И. Бурдонов, А. Косачев
Исследование графа взаимодействующими автоматами
10-ая российская конференция с международным участием "Новые информационные технологии в исследовании сложных структур".
9-13 июня 2014. Алтай.
42+1 слайд

Список слайдов:

1. Титул
2. 1994-2014
3. Редукция
4. Конечные автоматы и контрактные спецификации
5. Тестирование как обход графа. Тестирование с закрытым и открытым состоянием.
6. Частично определённые автоматы
7. Концепция безопасного тестирования
8. Недетерминизм
9. Факторизация
10. IOLTS: Input/Output Labelled Transition System
11. ***ioco*** : проблемы
12. ***ioco*** : Безопасное тестирование
13. $\boldsymbol{ioco}_{\beta\gamma\delta}$: Наблюдаемые блокировки стимулов
14. LTS общего вида и R\Q-семантика
15. Безопасность в реализации и спецификации
16. Гипотеза о безопасности и конформность saco
17. Полное тестирование с открытым состоянием
18. Полное тестирование с открытым состоянием
19. Пополнение: удаление ненаблюдаемых отказов
20. Проблема композиции: монотонное преобразование
21. Удаление из спецификации неконформных трасс
22. Финальная модель спецификации
23. Расширение R\Q-модели: медиаторы
24. Расширение R\Q-модели: приоритеты (1)

25. Расширение R\Q-модели: приоритеты (2)
26. Расширение R\Q-модели: слабая симуляция (1)
27. Расширение R\Q-модели: слабая симуляция (2)
28. Расширение R\Q-модели: слабая симуляция (3)
29. Расширение R\Q-модели: слабая симуляция (4)
30. Критика R\Q-модели (1)
31. Критика R\Q-модели (2)
32. Критика R\Q-модели (3)
33. Модель наблюдений: T/O-семантика (1)
34. Модель наблюдений: T/O-семантика (2)
35. Модель наблюдений: T/O-семантика (3)
36. Модель наблюдений: T/O-семантика (4)
37. Модель наблюдений: T/O-семантика (5)
38. Модель событий: T/E-семантика (1)
39. Модель событий: T/E-семантика (2)
40. Модель событий: T/E-семантика (3)
41. Параллельное тестирование
42. Спасибо

Развитие теории конформности: семантики, формальные модели, алгоритмы

Слайд 1. Титул

Развитие теории конформности: семантики, формальные модели, алгоритмы.

Слайд 2. 1994-2014

В Институте Системного Программирования мы начали заниматься тестированием конформности в 1994-ом году. Это был проект по верификации ядра операционной системы реального времени для канадской телекоммуникационной компании Nortel.

Хотя цели этой работы были чисто практическими, ставилась задача разработки общей методологии и технологии тестирования конформности, пригодной для широкого класса приложений.

Она получила название KVEST и впоследствии легла в основу семейства методологий и технологий под общим названием UniTESK, которое развивается и широко используется по настоящее время.

Слайд 3. Редукция

Мы рассматриваем трассовые конформности, которые основаны на тестовых воздействиях и наблюдениях.

В общем случае трасса – последовательность всего того, что происходит: тестовых воздействий и наблюдений.

Но пока нам достаточно считать, что трасса – это последовательность наблюдений, т.е. не важно, каким тестовым воздействием мы вызвали то или иное наблюдение.

В основном мы будем рассматривать конформности, которые основаны на разделении всех трасс на конформные и неконформные – ошибки.

Реализация конформна, если в ней нет ошибок.

Такие конформности мы называем редукциями.

Слайд 4. Конечные автоматы и контрактные спецификации

Мы начинали с модели конечного автомата, в котором каждый переход помечен двумя символами: стимулом и реакцией.

К тому времени уже была создана хорошо разработанная теория тестирования конечных автоматов, но только для случая автоматов детерминированных и всюду определённых по стимулам.

Нам же с самого начала пришлось иметь дело с автоматами частично определёнными и недетерминированными.

Дело в том, что мы опирались на, так называемые, контрактные спецификации, построенные как пред- и пост-условия операций.

Предусловие, если оно не тождественно истинно, в автоматной модели означает отсутствие в том или ином состоянии перехода по тому или иному стимулу. Иначе говоря, сразу требует частично определённых автоматов.

