в данной статье алгоритмы тестирования ориентированы на использование одного теста с рестартом системы в любой момент времени.

Прежде всего, для полноты тестирования безопасной симуляции ***ss*** требуется для каждого H-достижимого состояния i реализации и каждой H-безопасной в нем кнопки P верифицировать каждое имеющееся в реализации наблюдение $u \in P \cup \{P\}$ и постсостояние $i`$, то есть маршрут $i=\langle u\rangle\Rightarrow i`$. Для этого предполагается, что такой маршрут можно получить через конечное число нажатий кнопки P в состоянии i. Для того, чтобы можно было попасть из начального состояния реализации в каждое H-достижимое состояние, мы должны предположить, что некоторые маршруты $i=\langle u\rangle\Rightarrow i`$ получаются достаточное число раз (и каждый маршрут – за конечное время). Это предположение называется гипотезой о *глобальном тестировании* [5]. Без каких-либо ограничений на класс реализаций глобальное тестирование, очевидно, требует наблюдения каждого маршрута $i=\langle u\rangle\Rightarrow i`$ бесконечное число раз при бесконечной последовательности нажатий кнопки P в состоянии i. В дальнейшем будем считать, что гипотеза о глобальном тестировании выполняется.

4.2. Дерево вывода неконформности

Для конформности, основанной только на трассах наблюдений и не использующей соответствие состояний, доказано существование полного тестового набора для любой спецификации [3]. Для безопасной симуляции такой полный тестовый набор существует уже не всегда. Исследуем эту проблему.

Если $s_0=\langle\gamma\rangle\Rightarrow$, то все реализации конформны, и тестирование не требуется. Поэтому далее будем считать, что $s_0=\langle\gamma\rangle\not\Rightarrow$. Рассмотрим минимальное множество N пар состояний (i,s), которое порождается следующими правилами вывода: $\forall(i,s)\in H\ \forall P$ H-***safe*** $i\ \forall u\in P\cup\{P\}$

1. $i=\langle u\rangle\Rightarrow$ & P ***safe*** s & $s=\langle u\rangle\not\Rightarrow$ $\vdash (i,s)\in N$,
2. $i=\langle u\rangle\Rightarrow i'$ & P ***safe*** s & $\{i'\}\times(s$ ***after*** $\langle u\rangle)\subseteq N$ $\vdash (i,s)\in N$.

Пары из N будем называть неконформными.

Лемма 18: **I *ss* S** $\Leftrightarrow (i_0,s_0)\notin N$.

Доказательство. Очевидно, что неконформная пара не может принадлежать никакому конформному соответствию. Если $(i_0,s_0)\in N$, то конформного соответствия не существует, так как оно должно было бы содержать (i_0,s_0). Обратно, если $(i_0,s_0)\notin N$, то соответствие $H\backslash N$, очевидно, конформно. □

Заметим, что, если $i_0\Rightarrow i_0'$, то условие $(i_0',s_0)\in N$, очевидно, влечет $(i_0,s_0)\in N$. Поэтому нам достаточно проверить условие $(i_0',s_0)\in N$ для любого состояния $i_0'\in(i_0$ ***after*** $\langle\rangle)$.

Правила вывода для N определяют граф вывода. Вершинами этого графа являются пары $(i,s)\in N$. Если пара (i,s) получена применением 1-го правила вывода, соответствующую вершину будем называть вершиной 1-го типа, в противном случае (применением только 2-го правила вывода) – вершиной 2-го типа. Помеченная дуга (i,s)—$(u,i')\rightarrow(i',s')$ соответствует применению второго правила вывода для i,u,i',s и $s'\in(s$ ***after*** $\langle u\rangle)$. Набор таких дуг для всех $s'\in(s$ ***after*** $\langle u\rangle)$ проводится тогда и только тогда, когда

$i=\langle u\rangle\Rightarrow i` \ \&\ P\ \textbf{\textit{safe}}\ s\ \&\ \{i`\}\times(s\ \textbf{\textit{after}}\ \langle u\rangle)\subseteq N.$

Для каждой пары $(i,s)\in N$ в графе вывода существует (быть может, не единственное) дерево маршрутов, которое мы будем называть деревом вывода. Каждый маршрут, принадлежащий дереву, начинается в (i,s). Корень дерева – пустой маршрут. Листья дерева – маршруты, заканчивающиеся в вершинах 1-го типа. Каждый маршрут, принадлежащий дереву и заканчивающийся в графе вывода в вершине 2-го типа, продолжается в дереве теми и только теми дугами, которые соответствуют одному применению 2-го правила вывода, то есть помеченными одной и той же меткой $(u,i`)$.

Лемма 19: Дерево вывода конечно тогда и только тогда, когда оно имеет конечное ветвление: каждый маршрут продолжается в дереве конечным числом дуг, что эквивалентно конечности каждого множества $s\ \textbf{\textit{after}}\ \langle u\rangle$, участвующего в построении этого дерева.

Доказательство. Количество дуг, которыми продолжается маршрут в дереве вывода, по построению определяется применением 2-ого правила вывода и совпадает с количеством состояний в соответствующем множестве состояний $s\ \textbf{\textit{after}}\ \langle u\rangle$. Если дерево конечно, то, очевидно, оно имеет конечное ветвление и все эти множества состояний конечны. Обратно, если все эти множества состояний конечны, то дерево имеет конечное ветвление. А тогда, поскольку в дереве нет бесконечных маршрутов, оно конечно (по теореме Кёнига [6]). □

4.3. Общий алгоритм тестирования

Если $(i_0,s_0)\in N$, то для (i_0,s_0) существует (быть может, не единственное) дерево вывода. Опишем алгоритм тестирования, определяющий неконформность любой H-безопасной реализации, для которой хотя бы одно из таких деревьев конечно. Этот алгоритм будем называть *общим* алгоритмом тестирования.

