И. Бурдонов, А. Косачев
Финальная модель спецификации и удаление неконформных трасс
9-ая российская конференция с международным участием "Новые информационные технологии в исследовании сложных структур".
Алтай .
25 слайдов - предварительная версия

Финальная модель спецификации и удаление неконформных трасс.

Слайд 1. Читаю название

Слайд 2. Формализация тестирования.

Основные понятия

Прежде всего, определим основные понятия, которые мы будем использовать.

Реализация – это модель исследуемой системы.

Спецификация – это модель, описывающая требования к системе.

Отношение «конформности» – это отношение «правильности», которое говорит, что реализация удовлетворяет требованиям, которые описывает спецификация.

Тестирование – это проверка конформности в процессе эксперимента. Тест, подменяя собой окружение реализации, взаимодействует с реализацией и выносит вердикт ***pass*** – «проходит», или вердикт ***fail*** – ошибка.

Реализация проходит тест, если при любом прогоне теста всегда выносится вердикт ***pass***. То есть при любом недетерминированном поведении реализации.

Набор тестов полный для заданной спецификации, если реализация проходит каждый тест из набора тогда и только тогда, когда она конформна спецификации.

Слайд 3. Формализация тестирования.

Семантика взаимодействия

Первое, что мы должны сделать, – это формализовать взаимодействие теста и реализации. Иными словами, мы должны формально задать семантику такого взаимодействия.

Семантика взаимодействия формализует имеющийся набор тестовых возможностей по управлению реализацией и наблюдению за поведением реализации. При тестировании мы можем наблюдать только такое поведение реализации, которое, во-первых, «спровоцировано» тестом (управление) и, во-вторых, наблюдаемо во внешнем взаимодействии. Такое взаимодействие может моделироваться с помощью, так называемой, машины тестирования.

Она представляет собой «чёрный ящик», внутри которого находится реализация.

Управление сводится к тому, что оператор машины, выполняя тест (понимаемый как инструкция оператору), осуществляет тестовое воздействие, нажимая кнопки на клавиатуре машины, тем самым «разрешая» реализации выполнять те или иные действия, которые могут им наблюдаться.

Подчеркнём, что при управлении оператор разрешает реализации выполнять именно множество действий, а не обязательно одно действие. Реализация сама выбирает, какое действие из множества разрешённых действий, она будет выполнять. Этот выбор, вообще говоря, недетерминированный.

Мы предлагаем считать, что оператор может нажимать только одну кнопку, но каждой кнопке соответствует своё множество разрешаемых действий.

Наблюдения (на «дисплее» машины) бывают двух типов. Первый тип – это наблюдение некоторого *внешнего (наблюдаемого) действия*, выполняемого реализацией.

Второй тип наблюдения – это наблюдение *отказа* как отсутствия каких бы то ни было наблюдаемых действий. То есть все действия, разрешенные нажатой кнопкой, в реализации выполняться не могут.

После наблюдения кнопка отжимается, и все внешние действия запрещаются. Далее оператор может нажать другую (или ту же самую) кнопку.

Тестовые возможности определяются тем, какие «кнопочные» множества есть на клавиатуре машины, а также, для каких кнопок возможно наблюдение отказа. Тем самым, семантика взаимодействия определяется алфавитом внешних действий $\boldsymbol{L}$ и двумя наборами кнопок машины тестирования: с наблюдением соответствующих отказов – семейство $\boldsymbol{R}\subseteq 2^{\boldsymbol{L}}$ и без наблюдения отказа – семейство $\boldsymbol{Q}\subseteq 2^{\boldsymbol{L}}$.

Предполагается, что любое действие отлично от любого множества действий, кнопки в совокупности покрывают весь алфавит, и семейства **R** и **Q** не пересекаются.

Такую семантику мы называем ***R*/*Q*-семантикой**.

Вот два примера таких семантик.

Первый пример – это хорошо известная *failure trace semantics*. В соответствующей ей машине тестирования имеются кнопки для всех подмножеств алфавита, и все отказы наблюдаемы.

Второй пример – это семантика популярного отношения ***ioco***. В этой семантике действия разбиваются на стимулы (input) и реакции (output). Есть только одна ***R***-кнопка приема всех реакций, соответствующий отказ обозначается символом δ маленькая и называется по-английски *quiescence*, то есть молчание. Каждый стимул x посылается в реализацию с помощью отдельной ***Q***-кнопки $\{x\}$, $\boldsymbol{Q}=\{\{x\}|x\in\boldsymbol{I}\}$. Соответствующий отказ называется блокировкой стимула, но он ненаблюдаем.

Слайд 4. Формализация тестирования.
Семантика взаимодействия

Кроме внешних действий реализация может совершать внутренние (ненаблюдаемые) действия. Поскольку они ненаблюдаемы, они неразличимы между собой и обозначаются одним символом τ. Эти действия считаются всегда разрешенными (при нажатии любой кнопки или при отсутствии нажатой кнопки).

Для выполнимости любого действия (как внешнего, так и внутреннего) необходимо, чтобы оно было определено в реализации и разрешено оператором. Если этого условия также и достаточно, то есть на выполнение может быть выбрано любое действие, которое определено в реализации и разрешено нажатой кнопкой машины тестирования, то говорят, что в системе нет приоритетов. Сегодня мы ограничимся только системами без приоритетов.

Нам потребуются два «практических предположения»:
1) Любая конечная последовательность любых действий (как внешних, так и внутренних) совершается за конечное время, а бесконечная – за бесконечное время.
2) Передача тестового воздействия (нажатие кнопки) от машины тестирования в реализацию и передача наблюдения обратно от реализации на дисплей машины выполняются за конечное время.