Постусловие – это предикат, которому должны удовлетворять реакция и постсостояние для данных стимула и пресостояния. Тем самым, пресостояние и стимул, вообще говоря, неоднозначно определяют реакцию и постсостояние, что и означает недетерминизм.

Слайд 5. Тестирование как обход графа. Тестирование с закрытым и открытым состоянием.

Прежде всего, нужно сказать о двух принципиально различных направлениях: тестирование с закрытым состоянием и тестирование с открытым состоянием.

Тестирование с закрытым состоянием – это тестирование "чёрного ящика": можно только подавать на автомат стимулы и наблюдать ответные реакции.

При тестировании с открытым состоянием дополнительно можно наблюдать текущее состояние реализации. Я бы назвал это "чёрный ящик в белый горошек", потому что названия "белый ящик" и "серый ящик" используются для других целей.

Видимость состояния реализации очень упрощает жизнь, потому что тестирование, фактически, сводится к обходу неизвестного графа, а для этого существует много хороших алгоритмов полиномиальной сложности.

Разумеется, граф должен быть сильно-связным. Это требование выполнено автоматически, если в каждом состоянии есть рестарт, гарантированно возвращающий в начальное состояние.

При тестировании с закрытым состоянием используются разного рода последовательности. Вот они тут на слайде перечислены.

Ну, об этом студенты Нины Владимировны Евтушенко, наверное, знают лучше меня.

Мы же пошли с самого начала другим путём: нам хотелось, по-прежнему, всего-навсего обходить граф, потому что это дёшево и быстро.

Но какой граф, если состояния реализации не видны?

Мы обходили граф спецификации, а для того, чтобы такое тестирование могло считаться полным, приходилось опираться на разного рода гипотезы о том, как устроена реализация.

Простейший пример такой гипотезы – это когда реализация отличается от спецификации только реакциями на переходах.

Слайд 6. Частично определённые автоматы

Когда речь заходит о частично определённых автоматах, прежде всего, встаёт вопрос: что означает отсутствие в состоянии перехода по стимулу.

На этот вопрос есть три разных ответа.

Ответ первый: В спецификации это означает отсутствие требований к реализации: нет смысла подавать такой стимул в реализацию, поскольку мы просто не указали, что реализация должна или не должна делать, получая такой стимул. Реализация имеет право на любое поведение.

Второй ответ отличается от первого тем, что в "любое поведение" включается также и такое поведение, которого лучше бы избегать при тестировании.

Например, реализация может разрушиться, поломаться, взорваться и так далее.

Для такого поведения мы ввели специальное новое действие, которое назвали разрушением и обозначили символом γ.

Если после приёма стимула в реализации возможно разрушение, то такой стимул подавать нельзя. Он считается опасным. При тестировании можно подавать только безопасные стимулы, и такое тестирование мы назвали безопасным.

Именно второй ответ мы использовали для интерпретации предусловия.

Третий ответ: Это новое наблюдение.

Отсутствие перехода по стимулу называется блокировкой стимула. Оно относится к классу, так называемых, отказов.

Отказ – это отсутствие действий, когда реализация стоит и ничего не делает. В данном случае – не принимает посланный ей стимул.

Если такую блокировку можно наблюдать, то это новое наблюдение и, соответственно, новая конформность.

Такой стимул нужно обязательно подавать, чтобы проверить это новое наблюдение по спецификации.

Слайд 7. Концепция безопасного тестирования

Вот на этом слайде показаны опасности, которых следует избегать при безопасном тестировании.

Ну, **разрушение** опасно по определению.

Блокировка стимула и вообще любой отказ опасен, если он не наблюдаем. Послав такой стимул, мы будем бесконечно долго ждать, когда он будет принят.

Тут нужно сразу оговорить, что речь идёт о синхронном тестировании, когда между тестом и реализацией нет никаких буферных очередей. Если тест посылает стимул, а реализация его блокирует, и эта блокировка не наблюдаема, возникает deadlock: тест не может сдвинуться с места.

Третья опасность связана с **дивергенцией**, то есть бесконечным выполнением ненаблюдаемых переходов. В конечных автоматах таких переходов не бывает, так что это я немного забегаю вперёд.