Сначала рассмотрим необходимые для работы теста ограничения. Кроме глобального тестирования, нам требуется выполнение следующих условий:

1) начальное состояние спецификации s_0 известно;
2) перечислимо множество спецификационных состояний, достижимых из начального состояния по пустой трассе $S_0=(s_0\ \textbf{\textit{after}}\ \langle\rangle)$, и имеется итератор этого множества;

3) для каждого спецификационного состояния s перечислимо множество безопасных кнопок $P(s)=\{P\in\mathbf{R}\cup\mathbf{Q}\mid P$ ***safe*** $s\}$, и имеется итератор этого множества;
4) для каждого спецификационного состояния s и каждого безопасного в нем наблюдения u перечислимо множество постсостояний маршрутов с трассой $\langle u\rangle$: $S(s,u)=(s$ ***after*** $\langle u\rangle)$, и имеется итератор этого множества.

В каждый момент времени работы алгоритма будет построена некоторая LTS $\mathbf{I}`$, являющаяся представлением некоторой (конечной) части реализации $\mathbf{I}$. Множеством состояний $\mathbf{I}`$ будет конечное множество $V_{\mathbf{I}}`\subseteq V_{\mathbf{I}}$ «пройденных» реализационных состояний. Переход $i—u→i`$ добавляется тогда, когда обнаруживается, что реализация «прошла» маршрут $i=\langle u\rangle\Rightarrow i`$.[8] Вместе с каждым переходом $i—u→i`$ будем хранить *управляющую* кнопку $P(i—u→i`)$, то есть кнопку, нажатие которой вызвало появление этого перехода. По мере построения $\mathbf{I}`$ будет постепенно формироваться для каждого пройденного состояния i множество $H(i)$ соответствующих ему по H спецификационных состояний. Это множество в каждый момент времени будет конечным, но может постепенно расти.

Тест состоит из управляющего алгоритма и нескольких итераторов перечислимых множеств. Набор итераторов динамический и в каждый момент времени конечный: какие-то итераторы появляются и работают постоянно, а некоторые из появившихся могут потом исчезнуть. Одни из итерируемых множеств могут быть бесконечными, но они не меняются в процессе работы, другие множества конечны, но могут увеличиваться. Управляющий алгоритм вызывает работающие итераторы по циклу. Мы будем описывать эти перечислимые множества и их итераторы по мере описания алгоритма тестирования.

[8] Мы говорим о «представлении части», а не просто о «части» реализации, поскольку $i=\langle u\rangle\Rightarrow i`$ означает лишь существование маршрута с такими пре- и пост-состояниями и с такой трассой, а не сам этот маршрут, имеющийся в реализации, который остается неизвестным и может быть, вообще говоря, не единственным. Этот маршрут мы заменяем в $\mathbf{I}`$ на переход $i—u→i`$. Тем самым, алфавит LTS $\mathbf{I}`$ – это объединение $\mathbf{L}\cup\mathbf{R}$ множества внешних действий и **R**-кнопок LTS $\mathbf{I}$.

В начале тестирования, а также после рестарта, опрашиваем состояние и получаем не обязательно начальное состояние реализации i_0, а любое состояние $i_0` \in I_0 = (i_0 \textit{ after } \langle\rangle)$. После опроса состояния $i_0`$, оно добавляется во множество I_0 (до начала тестирования пустое) и во множество $V_I`$ и, если до этого было $i_0` \notin V_I`$, то устанавливаем $H(i_0`) := \varnothing$. Запускается[9] итератор $S_0(i_0`)$, который, итерируя множество $S_0$, добавляет состояния из этого множества к $H(i_0`)$. Будем считать, что в первую очередь добавляется состояние s_0. Если множество S_0 конечно, итератор $S_0(i_0`)$ удаляется после завершения итерации.

Каждый раз, когда добавляется новое состояние s к множеству $H(i)$, запускается итератор $P(i,s)$, итерирующий множество $P(s)$. Итератор $P(i,s)$ перечисляет безопасные в s кнопки и для каждой кнопки P запускает итератор тестового воздействия $T(i,s,P)$ для нажатия кнопки P в состоянии i. Если множество $P(s)$ конечно, итератор $P(i,s)$ удаляется после завершения итерации.

Итератор $T(i,s,P)$ работает бесконечно следующим образом. Сначала делается рестарт, после чего по пройденной части $\mathbf{I}`$ ищется маршрут, ведущий из состояния после рестарта $i_0` \in I_0$ в состояние i. Такой маршрут, очевидно, существует и соответствует H-безопасному маршруту реализации. Итератор пытается пройти такой маршрут, нажимая управляющие кнопки. Каждый раз, когда ему не удается пройти нужный маршрут до конца (из-за недетерминизма реализации), выполняется рестарт и попытка повторяется. Глобальное тестирование гарантирует, что каждый переход $i—u→i`$ в $\mathbf{I}`$ может быть получен за конечное время ($\mathbf{I}`$ может при этом расти, то есть появляться новые переходы), и, следовательно, итератор за конечное время попадет в состояние $i$. После этого нажимается кнопка $P$ и получается переход $i—u→i`$, который добавляется в $\mathbf{I}`$ вместе с кнопкой P как управляющей кнопкой. Итератор $T(i,s,P)$ – это единственный итератор, который взаимодействует с реализацией. По построению для H-безопасной реализации такое взаимодействие безопасно.

