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 .
22+4 - окончательная 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.

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

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, 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 её поведением. Такая семантика 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 ту или иную кнопку на клавиатуре машины. На каждой кнопке написано множество действий, которые разрешается выполнять реализации. Реализация сама выбирает, какое действие из этого множества она будет выполнять. Этот выбор, вообще говоря, недетерминированный.

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.

Тем самым, семантика взаимодействия определяется алфавитом внешних действий ***L*** и двумя наборами кнопок машины тестирования: семейство ***R*** кнопок, для которых возможно наблюдение отказа, и семейство ***Q*** кнопок, для которых отказ ненаблюдаем.

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

В качестве примера я приведу только одну семантику, которая нам понадобится дальше, тоже в примерах. 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. Каждый стимул *x* посылается в реализацию с помощью отдельной ***Q***-кнопки. Соответствующий отказ называется блокировкой стимула, но он ненаблюдаем.

Слайд 4. Формализация тестирования.*Семантика взаимодействия*

Кроме внешних действий реализация может совершать внутренние (ненаблюдаемые) действия, которые обозначаются символом τ. Эти действия всегда разрешены независимо от кнопок.

Бесконечная последовательность τ-действий называется *дивергенцией* и обозначается символом Δ большая.

Рассмотрим три опасности, которые могут возникать при тестировании.

Первые две описывают ситуацию, когда оператор не может ни продолжать тестирование, ни закончить его, поскольку не знает, нужно ли ждать наблюдения или его не будет. Это бывает при попытке выхода из дивергенции, то есть при нажатии любой кнопки во время дивергенции, поскольку бесконечно долго могут выполняться только внутренние действия. Также это бывает, когда нажимается **Q**-кнопка, а в реализации возможен ненаблюдаемый отказ.

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

Тестирование, избегающее этих трёх опасностей, называется *безопасным*.

Слайд 5. Модель. *LTS-модель*

Наиболее распространенной моделью реализации и спецификации является система помеченных переходов – LTS.

Я думаю, все знают, что такое LTS и их трассы. Поэтому обращу внимание только на важные вещи.

Нас будут интересовать трассы с отказами, дивергенцией и разрушением.

1►Чтобы их получить, мы преобразуем исходную LTS S в LTS S^+, со знаком «+» в верхнем индексе.

Отказы порождаются в стабильных состояниях. Они дорисованы в виде синих пунктирных петель. Поскольку мы ввели разрушение, которое всегда разрешено, стабильное состояние – это то, где нет не только тау-, но и гамма-переходов.

Дивергенция и разрушение в трассах нам нужны только для того, чтобы показать возможные опасности тестирования. Поэтому они могут быть только в конце трассы. Для этого мы добавим в

дивергентных состояниях переходы по символу дельта большая в терминальное состояние, они нарисованы зелёным штрихом. И туда же перенаправим все переходы по разрушению, они нарисованы красным штрихом.

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

Понятно, что не все реализации могут безопасно тестироваться для проверки их конформности заданной спецификации. Поэтому мы ограничиваемся теми реализациями, для которых это возможно.

Это ограничение формулируется как гипотеза о безопасности.

Прежде всего, формально определим отношение безопасности «кнопка *P* безопасна в модели после трассы σ». При безопасном тестировании будут нажиматься только такие кнопки. Это отношение различно для реализационной и спецификационной моделей.

В обоих случаях кнопка должна быть *неразрушающей* после трассы, а именно: её 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» обозначим $\boldsymbol{safe}_{\gamma\Delta}$.

В реализации отношение безопасности называется ***safe in***. Кнопка безопасна по ***safe in***, если она неразрушающая и ее нажатие не может вызвать ненаблюдаемый отказ.

В спецификации отношение безопасности называется ***safe by*** и отличается только для ***Q***-кнопок: пункт 2 – мы не требуем, чтобы после трассы не было ***Q***-отказа, но требуем, чтобы было хотя бы одно действие из кнопки. Кроме того, пункт 3 – если действие разрешается хотя бы одной неразрушающей кнопкой, то оно должно разрешаться какой-нибудь безопасной кнопкой.