В чём здесь опасность? В том, что тест бесконечно долго ждёт наблюдения, а его не будет, потому что реализация что-то такое внутри себя делает, но эти её действия не наблюдаемы.

Слайд 8. Недетерминизм

Недетерминизм может означать совсем разные вещи в спецификации и в реализации.

В спецификации недетерминизм часто означает вовсе не то, что реализация может или, тем более, должна быть недетерминированной, то есть выдавать разные реакции на один и тот же стимул в одном и том же состоянии.

Мы даже можем считать, что реализация детерминирована, но спецификация всё равно останется недетерминированной. Почему?

Потому что такой недетерминизм означает неоднозначность требований: спецификация не предписывает реализации какую-то

конкретную реакцию, а разрешает любую реакцию на выбор – но только из указанного в спецификации перечня.

Если реализация детерминирована, то тестирование по недетерминированной спецификации будет полным, если мы в каждом состоянии реализации попробуем каждый стимул один раз, а вовсе не столько раз, сколько реакций на этот стимул написано в спецификации.

При тестировании с закрытым состоянием мы, как я уже говорил, принимаем подходящую гипотезу о реализации и пытаемся в каждом состоянии спецификации дать каждый определённый в нём стимул.

Мы назвали это обходом по стимулам или Δ-обходом графа.

Вместо сильной связности графа требуется сильно-Δ-связность. Она означает, что адаптивный алгоритм, т.е. алгоритм, учитывающий получаемые реакции, может гарантированно попасть из любого состояния в любое другое состояние при любом недетерминированном выборе той или иной из имеющихся реакций в ответ на стимул.

В реализации недетерминизм – так сказать, "настоящий", это её как бы "природное" свойство. Реализация может выдать в том же состоянии в ответ на тот же стимул то одну реакцию, то другую – в зависимости от погодных условий.

Если не принимать никаких дополнительных гипотез о реализации, полное тестирование невозможно: реализация может бесконечно долго выдавать одну реакцию, и не выдавать другую, хотя та тоже определена в реализации.

Гипотеза о глобальном тестировании предполагает, что такого не бывает: если всё время подавать данный стимул в данном состоянии реализации, то рано или поздно реализация выдаст все реакции.

Но для конечного тестирования этого мало. Нам нужна гарантия, что все реакции будут выданы не неизвестно когда, а через ограниченное время.

Для этого используется гипотеза об ограниченном недетерминизме: стимул достаточно подавать число раз, ограниченное константой t.

При тестировании с закрытым состоянием и подходящих гипотезах о реализации для конечного тестирования мы должны, грубо говоря, выполнить Δ-обход *t* раз.

Слайд 9. Факторизация

Одним из способов борьбы с недетерминизмом спецификации является факторизация.

Она основана на фактор-гипотезе об эквивалентности состояний, переходов, или стимулов. По отношению эквивалентности строится фактор-граф спецификации, который уже может быть детерминированным.

Хотя стоит отметить, что основное назначение факторизации – это уменьшение размера спецификации.

Простейший пример: тестирование операции с параметром целого типа. Вместо того чтобы перебирать все возможные значения целого типа, обычно принимаются некоторые обоснованные гипотезы о том, что достаточно выбрать всего несколько значений: крайние, близкие к крайним, средние и т.п.

В UniTESKe факторизация – это составная часть тестовой системы. Результатом является, так называемая, тестовая модель, по которой и генерируются тесты. Правда, оракул, используемый тестом и проверяющий правильность реакций, генерируется всё же на основании исходной контрактной спецификации.

Слайд 10.IOLTS: Input/Output Labelled Transition System

Первые годы конечные автоматы нас вполне устраивали. Однако вскоре нам пришлось столкнуться с системой, которая адекватно не моделировалась конечным автоматом.

Это была многопроцессная программная система, интерфейс с которой основан на обмене сообщениями. В ответ на входное сообщение система могла выдать несколько выходных сообщений, причем, если в процессе их выдачи поступало следующее входное сообщение, оно могло изменить выходной поток.

Так возникла модель автомата, в которой переход помечен не парой стимул-реакция, а либо стимулом, либо реакцией, либо символом τ для обозначения ненаблюдаемого перехода.