Эти предположения гарантируют возможность наблюдения внешнего действия, выполняемого реализацией, через конечное время после нажатия кнопки, разрешающей это действие. Это нам нужно для того, чтобы конечный тест мог заканчиваться через конечное время.

Бесконечная последовательность τ-действий называется *дивергенцией* и обозначается символом Δ большая. Дивергенция сама по себе не опасна, но при попытке выхода из неё, когда оператор нажимает любую кнопку, он не знает, нужно ли ждать наблюдения или бесконечно долго будут выполняться только внутренние действия. Поэтому оператор не может ни продолжать тестирование, ни закончить его.

При отсутствии дивергенции после нажатия $\boldsymbol{R}$-кнопки через конечное время оператор наблюдает или разрешенное этой кнопкой внешнее действие или соответствующий отказ.

Однако, при нажатии ***Q***-кнопки отказ не наблюдаем, и возможен лишь один вид наблюдения – выполняемое действие. Поэтому такое наблюдение гарантируется только в том случае, когда в реализации не может возникнуть отказ. Если в реализации возможен отказ, то, поскольку этот отказ не наблюдаем, оператор не знает, нужно ли ему ждать наблюдения внешнего действия или такого действия не будет, поскольку возник отказ. Поэтому оператор не может ни продолжать тестирование, ни закончить его.

Кроме этого мы вводим специальное, не регулируемое кнопками действие, которое называем *разрушением* и обозначаем символом γ. Оно моделирует любое нежелательное поведение системы, в том числе и ее реальное разрушение, которого нельзя допускать при взаимодействии.

Разрушение аналогично понятию запрещенного состояния, которое иногда используется в model checking. Но разрушение как запрещенное действие лучше, поскольку мы не опираемся на понятие состояния, а имеем дело только с наблюдениями.

Тестирование, при котором не возникают попытки выхода из дивергенции, ненаблюдаемые отказы и разрушение, называется *безопасным*.

Слайд 5. Модель.
LTS-модель

Наиболее распространенной моделью реализации и спецификации является система помеченных переходов – LTS (Labelled Transition System), которая определяется как ориентированный граф, вершины которого называются состояниями, а дуги помечены внешними действиями или символами τ или γ и называются переходами.

Переход из пресостояния *s* в постсостояние *s`* по символу *z* обозначается стрелкой. Выделяется начальное состояние, с которого реализация начинает работать при каждом рестарте.

После рестарта реализация выполняет последовательность смежных переходов, то есть движение по маршруту, начинающемуся в

начальном состоянии, каждый переход которого помечен некоторым действием. Каждое такое действие разрешается текущей нажатой кнопкой машины тестирования. Разные действия, естественно, могут разрешаться разными кнопками. Действие, разрешаемое кнопкой P, это либо действие, принадлежащее P, либо внутреннее действие τ, либо разрушение γ.

Отказ *P* порождается в *стабильном* состоянии, то есть состоянии, из которого не выходят τ- и γ-переходы, при условии, что из этого состояния не выходят также переходы по действиям из *P*.

1►Добавим переходы-петли по отказам. В примере они помечены пунктиром.

Состояние *дивергентно*, если в нем начинается бесконечный τ-маршрут, то есть маршрут, все переходы которого помечены символом τ.

2►Добавим переходы из дивергентных состояний по символу дельта большая в терминальное состояние. В примере имеется одно дивергентное состояние.

3►Туда же перенаправим все переходы по разрушению.
После этих добавлений и перенаправлений получится LTS, которую мы будем обозначать со знаком «+» в верхнем индексе. Заметим, что её алфавит уже расширен отказами, то есть всеми подмножествами исходного алфавита **L**.

Для взаимодействия, основанного на наблюдениях, единственным результатом тестового эксперимента является чередующаяся последовательность кнопок (тестовых воздействий) и наблюдений, которую будем называть (*тестовой*) *историей*. Поскольку дивергенция и разрушение всегда разрешены, им не предшествует в истории никакая кнопка. Любое другое наблюдение (внешнее действие или ***R***-отказ) разрешается непосредственно предшествующей ему кнопкой. Подпоследовательность истории, состоящая только из наблюдений (включая Δ и γ), называется трассой.

Для систем без приоритетов важны только трассы, поскольку возможность или невозможность появления данного наблюдения после трассы определяется только тем, что нажимаемая кнопка разрешает данное наблюдение, и не зависит от того, какие еще наблюдения она разрешает. В этом случае для данной тестируемой системы множество ее историй однозначно восстанавливается по множеству ее трасс.

Трасса LTS – это последовательность пометок на переходах маршрута, начинающегося в начальном состоянии. Для заданной LTS **S** нас будут интересовать трассы с отказами, то есть трассы LTS $\mathbf{S}^+$. Будем обозначать множество этих трасс через $\boldsymbol{tr}(\mathbf{S}^+)$.

Слайд 6. Гипотеза о безопасности и безопасная конформность.
Отношение безопасности кнопок

Как возможно безопасное тестирование, если реализация неизвестна? Например, если в LTS-реализации переход по разрушению определен в начальном состоянии, то такую реализацию не только нельзя тестировать, но даже запускать на выполнение, поскольку она может разрушиться до первого тестового воздействия, то есть до первого нажатия кнопки.

Выход в том, чтобы ограничиться теми реализациями, которые можно безопасно тестировать для проверки конформности заданной спецификации.

Это ограничение формулируется как гипотеза о безопасности.

Безопасное тестирование, прежде всего, предполагает формальное определение на уровне модели отношения безопасности «кнопка *P* безопасна в модели после трассы σ». При безопасном тестировании будут нажиматься только безопасные кнопки. Это отношение различно для реализационной и спецификационной моделей.