Когда итератор $T(i,s,P)$ первый раз проходит переход $i—u→i`$, запускается итератор $S(i,s,u,i`)$, который, итерируя множество $S(s,u)$,

[9] Итератор $S_0(i_0`)$ можно не запускать, если он уже один раз запускался.

добавляет постсостояния s` маршрутов $s=\langle u\rangle\Rightarrow s`$ к множеству H(i`) и запускает при необходимости итератор P(i`,s`). Если множество S(s,u) конечно, итератор S(i,s,u,i`), обнаруживая это (то есть завершение итерации), переходит во второй режим работы – режим проверки.

В режиме проверки итератор S(i,s,u,i`) проверяет, пусто ли множество S(s,u). Если оно пусто, пара (i,s) добавляется к множеству N по 1-ому правилу вывода, а итератор заканчивает свою работу и удаляется. В противном случае итератор опрашивает по циклу состояния из (конечного) множества S(s,u), проверяя для каждого состояния $s`\in S(s,u)$ условие $(i`,s`)\in N$. Если это условие выполнено для всех состояний из S(s,u), пара (i,s) добавляется к множеству N по 2-ому правилу вывода, а итератор заканчивает свою работу и удаляется. В любом случае при удалении итератора S(i,s,u,i`) проверяется условие $i\in I_0$ & $s=s_0$. Если оно выполнено, тест заканчивает свою работу с вердиктом ***fail***. Пока итератор не удален, он работает в цикле по указанному алгоритму (после завершения итерации начинает ее сначала).

Итак, при тестировании работают итераторы: $S_0(i_0`)$, P(i,s), T(i,s,P) и S(i,s,u,i`). Число итераторов в каждый момент времени конечно, и они организованы в список, который перебирается по циклу управляющим алгоритмом, вызывающим итераторы по очереди. Однако список итераторов может расти. Поэтому для того, чтобы каждый итератор делал очередной шаг итерации через конечное время после предыдущего шага, новые итераторы добавляются в начало списка.

Из описания общего алгоритма следует, что тест, работающий по этому алгоритму, выносит вердикт ***fail*** только для таких реализаций, в которых для пары (i_0,s_0) существует конечное дерево вывода.

Теорема 3: Тест, работающий по общему алгоритму, является значимым на классе всех H-безопасных реализаций и полным на подклассе реализаций, в которых для пары (i_0,s_0) дерево вывода либо не существует (реализация конформна), либо конечно (реализация неконформна).

4.4. Достаточное условие полноты тестирования

Тест, работающий по общему алгоритму, не является полным на классе всех H-безопасных реализаций. Он работает бесконечно долго и не выносит вердикт ***fail*** для всех конформных реализаций (что правильно), но также (что

неправильно) и для тех неконформных реализаций, в которых для пары (i_0,s_0) существуют деревья вывода, но все такие деревья бесконечны.

На Рис.9 показан пример, когда реализация неконформна, но все деревья вывода бесконечны. Здесь $\mathbf{I}$ ***ss*** $\mathbf{S}$ для любой **R**/**Q**-семантики (из 22 возможных при алфавите $\mathbf{L}=\{x,y\}$), кроме трех семантик $\{\{x\}\}/\{\{y\}\}$, $\{\{y\}\}/\{\{x\}\}$ и $\varnothing/\{\{x\},\{y\}\}$, то есть для всех семантик таких, что $\{x\},\{y\}\in\mathbf{R}$ или $\{x,y\}\in\mathbf{R}\cup\mathbf{Q}$. Для этих семантик действия x и y безопасны во всех состояниях реализации и спецификации. Состояние 1 не может соответствовать состоянию s_0, так как $1=\langle y\rangle\Rightarrow$, но $s_0=\langle y\rangle\not\Rightarrow$; но оно не может соответствовать и любому другому состоянию спецификации, так как $1=\langle x,x,\ldots\rangle\Rightarrow$ для любого числа действий x, а в спецификации такие трассы есть только в состоянии s_0. Через $\mathbf{I}(n)$ обозначим под-LTS реализации $\mathbf{I}$, содержащую состояния $0,1,\ldots,n$ и все переходы между ними. Легко видеть, что $\mathbf{I}(n)$ ***ss*** $\mathbf{S}$ для каждого n ($\mathbf{I}(n)$ совпадает с частью спецификации $\mathbf{S}$, определяемой одним переходом $s_0 \text{—} x \rightarrow s$). Любой тест за конечное время n может исследовать часть реализации, содержащую не более n состояний. Эта часть реализации, очевидно, является частью конформной LTS $\mathbf{I}(n)$, и поэтому тест не может вынести вердикт о неконформности реализации $\mathbf{I}$ на основе этой ее части, то есть через время n. Тем самым никакой тест не может за конечное время вынести вердикт ***fail***, поэтому любой набор тестов не может быть исчерпывающим.