Такая модель называется Input/Output LTS.

Но тогда мы этого не знали и придумали своё собственное название: асинхронные автоматы.

Асинхронными они назывались потому, что, фактически, мы тестировали их в контексте входной очереди стимулов и выходной очереди реакций. Применялись даже автоматы с несколькими такими очередями.

Для асинхронных автоматов мы использовали новое наблюдение, которое называли стационарностью и которое совпадает с известным сегодня δ-наблюдением или *quiescence* – наблюдением отсутствия реакций.

Важным отличием от обычных IOLTS было использование приоритетов. Для той системы, о которой я говорил, нам нужно было, чтобы стимул прерывал поток выдаваемых реакций, то есть был более приоритетным. Идея приоритетов нам пригодилась и позже для более общего случая.

В качестве курьёза можно отметить, что в некоторых асинхронных автоматах использовались специальные переходы по отсутствию стимула. Но это не получило дальнейшего развития.

В общем, асинхронные автоматы оказались промежуточным звеном между конечными автоматами и LTS: сначала Input/Output LTS, а потом и LTS общего вида.

Слайд 11.*ioco* : проблемы

Для IOLTS наиболее популярно отношение конформности ***ioco***, которое придумал Ян Тритманс.

Наблюдать можно реакции и *quiescence* – отсутствие реакций. Блокировки стимулов ненаблюдаемы. Поэтому реализация предполагается без блокировок стимулов и без дивергенции.

Однако в спецификации блокировки допускаются. При тестировании стимул не подаётся только в том случае, когда трасса не продолжается стимулом, а только блокировкой.

При ***ioco***-тестировании проверяется, что наблюдение, т.е. реакция или отсутствие реакций, которое есть после трассы в реализации, есть после этой трассы и в спецификации.

С ***ioco*** связаны две серьёзные проблемы:

Первая проблема – это нерефлексивность и нетранзитивность ***ioco***, что является прямым следствием запрета на блокировки в реализации и допущение их в спецификации.

В чём проблема? В том, что спецификацию обычно понимают как модель правильного поведения, однако реализация, буквально списанная со спецификации, оказывается неконформной, поскольку в ней могут быть блокировки стимулов и, тем самым, она просто не попадает в домен ***ioco***. Кроме того, нерефлексивность и нетранзитивность конформности препятствуют применению методологии последовательного изменения уровня абстракции спецификации, что бывает необходимо при проектировании и тестировании.

Вторая проблема – это немонотонность ***ioco***, то есть несохранение конформности при композиции. А именно: композиция реализаций, конформных своим спецификациям, оказывается неконформной композиции этих спецификаций. Это препятствует построению правильной композиции спецификаций и затрудняет решение проблемы декомпозиции системных требований, то есть проверки того, что имеющаяся спецификация системы правильно разложена на спецификации её компонентов.

Важный частный случай проблемы композиции – это проблема асинхронного тестирования, т.е. тестирования в контексте. Здесь она проявляется в том, что асинхронные тесты ловят "ложные" ошибки.

Для решения этих проблем требуются соответствующие эквивалентные преобразования спецификации.

Пополнение даёт всюду определённую по стимулам спецификацию.

А монотонное преобразование гарантирует, что композиция конформных реализаций будет конформна композиции монотонно преобразованных спецификаций.

Такие преобразования были нами предложены, и разработаны соответствующие эффективные алгоритмы. Однако для этого нам пришлось немного выйти за рамки ***ioco***.

Слайд 12.ioco : Безопасное тестирование

Прежде всего, заметим, что безопасность стимула оказывается разной для реализации и для спецификации.

В реализации стимул безопасен, если после трассы нет блокировки стимула. Именно поэтому Тритманс требует всюду определённости реализации по стимулам.

Однако в спецификации стимул безопасен, если трасса продолжается стимулом, и не важно, продолжается она также в другом состоянии блокировкой этого стимула, или нет.

В общем, если после трассы нет стимула, то это эквивалентно тому, что трасса продолжается стимулом и далее разрушением. Результат один: стимул не подаётся при тестировании.