В обоих случаях кнопка, прежде всего, должна быть *неразрушающей* после трассы, то есть ее нажатие не может означать попытку выхода из дивергенции (трасса не продолжается дивергенцией) и не может вызывать разрушение (после действия, разрешаемого кнопкой). Такое

отношение «неразрушаемости кнопки после трассы» называется $\boldsymbol{safe}_{\gamma\Delta}$.

В реализации ***I*** отношение безопасности называется ***safe in***. Кнопка, безопасная по ***safe in***, должна быть, во-первых, неразрушающей и, во-вторых, нажатие кнопки не должно приводить к ненаблюдаемому отказу, если это ***Q***-кнопка.

.

В спецификации ***S*** отношение безопасности называется ***safe by*** и отличается только для ***Q***-кнопок: мы не требуем, чтобы после трассы σ не было ***Q***-отказа *Q*, но требуем, чтобы было хотя бы одно действие из *Q*. Кроме того, если действие разрешается хотя бы одной неразрушающей кнопкой, то оно должно разрешаться какой-нибудь безопасной кнопкой. Если это неразрушающая ***R***-кнопка, то она же и безопасна. Но если все неразрушающие кнопки, разрешающие действие, являются ***Q***-кнопками, то хотя бы одна из них должна быть объявлена безопасной.

Такое отношение безопасности всегда существует: достаточно объявить безопасной каждую неразрушающую кнопку, разрешающую действие, продолжающее трассу. Однако в целом указанные требования неоднозначно определяют отношение ***safe by***, и при задании спецификации дополнительно указывается конкретное отношение ***safe by***.

Слайд 7. Гипотеза о безопасности и безопасная конформность.
Безопасные наблюдения и трассы

Безопасность кнопок определяет безопасность наблюдений. ***R***-отказ *R* безопасен, если после трассы безопасна кнопка *R*. Действие *z* безопасно, если оно разрешается некоторой кнопкой, безопасной после трассы.

Теперь мы можем определить *безопасные трассы*. Трасса σ безопасна, если эта трасса есть в модели и
1) модель не разрушается с самого начала (сразу после включения машины ещё до нажатия первой кнопки), то есть, в ней нет трассы <γ>,

2) каждый символ трассы безопасен после непосредственно предшествующего ему префикса трассы.

Понятно, что безопасные трассы – это трассы, в которых встречаются отказы только из семейства **R**.

Множества неразрушающих трасс, безопасных по ***safe in*** трасс и безопасных по ***safe by*** трасс обозначим $\boldsymbol{Safe}_{\gamma\Delta}$ ***SafeIn*** и ***SafeBy***.

Заметим, что правила для отношения ***safe by*** составлены так, что множество безопасных по ***safe by*** трасс спецификации совпадает с множеством её неразрушающих трасс.

Слайд 8. Гипотеза о безопасности и безопасная конформность.

Требование безопасности тестирования выделяет класс *безопасно-тестируемых* реализаций ***SafeImp***, то есть таких, которые могут быть безопасно протестированы для проверки их конформности заданной спецификации ***S*** с заданным отношением ***safe by*** в заданной ***R***/***Q***-семантике.

Этот класс определяется следующей *гипотезой о безопасности*: реализация ***I*** *безопасно-тестируема* для спецификации ***S***, если 1) в реализации нет разрушения с самого начала, если этого нет в спецификации, 2) после общей безопасной трассы спецификации и реализации любая кнопка, безопасная в спецификации, безопасна после этой трассы в реализации.

Заметим, что для ***ioco***-семантики мы разрешаем в безопасно-тестируемых реализациях «безопасные» блокировки стимулов – блокировки стимулов после трасс, которые в спецификации этими стимулами не продолжаются. Это более либерально, чем требование всюду определенности реализации по стимулам, предлагаемое автором отношения ***ioco*** Яном Тритмансом [16,17].

Таким образом, мы устраняем «несогласованность» отношения ***ioco***, когда всюду определенная по стимулам реализация и реализация, отличающаяся от нее только тем, что в ней есть «безопасные» блокировки, неразличимы при ***ioco***-тестировании, однако первая

может быть конформна, а вторая заведомо неконформна, поскольку не является всюду определенной по стимулам и тем самым не входит в домен отношения ***ioco***.

После этого можно определить отношение (безопасной) *конформности*: реализация ***I*** *безопасно конформна* (или просто *конформна*) спецификации ***S***, если она безопасно-тестируема и выполнено *тестируемое условие*: любое наблюдение, возможное в реализации в ответ на нажатие безопасной (в спецификации) кнопки, разрешается спецификацией.

Это отношение определяет класс конформных реализаций ***ConfImp***.

Следует отметить, что гипотеза о безопасности не проверяема при тестировании и является его предусловием; тестирование проверяет тестируемое условие конформности.

Эта пара классов реализаций как раз и описывает все спецификационные требования: какие реализации тестируются и какие из них считаются конформными. Какой бы способ представления спецификационных требований мы ни выбрали, в конечном счёте они сводятся к определению этой пары классов реализаций. Выбрав представление в виде LTS-модели и отношения ***safe by***, мы, тем самым, определили возможные пары классов реализаций – это уже будут не все пары.

Слайд 9. Генерация тестов.
Определение теста

В терминах машины тестирования тест – это инструкция оператору машины. В каждом пункте инструкции указывается кнопка, которую оператор должен нажимать, и для каждого наблюдения – пункт инструкции, который должен выполняться следующим, или вердикт (***pass*** или ***fail***), если тестирование нужно закончить. В тесте после кнопки допускается только такое наблюдение, которое разрешается этой кнопкой.

Тест можно понимать как префикс-замкнутое множество конечных историй, в котором

1) каждая максимальная история заканчивается наблюдением, и ей приписан вердикт;
2) каждая немаксимальная история, заканчивающаяся кнопкой, может продолжаться во множестве только теми наблюдениями, которые разрешаются этой кнопкой, и обязательно продолжается теми наблюдениями, которые могут встречаться в безопасно-тестируемых реализациях.