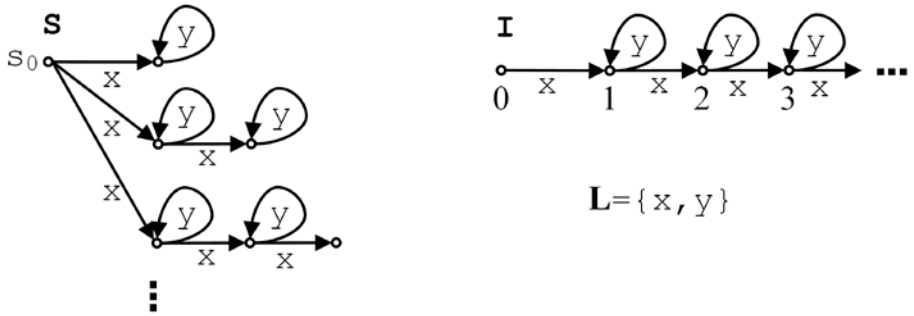


Рис.9. Пример отсутствия полного набора тестов

Теперь мы наложим на спецификацию ограничение, которое достаточно для существования полного теста на классе всех H-безопасных реализаций. Это ограничение *локально-конечно-ветвимости* и *τ-ограниченности* спецификации: 1) в каждом безопасно достижимом состоянии спецификации число переходов по каждому безопасному действию конечно, 2) из каждого такого состояния по τ-маршрутам достижимо конечное число состояний.

Теорема 4: На классе локально-конечно-ветвимых и τ-ограниченных спецификаций тест, работающий по общему алгоритму, полон на классе всех H-безопасных реализаций.

Доказательство. По лемме 19 и теореме 3 нам достаточно показать, что в спецификации все множества состояний s ***after*** $\langle u \rangle$, где $u \in P \cup \{P\}$ и P ***safe*** s конечны для всех безопасно-достижимых состояний s. А такие множества всегда конечны для локально-конечно-ветвящейся и τ-ограниченной LTS-спецификации. □

5. Практическое тестирование

5.1. Ограничения

Алгоритм полного тестирования, описанный в предыдущем разделе, для локально-конечно-ветвящихся и τ-ограниченных LTS-спецификаций обнаруживает ошибку за конечное время, но при отсутствии ошибок может выполняться бесконечно долго. Это не приемлемо на практике. В данном разделе мы сформулируем ограничения на семантику, спецификацию и реализацию, которые позволят выполнять полное тестирование за конечное время, и опишем алгоритм тестирования. Эти ограничения аналогичны тем, которые делают полное тестирование конечным для конформности ***saco***, основанной только на трассах наблюдений без соответствия состояний [4]. Сформулируем эти ограничения:

1) *Ограничения на семантику*: число кнопок конечно и задан алгоритм разрешения кнопки относительно всех действий. Наличие такого алгоритма не означает, что все кнопки (как множества действий) конечны (и, тем самым, конечен алфавит действий). Но для конечных кнопок алгоритм всегда существует.
2) *Ограничения на спецификацию*: LTS-спецификация конечна: конечно число состояний и переходов. В этом случае, очевидно, спецификация локально-конечно-ветвящаяся и τ-ограниченная.
3) *Ограничения на реализацию*: реализация (точнее, ее часть, порождаемая H-безопасными маршрутами) конечна и *ограниченно недетерминирована*.

Ограниченность недетерминизма – это более сильное предположение, чем глобальное тестирование. При глобальном тестировании требуется, чтобы каждое поведение, разрешаемое кнопкой, могло быть получено за *конечное* время, а при ограниченном недетерминизме – за *ограниченное* время. Это 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 $u \in P \cup \{P\}$, постсостояние $i`$). При $t=1$ реализация (наблюдаемо) детерминирована.

5.2. Модификация общего алгоритма тестирования

Сначала покажем, что при этих ограничениях можно так модифицировать общий алгоритм тестирования, описанный в предыдущем разделе, чтобы он всегда заканчивался за конечное время. Поскольку при наших ограничениях спецификация локально-конечно-ветвящаяся и τ-ограниченная, общий алгоритм обнаруживает ошибку в любой неконформной реализации, удовлетворяющей H-гипотезе, за конечное время. Нужно, чтобы он заканчивал свое выполнение за конечное время и для конформных реализаций.

Теорема 5: При указанных в подразделе 5.1 ограничениях на семантику, спецификацию и реализацию общий алгоритм может быть так модифицирован, что тест, работающий по этому алгоритму, полон.

Доказательство. При указанных ограничениях каждый из итераторов $S_0(i_0`)$ и $P(i,s)$ удаляется, а итератор $S(i,s,u,i`)$ переходит в режим проверки через конечное число шагов. При t-недетерминизме реализации итератор $T(i,s,P)$ модифицируется: теперь он выполняет не более t шагов (тестовое воздействие + получение наблюдения и постсостояния). Заметим, что если наблюдается отказ P в том же состоянии, кнопку P можно нажимать в этом состоянии не t раз, а только один раз: ничего другого, кроме отказа P, мы все равно не получим. При указанных ограничениях число всех возможных итераторов конечно. Поскольку в режиме проверки итератор $S(i,s,u,i`)$ не создает новых итераторов, через конечное число шагов останутся только итераторы $S(i,s,u,i`)$, работающие в режиме проверки. Очевидно, что, если за один цикл вызова всех этих итераторов не было изменения множества N (или, что эквивалентно, ни один итератор не был удален), то это множество уже больше не изменится. Управляющий алгоритм модифицируется так, чтобы в этом случае он заканчивал работу теста с вердиктом ***pass***. □

5.3. Алгоритм обхода реализации

Общий алгоритм выполняет «исследование» реализации и верификацию симуляции параллельно, поскольку для бесконечной реализации нельзя ее сначала исследовать (за конечное время), а потом провести верификацию. Теперь мы опишем частный алгоритм тестирования, работающий только при ограничениях из подраздела 5.1, но зато более быстрый, и дадим оценку его

сложности. Будем использовать обозначения, введенные для общего алгоритма. Из конечности семантики и реализации следует конечность LTS $\mathbf{I}`$, порождаемой H-безопасными маршрутами реализации $\mathbf{I}$. Из конечности спецификации следует конечность всех множеств $H(i)$ для состояний $i$ из $\mathbf{I}`$. Поэтому мы можем сначала построить LTS $\mathbf{I}`$ и множества $H(i)$ (будем называть это *обходом реализации*), а потом провести верификацию симуляции.