Поэтому требование всюду определённости реализации по стимулам избыточно – ведь после некоторых трасс стимул не подаётся. Из-за этого две реализации, отличающиеся только поведением по поводу такого стимула, не различимы при тестировании. Но одна из них может быть конформна, а другая – нет просто потому, что в ней стимул блокируется, и она не попадает в домен отношения ***ioco***.

Гипотеза о безопасности стимулов расширяет домен реализаций ***ioco***: если в спецификации после трассы есть стимул, то он должен быть после этой трассы и в реализации. А если нет, то в реализации может быть и блокировка стимула.

Слайд 13.$ioco_{\beta\gamma\delta}$: Наблюдаемые блокировки стимулов

На модификации ***ioco*** гипотезой о безопасности мы не остановились. Мы стали исследовать ***ioco***-подобную конформность, но уже с наблюдаемыми блокировками стимулов. Мы назвали её $ioco_{\beta\gamma\delta}$:

δ – это наблюдение отсутствия реакций, как для ***ioco***,

γ – это разрушение,

β – это наблюдаемые блокировки стимулов.

Дивергенцию мы в то время трактовали как разрушение, и только позже поняли, что опасна не сама дивергенция, а попытка выхода из неё, т.е. тестовое воздействие во время дивергенции.

Для спецификаций без дивергенции, блокировок и разрушения ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$, очевидно, совпадают.

Пополнение для ***ioco*** даёт спецификацию без блокировок. Поэтому для пополненной спецификации эти конформности тоже совпадают.

В отличие от ***ioco***, $\boldsymbol{ioco}_{\beta\gamma\delta}$ рефлексивно и транзитивно, т.е. это предпорядок.

Однако проблема сохранения конформности при композиции сохраняется.

Нам удалось разработать алгоритм монотонного преобразования для $\boldsymbol{ioco}_{\beta\gamma\delta}$, а для ***ioco*** – его упрощённую версию.

Слайд 14.LTS общего вида и *RQ*-семантика**

Мы переходим от IOLTS к LTS общего вида, в которой переходы помечены действиями из произвольного заданного алфавита ***L*** или символом τ.

Тестовое воздействие сводится к разрешению реализации выполнять действия из некоторого множества действий, определяемого этим тестовым воздействием.

Наблюдения делятся на два типа: наблюдается либо выполняемое действие, либо отказ – как отсутствие действий.

Ван Глаббек систематизировал конформности, определяемые такой семантикой взаимодействия. Но, как ни странно, в его классификацию не попадает отношение ***ioco***. Почему?

Да потому, что у Ван Глаббека тестовое воздействие – это любое подмножество алфавита действий. А, кроме того, либо все отказы наблюдаемы, либо все ненаблюдаемы. ***ioco*** этому не удовлетворяет: здесь тестовые воздействия не любые – только подача стимула или приём всех реакций, среди отказов есть наблюдаемые – *quiescence* и ненаблюдаемые – блокировки стимулов.

Обобщая это, мы предложили ***R***/***Q***-семантику, в которое тестовое воздействие принадлежит либо семейству ***R***, либо семейству ***Q***. Отказы из ***R*** наблюдаемые, а из ***Q*** – ненаблюдаемые. На самом деле, это класс семантик, параметризуемых семействами ***R*** и ***Q***.

Он включает не только ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$, но и многие другие трассовые конформности: трассовый предпорядок, failure trace семантику и прочее.

Слайд 15.*RQ*-семантика: Безопасность**

Как и для ***ioco***, в ***R***/***Q***-семантике безопасность определяется по-разному в реализации и в спецификации.

В реализации это отношение **safe_in**. Тестовое воздействие безопасно, если не возникает ни одна из трёх опасностей тестирования.

Отношение безопасности в спецификации называется **safe_by**. Оно отличается для ненаблюдаемых отказов, также как в случае **ioco** – для ненаблюдаемых блокировок стимулов.

В отличие от **ioco**, в общем случае действие может разрешаться несколькими тестовыми воздействиями, которые не приводят к разрушению и не подаются при дивергенции. Мы требуем, чтобы такое действие разрешалось хотя бы одним безопасным тестовым воздействием. Но не требуем, чтобы все такие тестовые воздействия были безопасными.

Эти требования к отношению **safe_by** минимальны, но определяют его неоднозначно. Поэтому оно должно задаваться дополнительно к самой LTS-модели спецификации.