Указанные требования неоднозначно определяют отношение ***safe by***, и при задании спецификации дополнительно указывается это отношение.

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

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

Теперь определим *безопасные трассы*. Трасса безопасна, если эта трасса есть в модели и

1) модель не разрушается с самого начала, то есть, в ней нет трассы $<\gamma>$,

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**.

Введем обозначения для множеств трасс, безопасных по отношениям $\boldsymbol{safe}_{\gamma\Delta}$, ***safe in*** и ***safe by***.

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

Слайд 8. Гипотеза о безопасности и безопасная конформность.

Требование безопасности тестирования выделяет класс *безопасных* реализаций ***SafeImp***, который определяется следующей *гипотезой о безопасности*: реализация *безопасна* для спецификации, 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.

Теперь определим отношение *конформности*: реализация *конформна* спецификации, если она безопасна – это предусловие тестирования, и выполнено *тестируемое условие*: любое наблюдение, возможное в реализации в ответ на нажатие безопасной в спецификации кнопки, имеется в спецификации.

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

Слайд 9. Генерация тестов. *Определение теста*

Безопасные тесты генерируются по безопасным трассам спецификации. Для этого в трассы 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.

Пройдя некоторую трассу и нажав кнопку, мы получаем наблюдение. Если трасса, продолженная этим наблюдением, не максимальная в тесте, тест продолжается. В противном случае выносится вердикт. Если полученное наблюдение есть в спецификации после такой трассы, то выносится вердикт ***pass***, а если нет, то вердикт ***fail***.

Понятно, что в тесте достаточно обрабатывать только такие наблюдения, которые разрешаются нажимаемой кнопкой и встречаются в безопасных реализациях. Такие наблюдения будем называть ***актуальными***.

На практике используются только конечные тесты, то есть содержащие конечное число конечных трасс.

Слайд 10. Финальная модель. *Финальные трассы*

Для определения гипотезы о безопасности и безопасной конформности, равно как и для генерации тестов, очевидно, достаточно знать только безопасные трассы и кнопки, безопасные после них. Это задается с помощью трасс, которые мы назвали *финальными*. Подробнее об этом можно прочитать в нашей работе, ссылка на которую находится вверху слайда.

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.

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 $\boldsymbol{safe}_{\gamma\Delta} \setminus$ ***safe by***. Будем считать, что отношение ***safe by*** задается этой разностью.

Для того, чтобы это отобразить во множестве трасс, достаточно добавить продолжение безопасной трассы **Q**-отказом Q, если кнопка Q после трассы неразрушающая, но опасна по ***safe by***. Такое добавление делается независимо от того, продолжается трасса в спецификации **Q**-отказом или нет.

Слайд 11. Финальная модель.
Финальная RTS (Refusal Transition System)

Для того, чтобы изобразить множество финальных трасс конечным образом, оно должно быть регулярным, то есть должен существовать конечный граф, порождающий это множество. Известно, что такой граф всегда можно сделать детерминированным. Поскольку множество финальных трасс префикс-замкнуто, в порождающем графе может быть одна вершина начальная и все вершины конечные. Поэтому такой граф есть, по сути, детерминированная LTS в расширенном алфавите, состоящем из действий, **R**-отказов и символа дивергенции. Будем называть её финальной RTS – Refusal Transition System.

Без ограничения общности можно считать, что все переходы по дивергенции, разрушению и **Q**-отказам заканчиваются в одном терминальном состоянии ω, в котором другие переходы не заканчиваются. Кроме того, переход по разрушению только один, и он ведёт 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 12. Финальная модель. *Построение финальной RTS*

Для построения конечной финальной RTS 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 регулярной разности отношений $\boldsymbol{safe}_{\gamma\Delta} \setminus \boldsymbol{safe\ by}$.

В нашей работе о финальных моделях предложен алгоритм такого построения.

Частным случаем регулярного отношения ***safe by*** является ограниченное отношение ***safe by***, заданное на конечной LTS-спецификации. Это когда после трасс, заканчивающихся в одном множестве состояний LTS, определяется одинаковая безопасность кнопок по ***safe by***.