С каждым пройденным состоянием i свяжем, кроме множества $H(i)$, которое будет пополняться постепенно, множество $P(i)=\cup\{P(s)|s\in H(i)\}$ кнопок безопасных хотя бы в одном состоянии из $H(i)$, а с каждой такой кнопкой P – счетчик $C(i,P)$ числа нажатий кнопки P в состоянии i.

Будем называть кнопку $P\in P(i)$ *полной в состоянии* i, если 1) $c(P,i)=1$ и в $\mathbf{I}`$ получен переход $i—\langle P\rangle\rightarrow i$ или 2) $c(P,i)=t$. Это означает, что уже получены все возможные наблюдения и постсостояния при нажатии кнопки P в состоянии i. В случае 1 после первого нажатия кнопки P в состоянии i наблюдался отказ P в этом же состоянии, повторное нажатие кнопки P даст тот же отказ. В случае 2 после t нажатий кнопки получены все возможные переходы. Состояние i будем называть *полным*, если каждая кнопка из $P(i)$ полна в нем. Это означает, что все возможные переходы $i—\langle u\rangle\rightarrow i`$, где $u\in\mathbf{L}\cup\mathbf{R}$, уже получены.

Общая схема обхода реализации изображена на Рис.10. В начале тестирования после опроса состояния $i_0`\in I_0$ имеем: $H(i_0`)=S_0$, $P(i_0`)=\cup\{P(s)|s\in S_0\}$, $C(i_0`,P)=\varnothing$ для каждой кнопки $P\in P(i_0`)$.

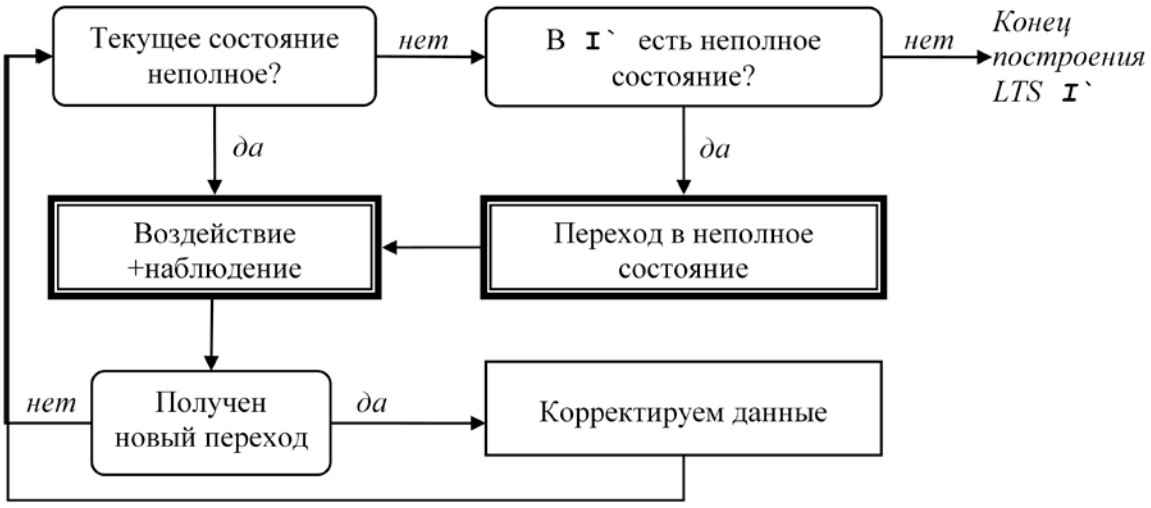


Рис.10. **Схема алгоритма обхода реализации**

Пусть тест находится в некотором текущем состоянии `i` из **`I`**`. Если текущее состояние `i` неполное, то некоторая кнопка `P∈P(i)` неполна в `i`. В этом случае осуществляем тестовое воздействие и наблюдение: нажимаем кнопку `P` и получаем переход `i—⟨u⟩→i`, постсостояние `i`` становится новым текущим состоянием. Увеличиваем счетчик `c(P,i):=c(P,i)+1`. Если получен новый переход `i—⟨u⟩→i`, то добавляем его в **`I`**`, запоминаем кнопку `P` как управляющую кнопку `P(i—u→i`)` этого перехода, и корректируем `H(i`):=H(i`)∪S(s,u)` для каждого `s∈H(i)` при условии `P` ***safe*** `s`.

Каждый раз, когда новое состояние `s` добавляется во множество `H(i)`, корректируем `P(i):=P(i)∪P(s)` и для каждого полученного ранее перехода `i—⟨u⟩→i`` при условии `P(i—⟨u⟩→i`)` ***safe*** `s` корректируем `H(i`):=H(i`)∪S(s,u)`, отмечая вновь добавленные состояния. Эта рекурсивная процедура повторяется до тех пор, пока возможно. В силу конечности реализации и спецификации процедура закончится за конечное число шагов.