Тест безопасен тогда и только тогда, когда в каждой его истории каждая кнопка безопасна в спецификации после подтрассы непосредственно предшествующего этой кнопке префикса истории. Иными словами, тест безопасен тогда и только тогда, когда подтрассы всех его историй являются тестовыми, где *тестовая трасса* – это безопасная трасса или безопасная трасса, продолженная безопасным после неё наблюдением, но не обязательно имеющимся в спецификации после этой трассы.

Слайд 10. Генерация тестов.
Полный набор тестов

Реализация *проходит* тест, если её тестирование с помощью этого теста всегда заканчивается с вердиктом ***pass***. Реализация проходит набор тестов, если она проходит каждый тест из набора. Набор тестов *полный*, если реализация проходит его в том и только том случае, когда она конформна спецификации.

Для проверки конформности любой безопасно-тестируемой реализации ставится задача генерации полного набора тестов по спецификации.

Полный набор тестов всегда существует, в частности, им является набор всех *примитивных* тестов.
Примитивный тест строится по одной выделенной немаксимальной безопасной трассе спецификации. Для этого в трассу вставляются кнопки, которые оператор должен нажимать:

перед каждым отказом R вставляется кнопка R,
перед каждым действием z – какая-нибудь безопасная кнопка P, разрешающая действие z,
а после всей трассы – любая безопасная после нее кнопка P`.

По одной безопасной трассе спецификации можно сгенерировать, вообще говоря, несколько разных примитивных тестов, выбирая разные кнопки. Однако множества тестов, сгенерированных по разным трассам, не пересекаются.

Если наблюдение, полученное после нажатия кнопки, продолжает трассу, тест продолжается (немаксимальная в тесте история). Наблюдение, полученное после нажатия последней кнопки, и любое наблюдение, «ответвляющееся» от трассы, всегда заканчивают тестирование (максимальная в тесте история).

Вердикт ***pass*** выносится, если подтрасса полученной максимальной истории есть в спецификации, а вердикт ***fail*** – если нет.

Такие вердикты соответствуют *строгим* тестам, которые, во-первых, не фиксируют ложных ошибок, то есть для конформных реализаций всегда выносят вердикт ***pass,*** и, во-вторых, не пропускают обнаруженных ошибок, то есть после получения трассы, отсутствующей в спецификации, выносят вердикт ***fail***.

Любой строгий тест (как множество историй) равен объединению некоторого множества примитивных тестов, то есть они обнаруживают те же самые ошибки. Поэтому в теории можно ограничиться рассмотрением только примитивных тестов.

Слайд 11. Финальная модель.
Финальные трассы

Для определения гипотезы о безопасности и безопасной конформности, равно как и для генерации тестов, не нужно знать все трассы спецификации и все кнопки, безопасные после них по ***safe by***.

Достаточно знать только безопасные по ***safe by*** трассы и кнопки, безопасные только после таких трасс по ***safe by***.

Безопасные трассы спецификации – то же самое, что неразрушающие трассы, то есть они не зависят от ***safe by***.
Безопасные по ***safe by*** кнопки – это, прежде всего, неразрушающие кнопки, то есть безопасные по $\boldsymbol{safe}_{\gamma\Delta}$.

Для того, чтобы было видно, какие кнопки разрушающие после безопасных трасс, а какие неразрушающие, нужно, кроме безопасных трасс, взять их продолжения дивергенцией и действиями, за которыми следует разрушение.
Такие продолжения будем называть *финальными суффиксами*.
На слайде это первые 4 пункта в определении финальных трасс.

Отношение ***safe by*** отличается от отношения $\boldsymbol{safe}_{\gamma\Delta}$ только для неразрушающих **Q**-кнопок: такая кнопка может быть как опасна, так и безопасна после безопасной трассы. Это означает, что разные отношения ***safe by*** отличаются друг от друга только тем, какова разность отношений $\boldsymbol{safe}_{\gamma\Delta}$ \ ***safe by***. Когда мы будем говорить, что у нас задано отношение ***safe by***, мы будем иметь в виду, что задана именно эта разность.
Для того, чтобы это отобразить во множестве трасс, достаточно добавить продолжение безопасной трассы σ **Q**-отказом Q тогда и только тогда, когда кнопка Q после трассы σ неразрушающая, но опасна по ***safe by***.
Такое продолжение **Q**-отказом будем называть **Q**-суффиксом.
Добавление **Q**-суффиксов делается независимо от того, продолжается трасса σ в спецификации отказом Q или нет.
На слайде такие добавленные трассы описываются в пункте 5.

Первые 4 пункта в определении финальных трасс задают отношение $\boldsymbol{safe}_{\gamma\Delta}$, а пункт 5 задаёт разность отношений $\boldsymbol{safe}_{\gamma\Delta}$ \ ***safe by***, то есть множество трасс с **Q**-суффиксами, описываемое в пункте 5.

В результате у нас получится множество трасс, которые мы будем называть *финальными трассами*.

Слайд 12. Финальная модель.
Финальная RTS (Refusal Transition System)

Теперь мы хотим изобразить множество финальных трасс конечным образом, если это возможно. Аналогично тому, как конечная LTS задаёт множество своих трасс, которое может быть и бесконечным.

Для этого множество финальных трасс должно быть регулярным. Это значит, что существует конечный граф, порождающий это

множество. Поскольку множество финальных трасс префикс-замкнуто, можно выбрать порождающий граф, в котором одна вершина начальная и все вершины конечные. Такой граф есть, по сути, LTS в расширенном алфавите, состоящем из действий, **R**-отказов и символа дивергенции.