Слайд 13. Удаление неконформных трасс.
Неконформные трассы и нерефлексивность конформности

Теперь мы переходим к следующей теме: удаление неконформных трасс.

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

Понятно, что для генерации тестов не нужны неконформные трассы. Поэтому возникает задача удаления неконформных трасс из спецификации. Преобразование спецификации, которое это делает, 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.

Пополнение решает заодно и проблему нерефлексивности.

Слайд 14. Удаление неконформных трасс. *Неконформные трассы*

Сейчас я покажу на примерах, что неконформные трассы могут быть в спецификациях и как строится пополнение. Будет использована семантика отношения ***ioco***.

Вообще для ***ioco*** алгоритм пополнения описан в нашей статье, ссылку на которую вы видите на слайде.

В наших рассуждениях мы будем использовать тот простой факт, что LTS вместе с каждой своей трассой содержит и все её подтрассы, которые получаются удалением каких-то отказов. Это следствие того, что отказы изображаются переходами-петлями.

В примерах на слайде трасса $\langle \delta, x, \delta \rangle$ – конформна в левой спецификации и неконформна в правой спецификации. Почему? Допустим в некоторой конформной реализации есть такая трасса.

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

2►После трассы $\langle \delta,x,\delta,x \rangle$ должно быть сколько угодно стимулов x, поскольку это имеет место в спецификации после подтрассы $\langle x,x \rangle$.

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 подтрасса ⟨x,x,δ⟩, а это неконформно – средняя и нижняя ветви спецификаций.

5►Аналогично реакция *b* неконформна из-за средней ветви в спецификациях.

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

7►Однако, для правой спецификации реакция *a* неконформна из-за нижней ветви спецификации.

Получается, что в правой спецификации после трассы ⟨δ,x,δ,x⟩ при нажатии кнопки приема реакций все возможные наблюдения неконформны.

8►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 вся верхняя ветвь при пополнении удаляется.

Слайд 15.Удаление неконформных трасс. *Неконформные трассы*

Второй пример отличается тем, что переходы-петли по реакциям в состояниях 3 и 5 заменяются τ-переходами.

1►После удаления верхней ветви правой спецификации, тау-переход из состояния 5 становится висящим.

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.

5▶Придется его удалить.

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

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

8▶У нас опять получается висящий тау-переход из состояния 3. Поэтому аналогично придется удалить нижнюю ветвь правой спецификации.

9▶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 трассы правой спецификации неконформны. Но пустая трасса есть в любой реализации, следовательно, все реализации неконформны.

Эти примеры показывают, насколько полезным может оказаться анализ неконформных трасс. Полный набор 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 спецификации $\mathbf{S_2}$ из первого примера после получения трассы $\langle \delta, x, \delta \rangle$ можно сразу закончить тестирование с вердиктом ***fail***.

Для спецификации $\mathbf{S_4}$ тестирование вообще излишне, поскольку у неё нет конформных реализаций. А это определяется без всякого тестирования простым анализом спецификации.

Слайд 16. Удаление неконформных трасс. *Неконформные трассы*

В этих примерах в результате пополнения либо какие-то трассы добавляются – в левых спецификациях, либо удаляются –в правых спецификациях. В общем случае может быть и то и другое.

1▶Для примера достаточно объединить левую и правую спецификации. Вот как на картинке.

2►Мы видим, что после пополнения добавились синие трассы в левой части, и удалились старые трассы из верхней ветви в правой части.

Слайд 17. Удаление неконформных трасс. *Семантика пополнения*

Для отношения ***ioco*** удается построить пополнение не только для этих примеров, но и для любой спецификации. В общем случае пополнение может не существовать. Более точно: оно не существует в той же самой семантике, в которой задана исходная спецификация.

Подробнее о пополнении в общем случае можно прочитать в нашей работе, ссылка на которую находится вверху слайда.

Рассмотрим пример.

Семантика спецификации $\mathbf{S}_5$ получается из ***ioco***-семантики добавлением пустой **R**-кнопки, которая позволяет наблюдать переход реализации в стабильное состояние.

Эта спецификация как свою часть использует спецификацию $\mathbf{S_4}$, в которой все трассы неконформны.