Если текущее состояние `i` полное, то в нем все тестовые воздействия выполнены нужное число раз и получены все возможные переходы в **`I`**`. Для продолжения тестирования нужно перейти в любое неполное состояние. Если таких состояний нет, построение **`I`**` заканчивается.

Рассмотрим переход в неполное состояние. В LTS **`I`**` всегда существует лес деревьев, ориентированных к своим корням, которыми являются все неполные состояния. Этот лес покрывает все состояния. Выберем любой такой лес и для каждого его перехода `i—u→i`` обозначим связанную с ним управляющую кнопку через `A(i)=P(i—u→i`)`. Будем двигаться, нажимая в каждом текущем состоянии `i` кнопку `A(i)`. Из-за недетерминизма мы можем оказаться не в состоянии `i`, а в другом состоянии `i``, где будем нажимать кнопку `A(i``)`.

Обозначим: `b` – число кнопок,
`n` – число состояний реализации,
`m` – число переходов реализации,

k – число состояний спецификации.

Заметим, что $m=O(bnt)$.

Лемма 20: Алгоритм обхода реализации, описанный в данном подразделе, при указанных в 5.1 ограничениях заканчивает свою работу за конечное время и строит LTS **I`**, порождаемую H-безопасными маршрутами реализации **I**, и множества $H(i)$ для всех состояний из **I`**. При этом число тестовых воздействий имеет оценку $O(bt^n)$ для $t>1$ и $O(bn^2)$ для $t=1$. Объем вычислений имеет оценку $O(bnt^n)+O(bnm)+O(mk)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $O(bn^3)$.

Доказательство. В силу конечности семантики и реализации, а также ограниченности недетерминизма реализации, блок «Воздействие+наблюдение» вызывается конечное число раз. Поскольку каждый цикл в схеме алгоритма содержит блок «Воздействие+наблюдение», любой другой блок алгоритма также вызывается конечное число раз. При вызове каждого блока, кроме перехода в неполное состояние и коррекции данных, выполняется, очевидно, конечное число действий. Каждый переход в неполное состояние также выполняется за конечное число шагов, как показано в [4]. Рекурсивная процедура коррекции данных работает конечное время в силу конечности реализации и спецификации. Тем самым алгоритм заканчивает свою работу за конечное время.

Алгоритм заканчивается, когда все пройденные состояния стали полными. В этом случае, очевидно, все H-достижимые состояния реализации и H-безопасные переходы из них уже получены, то есть построенная LTS **I`** порождена всеми H-безопасными маршрутами реализации. Все множества $H(i)$ для всех состояний из **I`** также построены, поскольку коррекция данных завершена.

Число тестовых воздействий определяется главным образом временем работы блока перехода в неполное состояние. Хотя один такой переход может потребовать $O(t^n)$ для $t>1$ и $O(n)$ для $t=1$ тестовых воздействий, суммарное число тестовых воздействий, как показано в [7], имеет оценку для $t>1$ не $O(nt^n)$, а $O(bt^n)$, и $O(bn^2)$ для $t=1$.

Оценка объема вычислений состоит из трех слагаемых.

1) Число тестовых воздействий умножается на n для оценки числа операций, требуемых для поиска полученного постсостояния среди уже имеющихся состояний. Суммарная оценка: $O(bnt^n)$ для $t>1$ и $O(bn^3)$ для $t=1$.

2) Построение леса деревьев. Лес деревьев строится не более bn раз, так как каждое состояние становится полным только в тот момент времени, когда в нем становится полной некоторая кнопка[10]. Как показано в [7], лес деревьев строится за $O(m)$ операций. Суммарная оценка $O(bnm)$.
3) Коррекция данных при получении нового перехода (рекурсивная процедура). Для каждого перехода $i—u→i`$ и каждого состояния $s∈H(i)$ один раз выполняется следующее корректирующее действие: состояния из $S(s,u)$ добавляются в $H(i`)$ и определяются те из них, которые ранее не принадлежали $H(i`)$. Сложность корректирующего действия зависит от представления данных. Заметим, что множества $S(s,u)$ определяются только спецификацией и могут быть подготовлены заранее до начала тестирования. Корректирующее действие может быть выполнено за $O(1)$ операций, если множества состояний $S(s,u)$ и $H(i`)$ задаются в виде битовых шкал[11]. Число пар $(i—u→i`,s)$ имеет оценку $O(mk)$. Поэтому суммарная оценка равна $O(mk)$.

Итоговая оценка объема вычислений $O(bnt^n)+O(bnm)+O(mk)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $O(bn^3)$. □

5.4. Два алгоритма верификации симуляции

Мы опишем два алгоритма верификации симуляции после построения LTS $\mathbf{I}`$. Каждый из них пытается построить конформное соответствие R, выдавая вердикт ***pass***, если такое соответствие существует и построено, или вердикт ***fail***, если такого соответствия быть не может. Один из этих алгоритмов строит наибольшее конформное соответствие R_1, а другой – соответствие $R_2=R_1∩H$.

Начнем со второго алгоритма. Сначала строится двудольный граф. Вершины первого типа – это пары (i,s), где i – состояние $\mathbf{I}`$, а $s∈H(i)$. Вершины второго типа – это пары $(i—u→i`,s)$, где $i—u→i`$ – переход $\mathbf{I}`$, а $s∈H(i)$. В каждую вершину второго типа входит одна дуга (дуга первого

[10] Заметим, что полное состояние i может впоследствии стать неполным, если в $P(i)$ добавляется новая H-безопасная кнопка. Однако это не может случиться после того, как все пройденные состояния стали полными, и коррекция данных завершена.
[11] Если множество $S(s,u)$ задано в виде списка, а множество $H(i`)$ – в виде битовой шкалы, то потребуется $O(k)$ операций. Если оба множества заданы в виде списка, то потребуется $O(k^2)$ операций.