Известно, что порождающий граф всегда можно выбрать детерминированным. Детерминированную LTS в расширенном алфавите $L \cup R \cup \{\Delta\}$ будем называть RTS – Refusal Transition System.

RTS, множество трасс которой – это множество финальных трасс исходной LTS-спецификации, будем называть финальной RTS.

В финальной RTS все переходы по дивергенции, разрушению и **Q**-отказам должны заканчиваться в терминальных состояниях. Без ограничения общности можно считать, что все они заканчиваются в выделенном терминальном состоянии ω, в котором не заканчиваются другие переходы.

Также в финальной RTS из начала перехода по разрушению не выходят другие переходы. Без ограничения общности можно считать, что имеется только один переход по разрушению, ведущий из выделенного состояния γ в состояние ω.

Так у нас получается ещё одна модель спецификации, которая выгодно отличается от исходной LTS-спецификации.

Во-первых, финальная RTS детерминирована: каждая её трасса заканчивается ровно в одном состоянии.

Во-вторых, легко определяются безопасные трассы: это все трассы, заканчивающиеся в состояниях, отличных от ω и γ.

В-третьих, легко определяются безопасные кнопки: кнопка безопасна после трассы, если в состоянии, где заканчивается трасса, нет перехода по дивергенции, нет перехода по действию из этой кнопки, ведущего в состояние γ, и если это **Q**-кнопка Q, то нет перехода по отказу Q.

Слайд 13. Финальная модель.
Построение финальной RTS

Для построения конечной финальной RTS **F** будем предполагать, что нам заданы:
1) конечная **R**/**Q**-семантика, то есть семейства **R** и **Q** конечные,
2) конечная исходная LTS **S**,
3) конечный порождающий граф разности отношений $\boldsymbol{safe}_{\gamma\Delta} \setminus \boldsymbol{safe\ by}$.

Построение финальной RTS состоит из трёх этапов.

На первом этапе мы строим LTS $\mathbf{S}^{+}$, добавляя переходы-петли **R**-отказов, проводя переходы по символу Δ из дивергентных состояний в новое терминальное состояние ω, и перенаправляя в это состояние все переходы по разрушению.

После этого на втором этапе проводится процедура детерминизации. Строится RTS **D**. Её состояниями будут подмножества состояний LTS $\mathbf{S}^{+}$. Переход по символу u ведёт из множества A во множество B, если хотя бы в одном состоянии из A начинается трасса u, а B – множество концов таких трасс. Здесь u – это внешнее действие, **R**-отказ, символ дивергенции или разрушения.

Порождающий граф разности $\boldsymbol{safe}_{\gamma\Delta} \setminus \boldsymbol{safe\ by}$ детерминизируется аналогичным образом. Этот граф задаётся RTS **G** и множеством конечных состояний, в которых заканчиваются переходы по **Q**-отказам и только они. В остальных достижимых состояниях заканчиваются безопасные трассы.

Слайд 14. Финальная модель.
Построение финальной RTS

На третьем этапе финальная RTS **F** строится из RTS **D** и **G**. В финальной RTS, кроме безопасных трасс, должны быть все их продолжения как финальными суффиксами, которые определяются по RTS **D**, так и **Q**-суффиксами, которые определяются по RTS **G**.

RTS **D** содержит все безопасные трассы, однако некоторых безопасных трасс может не быть в RTS **G**. Если после безопасной

трассы и любого её безопасного продолжения нет неразрушающих, но опасных по ***safe by*** **Q**-кнопок, то такая трасса не попадёт в RTS **G**. В этом случае все продолжения этой трассы в финальной RTS приходится строить только по RTS **D**.

Для этого добавим в RTS **G** одно изолированное терминальное состояние "-" – прочерк. После этого состояниями финальной RTS будут пары состояний RTS **D** и **G**, а также два выделенных состояния γ и ω. Начальное состояние – пара начальных состояний.

Переходы определяются правилами вывода, изображенными на слайде.

Первые три правила определяют финальные суффиксы: это переходы по разрушению, по действиям, за которыми непосредственно следует разрушение, и по дивергенции. Эти переходы определяются только по RTS **D**.

Правило 4 определяет **Q**-суффиксы: это переходы по **Q**-отказам. Эти переходы, наоборот, определяются только RTS **G**.

Правила 5 и 6 определяют переходы по безопасным наблюдениям. Здесь такое наблюдение обозначено как u – это действие или **R**-отказ.

Правило 5 применяется, когда переход по u есть в RTS **G**. В этом случае наблюдение u безопасное, поскольку в **G** все трассы без **Q**-отказа в конце безопасные.
Переход из пары (d,g) пре-состояний переходов по u ведет в пару (d`,g`) пост-состояний этих переходов.

Правило 6 применяется, когда перехода по u в **G** нет, но в **D** наблюдение u безопасное.
В этом случае переход ведет в пару (d`,-). Поскольку добавленное состояние "прочерк" изолированное в **G**, дальнейшие переходы будут определяться только по RTS **D**, то есть правилами 1,2,3 и 6.

Слайд 15. Финальная модель.
Ограниченный ***safe by***

Для конечных **R**/**Q**-семантики и исходной LTS **S** частным случаем регулярного отношения ***safe by*** является ограниченное отношение ***safe by***. Это такое отношение, при котором после трасс, заканчивающихся в одном множестве состояний LTS **S**, одинакова безопасность кнопок. Разность $\boldsymbol{safe}_{\gamma\Delta} \setminus \boldsymbol{safe\ by}$ задаётся множеством пар: первый элемент пары – подмножество состояний LTS **S**, а второй элемент – **Q**-кнопка, неразрушающая, но опасная по ***safe by*** в этом подмножестве состояний.