Поэтому в спецификации $\mathbf{S}_5$ трасса $<a>$ неконформна.

1►Но если мы удалим переход по a, ведущий в начальное состояние LTS $\mathbf{S}_4$, то у нас возникнет отказ δ, после которого стимул x станет опасным, что приведет к расширению класса конформных реализаций.

Вот реализация справа на слайде. Она неконформна, поскольку в ней после трассы $\langle \delta, x \rangle$ есть δ, а в спецификации допускается только реакция b – нижняя ветвь спецификации.

2►Однако, после преобразования эта реализация станет конформной, поскольку x станет опасным после δ. Поэтому такое преобразование не является пополнением.

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

ioco-семантика не такая: в ней только одна **R**-кнопка приёма всех реакций.

З►В то же время пополнение можно построить 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 $\mathbf{S}_5$ на слайде справа изображено её пополнение в решётчатой семантике. Вместо неконформного перехода по реакции *a* мы проводим переход по фиктивной реакции не-дельта, поэтому не возникает лишнего отказа δ.

Слайд 18. Удаление неконформных трасс. *Определение в общем случае*

Теперь определим строго, что такое пополнение в решётчатой семантике.

Будем считать, что исходная спецификация задаётся *парой*: **R/Q**-семантика и финальная RTS **S**. Пополнение задаётся решётчатой парой $(\mathbf{R}^{\#}/\mathbf{Q}^{\#}, \mathbf{S}^{\#})$.

«Решётчатая» семантика эквивалентна исходной для реализаций в алфавите **L**. Если в машину тестирования поместить такую реализацию, то поскольку она не содержит не-отказов, нажатие кнопки на $\mathbf{R}^{\#}/\mathbf{Q}^{\#}$-машине тестирования вызывает тот же эффект, что нажатие соответствующей кнопки на **R**/**Q**-машине.

Будем говорить, что исходная пара **L**-вложена в решётчатую пару, если при переходе от исходной пары к решётчатой паре все старые безопасные реализации остаются безопасными, а конформными будут те и только те из них, которые были конформны раньше.

Это значит, что решётчатую пару можно использовать вместо исходной пары.

Пополнение в решётчатой семантике – это такая решётчатая пара, в которую **L**-вложена исходная пара и которая не содержит неконформных трасс.

Слайд 19. Удаление неконформных трасс. *Условия конечности*

С помощью не-отказов удаётся построить пополнение для любой исходной спецификации в любой 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 можно построить из конечной LTS, если конечна семантика и регулярно отношение ***safe by***.

Слайд 20. Удаление неконформных трасс. *∇-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.

В этой RTS, кроме состояний γ и ϖ, выделены ещё два состояния: Δ и $\Delta`$.

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

∇-финальная RTS обладает целым рядом полезных свойств.

1. Рефлексивность. Она конформна сама себе, тем самым решается проблема нерефлексивности конформности. Более строго: по ∇-финальной RTS можно построить LTS с тем же множеством финальных трасс, которую можно рассматривать как конформную реализацию.

2. Детерминированность как в любой финальной RTS.

3. Безопасные трассы.

Все безопасные трассы без не-отказов конформны. Это те и только те трассы, которые не заканчиваются в четырёх выделенных состояниях.

Слайд 21. Удаление неконформных трасс. *∇-финальная RTS*

4. Безопасные кнопки.

Как в любой финальной RTS безопасность кнопок после трассы определяется конечным состоянием трассы. Если из состояния переход по не-отказу ведёт в состояние γ, то соответствующая кнопка опасна. В противном случае она безопасна.

5. Актуальные наблюдения.

В тесте после нажатия кнопки нужно обрабатывать только актуальные наблюдения, то есть те, которые встречаются в безопасных реализациях. Если наблюдение есть в спецификации, то оно не только актуально, но и конформно, поскольку все трассы конформны. Поэтому речь идёт только об отсутствующих в спецификации безопасных наблюдениях.

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

А для определения актуальности безопасного **R**-отказа используются состояния Δ и Δ`. Такой отказ актуален в состоянии, если в этом состоянии нет перехода по соответствующему не-отказу в состояние Δ`.