Слайд 16.*RQ*-семантика: Конформность *saco***

При безопасном тестировании используются только безопасные трассы, в которых каждое наблюдение получается в ответ на безопасное тестовое воздействие.

Тесты генерируются по безопасным трассам спецификации. Для того чтобы гарантировать безопасность тестирования принимается гипотеза о безопасности. Она требует, чтобы после трассы, которая безопасна и в реализации и в спецификации, любое тестовое воздействие, которое безопасно в спецификации, было безопасным и в реализации.

Конформность мы называли **saco** – SAfe COnformance. Как обычно для редукции, требуется, чтобы всё, что наблюдается в реализации, было определено в спецификации. Но тестирование должно быть безопасным. Поэтому считаются только такие

наблюдения, которые получены в ответ на тестовые воздействия, *безопасные* в спецификации после *безопасных* трасс. По гипотезе о безопасности такие тестовые воздействия и трассы будут безопасными и в реализации.

Слайд 17.*RQ*-семантика: Полное тестирование**
Слайд 18.*RQ*-семантика: Полное тестирование**

Гипотеза о глобальном тестировании гарантирует, что любая ошибка может быть найдена за конечное время. Однако, если у нас нет других гипотез о реализации, то конформность, т.е. отсутствие ошибок, за конечное время доказать не удаётся.

Ограничение на число состояний реализации делает полный набор тестов конечным, но каждый тест всё равно нужно прогонять бесконечное число раз.

А для того, чтобы тест прогонять ограниченное число раз, нужна уже гипотеза об ограниченном недетерминизме.

При тестировании с открытым состоянием нам не нужно ограничивать число состояний реализации, лишь бы оно было конечным. Но гипотеза об ограниченном недетерминизме всё равно нужна.

Для этого случая мы разработали алгоритм конечного полного тестирования с открытым состоянием.

Алгоритм состоит из двух частей, которые выполняются параллельно. Одна часть – это обход графа реализации, другая часть – верификация конформности.

Можно отметить, что при безопасном тестировании мы будем обходить не весь граф реализации, а только ту его часть, которая покрывается трассами, безопасными в спецификации.

Слайд 19.*RQ*-семантика: удаление ненаблюдаемых отказов**

Отношение **saco**, обобщая отношение **ioco**, наследует и связанные с **ioco** проблемы.

Из-за ненаблюдаемых отказов **saco** нерефлексивно и нетранзитивно. Разработан алгоритм пополнения, который удаляет из спецификации ненаблюдаемые отказы. Однако здесь проблема: пополнение не всегда можно сделать в той же ***R***/***Q***-семантике.

Гипотеза о безопасности и конформность задаются тройкой: 1) семантика, 2) LTS-модель, 3) отношение **safe_by**.

Для тестирования важны ***L***-эквивалентные семантики, которые определяют одинаковые тестовые возможности по управлению и наблюдению для реализаций в исходном алфавите ***L***.

Для пополнения и потом для монотонного преобразования используются ***L***-вложенные преобразования, которые:

во-первых, меняют семантику на ***L***-эквивалентную ей,

во-вторых, сохраняют класс конформных реализаций,

в-третьих, не сужают класс тех реализаций, которые можно безопасно тестировать.

Пополнение удаётся сделать с помощью добавления специальных действий, которые мы назвали "не-отказами". Если трасса продолжается ненаблюдаемым отказом, но не продолжается действиями из него, добавляется продолжение соответствующим "не-отказом".

Слайд 20.*R\Q*-семантика: Проблема композиции

Глубинная причина несохранения конформности при композиции – в различии уровней абстракции конформности и композиции. Конформность определена только через трассы, а композиция дополнительно использует состояния.

Проблема в том, что на трассах наблюдений невозможно определить такую композицию трасс, которая обладала бы свойством аддитивности. Это свойство такое: множество трасс композиции LTS равно множеству всех попарных композиций трасс LTS-операндов.

Поэтому вместо трасс наблюдений мы использовали, так называемые, ϕ-трассы. Они похожи на трассы готовности (ready traces), только вместо множества готовности используется его дополнение. По ϕ-трассам LTS легко вычисляются все её трассы наблюдений, но не наоборот!