Правила вывода для определения переходов финальной RTS модифицируются так, как показано на слайде.
Отличия в правилах 4 и 5.
Правило 4 определяет **Q**-суффиксы, то есть переходы по **Q**-отказам.
Правило 5 определяет переходы по безопасным наблюдениям: действиям и **R**-отказам.
Здесь состояние RTS **D** – это как раз и есть подмножество состояний исходной LTS **S.**

Слайд 16.Пополнение спецификации.

Неконформные трассы и нерефлексивность конформности

Теперь мы переходим к следующей теме: пополнение спецификации.

Будем говорить, что безопасная трасса спецификации *конформная*, если она встречается хотя бы в одной конформной реализации.

Понятно, что без потери полноты тестирования мы можем удалить из набора тестов все тесты, сгенерированные по неконформным безопасным трассам спецификации.

Поэтому возникает задача удаления таких неконформных трасс из спецификации. Такое преобразование спецификации мы и будем называть пополнением.

Наличие неконформных трасс в спецификации является следствием нерефлексивности конформности ***saco***, в том числе и отношения ***ioco*.**

Если бы спецификация была конформна сама себя, она была бы конформной реализацией и, следовательно, в ней не могло бы быть неконформных трасс.

Нерефлексивность конформности неприятна ещё и тем, что реализация буквально «списанная» со спецификации в общем случае оказывается заведомо ошибочной, поскольку не удовлетворяет гипотезе о безопасности.

Заметим, что если спецификация удовлетворяет собственной гипотезе о безопасности, то она конформна сама себе. Иными словами, если спецификация не конформна сама себе, то она не безопасно-тестируемая.

Также заметим, что нерефлексивность конформности является следствием наличия в семантике **Q**-отказов. Если все отказы наблюдаемы, любая спецификация в такой семантике конформна сама себе.

Слайд 17. Пополнение спецификации.

Неконформные трассы

Для отношения ioco алгоритм пополнения описан в нашей статье, ссылку на которую вы видите на слайде.

Пополнение в общем случае мы определим строго попозже. А сейчас рассмотрим примеры неконформных трасс.

В примере на слайде используется обычная ***ioco***-семантика. Имеется один стимул *x* и две реакции *a* и *b*. Трасса $\langle \delta, x, \delta \rangle$ – конформна в левой спецификации и неконформна в правой спецификации. Почему? Допустим в некоторой конформной реализации есть трасса $\langle \delta, x, \delta \rangle$.

1►Тогда после этой трассы в реализации должно быть наблюдение *x*, поскольку в противном случае после этой трассы будет ненаблюдаемый отказ – блокировка {x}. Но тогда такая блокировка будет и после трассы $\langle x \rangle$, что противоречит гипотезе о безопасности: в спецификации после трассы $\langle x \rangle$ стимул x безопасен – имеется трасса $\langle x,x \rangle$.

2►После трассы ⟨δ,x,δ,x⟩ должно быть сколько угодно стимулов x, поскольку это имеет место в спецификации после трассы ⟨x,x⟩.

3►Теперь посмотрим, какие наблюдения конформны после трассы ⟨δ,x,δ,x⟩ при нажатии кнопки приема реакций.

4►Отказ δ неконформен. Действительно, если в реализации есть трасса ⟨δ,x,δ,x,δ⟩, то есть и трасса ⟨x,x,δ⟩, а это неконформно – средняя и нижняя ветви спецификаций.

5►Реакция *b* тоже неконформна. Действительно, если в реализации есть трасса ⟨δ,x,δ,x,*b*⟩, то есть и трасса ⟨δ,x,x,*b*⟩, а это неконформно – средняя ветвь в спецификациях.

6►Для левой спецификации реакция *a* конформна, поскольку она есть после любой подтрассы трассы ⟨δ,x,δ,x⟩.

7►Однако, согласно правой спецификации, реакция *a* неконформна, поскольку ее нет после подтрассы ⟨x,δ,x⟩ – нижняя ветвь правой спецификации.

8►Получается, что в правой спецификации после трассы ⟨δ,x,δ,x⟩ при нажатии кнопки приема реакций все возможные наблюдения неконформны. Поэтому нам нужно удалить все добавленные трассы, выделенные синим цветом.

9►Из-за этого нам придётся удалить трассу ⟨δ,x,δ⟩.

10►А тогда остается трасса ⟨δ,x⟩, которая и так есть в средней ветви спецификации. Поэтому всю верхнюю ветвь можно удалить.

Слайд 18.Пополнение спецификации.

Неконформные трассы

Второй пример отличается тем, что вместо перехода-петли по реакции *a* в 5-ом состоянии имеется τ-переход.

1▶Из-за этого в правой спецификации появляется висящий тау-переход из состояния 5.

2▶Удаляя этот переход, мы получаем в состоянии 5 отсутствие конформных реакций и отказа δ.

3▶Поэтому придется удалить состояние 5. Из-за этого становится недостижимой часть спецификации.

4▶Поэтому удаляем эту часть. Также образуется висящий переход по стимулу *x* из состояния 2. Придется его удалить.

5▶А тогда нужно удалить и состояние 2, поскольку из него нет перехода по стимулу *x*, хотя этот стимул безопасен.

6▶Из-за этого появляется висящий тау-переход из состояния 0. Удаляем его тоже.

Итак, мы удалили верхнюю и среднюю ветви правой спецификации.

Слайд 19.Пополнение спецификации.

Неконформные трассы

В третьем примере переход-петля по реакции *b* в 3-ем состоянии тоже заменяется на τ-переход.

1▶Теперь, повторяя все предыдущие рассуждения и действия, мы получаем висящий тау-переход из состояния 3.

2▶Удаляя этот переход, мы получаем в состоянии 3 отсутствие конформных реакций и отказа δ.

3▶Поэтому придется удалить состояние 3. Из-за этого становится недостижимой часть спецификации.