типа) $(i,s)\to(i—u\to i`,s)$. Дуга второго типа $(i—u\to i`,s)\to(i`,s`)$ проводится тогда, когда $s`\in S(s,u)$. Одновременно составляется список терминальных вершин второго типа.

После построения двудольного графа начинается собственно верификация. Каждая терминальная вершина v_2 второго типа удаляется вместе с входящей в нее дугой $v_1\to v_2$, начальной вершиной v_1 этой дуги – вершиной первого типа, и каждой входящей в нее дугой $v_1`\to v_1$. Одновременно для $v_1=(i,s)$ состояние $s$ удаляется из множества $H(i)$. Эти операции повторяются до тех пор, пока не будет удалена одна из вершин первого типа $(i_0`,s_0)$, где $i_0`\in I_0$, или пока не будут удалены все терминальные вершины второго типа. В первом случае алгоритм заканчивается с вердиктом ***fail***, а во втором случае – с вердиктом ***pass***.

Первый алгоритм отличается от второго тем, что при построении двудольного графа мы каждое $H(i)$ делаем равным множеству V_S всех состояний спецификации.

Лемма 21: Алгоритмы верификации, описанные в данном подразделе, при указанных в 5.1 ограничениях заканчивают свою работу за конечное время и выносят вердикт ***fail***, если реализация неконформна, и вердикт ***pass***, если реализация конформна. В последнем случае строится соответствие $R=\{(i,s)|i\in V_{I`} \ \&\ s\in H(i)\}$, которое для первого алгоритма совпадает с R_1, а для первого – с R_2. Объем вычислений имеет оценку $O(mk^2)$.

Доказательство. Поскольку реализация и спецификация конечны, двудольный граф тоже конечен и, следовательно, строится за конечное время. Продолжение верификации связано с удалением некоторых вершин и друг из этого конечного графа, поэтому эта часть каждого из алгоритмов также завершается за конечное время.

По построению этих алгоритмов очевидно, что они выносят правильный вердикт и при вердикте ***pass*** строят требуемое соответствие.

Оценим сложность алгоритмов. Число вершин второго типа, то есть пар $(i—u\to i`,s)$, имеет оценку $O(mk)$. Поскольку в каждую вершину второго типа входит одна дуга первого типа, число таких дуг имеет оценку $O(mk)$. Поскольку из каждой вершины второго типа выходит число дуг, не превышающее число состояний спецификации, число таких дуг имеет оценку

$O(mk^2)$. Поэтому суммарно построение двудольного графа требует $O(mk^2)$ операций.

В наихудшем случае верификация требует однократного просмотра всех дуг двудольного графа, то есть имеет оценку $O(mk^2)$ операций. Эта оценка сложности верификации по двудольному графу совпадает с оценкой сложности его построения. □

Последние две леммы дают следующую теорему.

Теорема 6: Тест, основанный на алгоритме обхода реализации из подраздела 5.3 и любом из алгоритмов верификации из данного подраздела, является полным на классе семантик, спецификаций и реализаций, удовлетворяющих ограничениям из подраздела 5.1. Если тест заканчивает свою работу с вердиктом ***pass***, то строится соответствие $R=\{(i,s)|i\in V_{\mathbf{I}`} \ \&\ s\in H(i)\}$, которое для первого алгоритма совпадает с R_1, а для второго – с R_2. При этом число тестовых воздействий имеет оценку $O(bt^n)$ для $t>1$ и $O(bn^2)$для $t=1$. Объем вычислений имеет оценку $O(bnt^n)+O(bnm)+O(mk^2)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $O(bn^3)$.

Учитывая, что $m=O(bnt)$, оценку объема вычислений можно записать в виде $O(bnt^n)+O(b^2n^2t)+O(bntk^2)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $O(bn^3)$.

5.5. Пример верификации симуляции

На Рис.11 приведен пример верификации симуляции. Семантика содержит как **R**-, так и **Q**-кнопки. Спецификация $\mathbf{S}$ демонстрирует все виды опасности: ненаблюдаемый отказ $\{y\}$ (в состоянии 3), дивергенцию (в состоянии 4) и разрушение (в состоянии 5). Спецификация удовлетворяет собственной H-гипотезе[12] и, по лемме 10, конформна сама себе как реализация. Также приведены примеры еще одной конформной реализации $\mathbf{I_1}$ и одной неконформной реализации $\mathbf{I_2}$. После обхода реализаций будут построены LTS $\mathbf{S`}$, $\mathbf{I_1`}$ и $\mathbf{I_2`}$. Приведены двудольные графы для верификации соответствия по второму алгоритму с указанием соответствия H, в которых серым фоном отмечены терминальные вершины второго типа, а пунктиром – удаляемые дуги.

[12] Единственное проявление недетерминизма в спецификации – это два перехода $0—y→1$ и $0—y→3$. Но в этих состояниях безопасны одни и те же кнопки (кнопка $\{x\}$).

Для конформных реализаций приведены соответствия R_2, а также соответствия R_1 без двудольных графов, которые строятся аналогично.