4▶Поэтому удаляем эту часть. Также образуется висящий тау-переход из состояния 0.

5▶Придется его удалить.

6▶А тогда нужно удалить и состояние 0.

Итак, мы удалили все ветви правой спецификации.

Это значит, что даже пустая трасса, а тем самым и все имеющиеся трассы неконформны в правой спецификации. Но пустая

трасса есть в любой реализации, следовательно, все реализации неконформны.

Эти примеры показывают, насколько полезным может оказаться анализ неконформных трасс. Полный набор примитивных тестов для любой из спецификаций в этих примерах бесконечен, поскольку бесконечно число безопасных трасс (за счет цикла в состоянии 7). Тем самым полное тестирование бесконечно.

Для правой спецификации $\mathbf{S_2}$ из первого примера после получения трассы $\langle \delta, x, \delta \rangle$ можно сразу закончить тестирование с вердиктом ***fail***.

Для правой спецификации $\mathbf{S_4}$ из второго примера то же самое относится уже к более короткой трассе $\langle \delta \rangle$.

Для правой спецификации $\mathbf{S_6}$ из третьего примера тестирование вообще излишне. У этой спецификации нет конформных реализаций, что определяется без всякого тестирования простым анализом спецификации.

Слайд 20.Пополнение спецификации.

Неконформные трассы

Итак, мы видели на этих примерах, что при пополнении сначала добавляются некоторые трассы, помеченные синим цветом, а потом удаляются как некоторые из добавленных трасс, так и некоторые старые трассы, помеченные бледным цветом.

В общем случае после пополнения могут остаться как новые – синие – трассы, так и удалиться старые – бледные – трассы.

1►Рассмотрим спецификации $\mathbf{S_5}$ и $\mathbf{S_2}$. Мы сделаем из них новую специкацию $\mathbf{S_{52}}$.

2►Для этого в семантику добавим ещё один стимул *x`*.

3►В правой спецификации $\mathbf{S_2}$ заменим стимул *x* на *x`*. Также пометим штрихами состояния.

4►Новым начальным состоянием будет состояние 0 левой спецификации, из которого проведем тау-переход в состояние 0` правой спецификации.

5►Теперь после добавления и удаления трасс мы получаем вот такую картинку. Мы видим, что добавились и потом не удалились синие трассы в левой части, и удалились старые трассы из верхней ветви в правой части.

Слайд 21.Пополнение спецификации.

Семантика пополнения

Определение пополнения в общем случае и алгоритм пополнения можно прочитать в нашей работе, ссылка на которую находится вверху слайда.

Для отношениия ***ioco*** удается построить пополнение в исходной семантике. Однако, в общем случае это не так: пополнение не существует в той же самой семантике, в которой задана исходная спецификация.

Сначала рассмотрим пример.

Спецификация $\mathbf{S}_7$ как свою часть использует спецификацию $\mathbf{S_6}$ из примера 3, в которой все трассы неконформны.

Семантика спецификации $\mathbf{S}_7$ получается из ***ioco***-семантики добавлением пустой **R**-кнопки, которая позволяет наблюдать переход реализации в стабильное состояние.

1►В спецификации $\mathbf{S}_7$ трасса <*y*> неконформна.

2►Но если мы удалим переход по *y*, ведущий в начальное состояние LTS $\mathbf{S}_6$, то у нас возникнет отказ δ, после которого стимул *x* станет опасным, что приведет к расширению класса конформных реализаций.

Например, рассмотрим безопасно-тестируемую реализацию справа на слайде. В ней есть трасса ⟨δ,x,δ⟩. Она неконформна для $\mathbf{S}_7$, поскольку после ⟨δ,x⟩ допускается только *y* – нижняя ветвь спецификации.

3►А после преобразования эта реализация станет конформной, поскольку *x* станет опасным после δ. Поэтому такое преобразование не является пополнением.

Более строгое доказательство этого утверждения я опускаю для экономии времени.

На самом деле пополнения нет в любой семантике, где есть хотя бы две разные **R**-кнопки и какое-то действие, этим кнопкам не принадлежащее.

Более точно: в такой семантике существует такая спецификационная модель и такое отношение ***safe by***, что конформные реализации существуют, но не существует пополнения в той же семантике.

Заметим, что ***ioco***-семантика не относится к такому классу семантик: в ней есть только одна **R**-кнопка приёма всех реакций.

А семантика спецификации $\mathbf{S}_7$ относится к такому классу семантик: в ней есть ещё одна – пустая **R**-кнопка и стимул x, не являющийся реакцией и не принадлежащий пустому множеству.

4►Однако можно построить пополнение в другой, расширенной семантике. Для этого в каждую кнопку P добавим новое фиктивное действие не из алфавита **L**, которое назовем не-отказом – не-P и обозначим двойным зачёркиванием. Для исходной **R**/**Q**-семантики такую расширенную семантику будем обозначать $\mathbf{R}^{\#}/\mathbf{Q}^{\#}$ и называть решётчатой семантикой.

Для спецификации $\mathbf{S}_7$ на слайде справа изображено её пополнение в решётчатой семантике. Мы удаляем неконформный переход по реакции *y*, но при этом не возникает лишнего отказа δ, поскольку мы проводим переход по фиктивной реакции не-дельта.

Слайд 22.Пополнение спецификации.

Определение в общем случае

Теперь мы определим строго, что такое пополнение в общем случае.

Будем считать, что спецификация задаётся *спецификационной парой* : **R/Q**-семантика и финальная RTS **S**.

Определим отношение **L**-вложенности спецификационных пар.

Будем говорить, что пара (**R/Q**, **S**) **L**-вложена в пару (**R`/Q`**, **S`**), если последняя определяет не меньший класс безопасно-тестируемых и тот же самый класс конформных реализаций в алфавите **L**.

То есть она определяет не меньшее пересечение класса безопасно-тестируемых реализаций и, соответственно, то же самое пересечение класса конформных реализаций с классом реализаций в заданном алфавите **L**.

Заметим, что это определение годится не только в том случае, когда две спецификации определены для одной и той же семантики взаимодействия.

Для того, чтобы сохранить возможности тестирования реализаций в алфавите **L**, **R`/Q`**-семантика должна быть **L**-эквивалентна исходной **R/Q**-семантике.

А именно: если взять пересечения всех её **R`**-кнопок с алфавитом **L** должно получиться семейство **R**, а если взять пересечения всех её **Q`**-кнопок с алфавитом **L** должно получиться семейство **Q**. Иными словами, её кнопки отличаются от кнопок исходной семантики только наличием в них каких-то дополнительных действий не из алфавита **L**.

Понятно, что реализация в алфавите **L** не содержит этих дополнительных действий. Поэтому нажатие кнопки на **R`/Q`**-машине тестирования, внутри которой находится реализация в алфавите **L**, вызывает тот же эффект, что нажатие соответствующей кнопки на **R/Q**-машине, внутри которой находится та же реализация.

Решётчатая семантика является частным случаем **L**-эквивалентной семантики: здесь в каждую кнопку добавляется ровно одно действие – не-отказ.

L-конусом назовём множество спецификационных пар, в которые **L**-вложена исходная пара (**R/Q**, **S**) и которые имеют семантики, **L**-эквивалентные исходной **R/Q**-семантике.

Смысл этого определения в том, что любая пара из **L**-конуса может заменять собой исходную пару (**R/Q**, **S**), то есть использоваться вместо неё.

Действительно, тестовые возможности для тестирования реализаций в алфавите **L** те же самые.

Но нас интересует не просто тестирование, а безопасное тестирование. Поскольку класс безопасно-тестируемых реализаций в алфавите **L** не сужается, мы по-прежнему, можем безопасно-тестировать те же самые реализации, что для исходной спецификационной пары.

Наконец, класс конформных реализаций в алфавите **L** не меняется. А это означает, что мы предъявляем к реализациям те же самые спецификационные требования, но только сформулированные несколько иным способом.

Пополнением называется спецификационная пара из **L**-конуса, в которой RTS не содержит неконформных трасс.

Слайд 23.Пополнение спецификации.

Условия конечности

С помощью не-отказов удаётся построить пополнение для любой исходной спецификации в любой исходной семантике с любым отношением ***safe by***.

Когда такое пополнение можно быть конечным, например, заданным в виде конечной финальной RTS? И когда его можно построить алгоритмически?

Достаточные условия следующие:

1. Конечность алфавита внешних действий.
2. Конечность финальной RTS-модели исходной спецификации.

Мы уже знаем, что конечную финальную RTS-модель исходной спецификации можно построить из конечной исходной LTS-модели спецификации, если конечна семантика и регулярно отношение ***safe by***.

Слайд 24.Пополнение спецификации.

∇-финальная RTS

Вообще говоря, для данной спецификации может быть много разных пополнений, в том числе в виде финальных RTS.

В нашей работе, ссылка на которую находится на слайде, предложен алгоритм построения пополнения в виде финальной RTS, которую мы назвали ∇-финальной RTS.

Она обладает целым рядом полезных свойств.

В RTS выделены четыре состояния: γ, Δ, Δ`, ϖ.

Состояние ϖ терминально. В нём заканчиваются только переходы по дивергенции и разрушению.

Имеется единственный переход по разрушению, ведущий из состояния γ в состояние ϖ, и два перехода по дивергенции, ведущие из состояний Δ и Δ` в состояние ϖ.

Переходы по не-отказам заканчиваются только в состояниях γ, Δ и Δ`, а другие переходы здесь не заканчиваются.

Детерминированность RTS гарантирует, что каждая её трасса заканчивается только в одном состоянии.

Безопасные трассы.

Все трассы, заканчивающиеся в выделенных состояниях содержат не-отказы. Все остальные трассы, то есть трассы, не заканчивающиеся в выделенных состояниях, являются трассами без не-отказов и все эти трассы безопасны и конформны. Именно по ним должны генерироваться тесты.

Безопасные кнопки.

Безопасность кнопок после трасс, заканчивающихся в одном состоянии, одинаковая. Она определяется очень просто. Если из состояния переход по не-отказу ведёт в состояние γ, то соответствующая кнопка опасна. В противном случае она безопасна.

Актуальные наблюдения.

Будем говорить, что наблюдение, безопасное после трассы, актуально, если такое наблюдение встречается после этой трассы в безопасно-тестируемых реализациях. Понятно, что в тесте после нажатия кнопки нужно уметь обрабатывать только актуальные

наблюдения. Обработку неактуальных наблюдений можно из теста удалить.

Два состояния Δ и Δ` введены для того, чтобы различать актуальные и неактуальные наблюдения, которые безопасны, но отсутствуют в спецификации. Это позволяет легко оптимизировать каждый тест, оставляя в нём только актуальные наблюдения.

Безопасное действие актуально в состоянии, если оно не принадлежит отказам, которыми помечены переходы-петли в этом состоянии.

Безопасный **R**-отказ актуален в состоянии, если в этом состоянии нет перехода по соответствующему не-отказу в состояние Δ`. Это эквивалентно тому, что в этом состоянии либо есть переход по отказу, либо есть переход по не-отказу, ведущий в состояние Δ.

Рефлексивность. ∇-финальная RTS конформна сама себе, тем самым решается проблема нерефлексивности конформности. Более строго: по ∇-финальной RTS можно построить LTS-модель спецификации, в которой не будет **Q**-отказов, поэтому она удовлетворяет собственной гипотезе о безопасности и, следовательно, конформна сама себе.