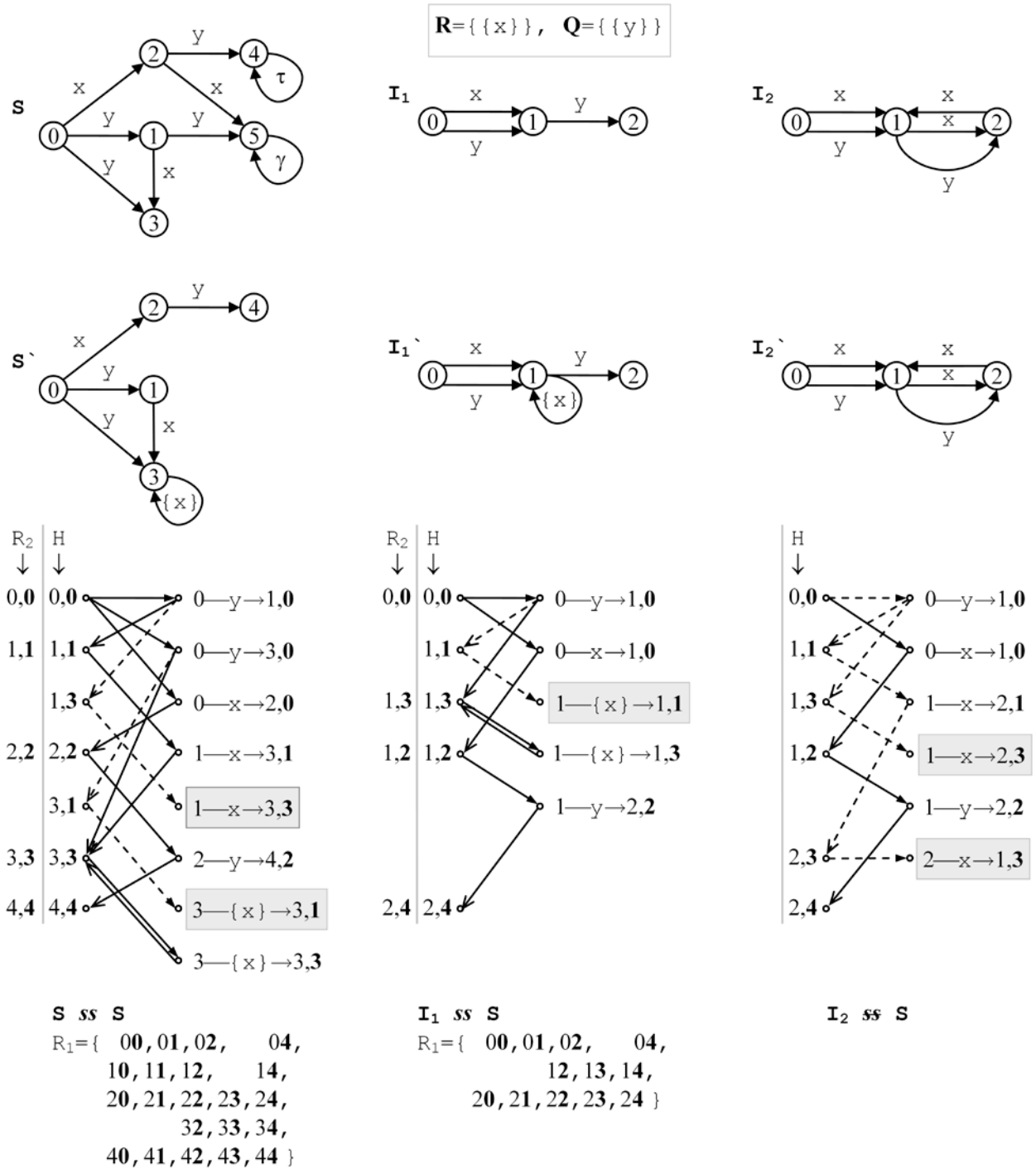


Рис.11. Пример верификации симуляции

Литература

1. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Формализация тестового эксперимента. «Программирование», 2007, No. 5.
2. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. «Наука», 2008.
3. Бурдонов И.Б. Теория конформности для функционального тестирования программных систем на основе формальных моделей. Диссертация на соискание ученой степени д.ф.-м.н., Москва, 2008.
   http://www.ispras.ru/~RedVerst/RedVerst/Publications/TR-01-2007.pdf
4. Бурдонов И.Б., Косачев А.С. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. «Программирование», 2009, No. 6.
5. van Glabbeek R.J. The linear time - branching time spectrum II; the semantics of sequential processes with silent moves. Proceedings CONCUR ’93, Hildesheim, Germany, August 1993 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993, pp. 66-81.
6. König D. Über eine Schlusswaise aus dem Endlichen ins Unendliche. Acta Litt. Ac. Sci. Hung. Fran. Josep. No 3. 1927. pp. 121-130. См. Также Куратовский К., Мостовский А. Теория множеств. М.«Мир», 1970.
7. Lee D., Yannakakis M. Principles and Methods of Testing Finite State Machines – A Survey. Proceedings of the IEEE 84, No. 8, 1090–1123, 1996.
8. Milner R. Lectures on a calculus for communicating systems. Seminar on Concurrency, LNCS 197, Springer-Verlag, pp. 197-220.
9. Milner R. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor: Handbook of Theoretical Computer Science, chapter 19, Elsevier Science Publishers B.V. (North_Holland), pp. 1201-1242. Alternatively see Communication and Concurrency, Prentice-Hall International, Englewood Cliffs, 1989, of which an earlier version appeared as A Calculus of Communicating Systems, LNCS 92, Springer-Verlag, 1980.