ϕ-трассы занимают промежуточный уровень абстракции между множеством трасс наблюдений и LTS. Это уровень уже достаточен для того, чтобы выполнялось свойство аддитивности.

Для решения проблемы композиции на первом шаге строится пополнение спецификации, а на втором – собственно монотонное преобразование. Оба преобразования ***L***-вложенные.

Слайд 21.*RQ*-семантика: Удаление неконформных трасс**

Мы уже говорили, что из-за ненаблюдаемых отказов в спецификации конформность оказывается нерефлексивной.

А это значит, что в спецификации есть неконформные трассы.

Понятно, что такие трассы не нужны для генерации тестов. Можно оптимизировать тестирование, если сразу выносить вердикт о неконформности, как только в реализации обнаружена неконформная трасса, даже если такая трасса есть в спецификации.

В общем, нужно проанализировать спецификацию, найти в ней все неконформные трассы и удалить их.

Пополнение, казалось бы, решает эту проблему, удаляя из спецификации ненаблюдаемые отказы. Ведь если в спецификации нет ***Q***-отказов, она конформна сама себе.

Для ***ioco*** этого оказывается достаточно, поскольку пополнение для ***ioco*** не меняет семантику.

Но в общем случае, как я уже говорил, не удаётся сделать пополнение в той же семантике и приходится расширять алфавит действий. А это приводит к расширению класса конформных реализаций, уже в расширенном алфавите.

Из-за этого может оказаться, что трасса спецификации конформна, то есть встречается в конформных реализациях, но – все эти реализации добавленные, т.е. не в исходном алфавите. Тем самым, эта трасса не конформна на исходном классе реализаций.

Поэтому после пополнения приходится делать дополнительное, естественно, ***L***-вложенное, преобразование для удаления неконформных трасс.

Слайд 22.*RQ*-семантика: Финальная модель спецификации**

LTS-модель удобна как способ конечного представления регулярных множеств трасс. Однако она обладает серьёзными недостатками.

Во-первых, недетерминизм. Это прямое следствие неявного представления отказов. Если трасса продолжается и отказом P, и действием из P, она не может заканчиваться в одном состоянии.

Во-вторых, для ***R***/***Q***-семантики приходится дополнительно задавать отношение безопасности **safe_by**.

Существует простая процедура детерминизации LTS. Однако для ***R***/***Q***-семантики появляются переходы по ***R***-отказам, и это не петли. Кроме того, теряется информация о дивергенции.

На основе этой процедуры мы определили новую модель спецификации, которую назвали RTS – Refusal Transition System. Кроме переходов по действиям и ***R***-отказам, в ней есть переходы по разрушению и явные переходы по символу Δ большая для отображения дивергенции.

Также определено преобразование спецификации в особую, финальную RTS, которая обладает уже целым рядом свойств, полезных для генерации тестов.

Во-первых, она детерминированная.

Во-вторых, отношение **safe_by** не нужно задавать дополнительно.

В-третьих, очень легко определяются безопасные трассы и безопасные тестовые воздействия.

Любую LTS-спецификацию с любым отношением **safe_by** можно преобразовать в эквивалентную ей (по классам безопасных и конформных реализаций) финальную RTS-спецификацию. Предложен алгоритм такого преобразования, который строит конечную финальную RTS по конечной LTS и конечному порождающему графу регулярного отношения **safe_by**.

Слайд 23. Расширение *RQ*-модели: медиаторы**

Теперь я хочу рассказать о трёх расширениях ***R***/***Q***-модели: тестирование с преобразованием семантик, приоритеты и симуляция.

До сих пор мы считали, что реализация и спецификация рассматриваются в одной семантике. Однако на практике это часто не так: спецификация всё-таки абстрагируется от деталей программно-аппаратной среды, в которой выполняется реализация.

Поэтому требуются преобразования тестовых воздействий спецификации в тестовые воздействия реализации, и обратное преобразование наблюдений.

Такое преобразование делает специальная программа-медиатор.

Тестирование с преобразованием семантик похоже на тестирование в контексте: в обоих случаях взаимодействие с реализацией опосредованно: средой или медиаторными преобразованиями. Но всё-таки при тестировании в контексте семантика не меняется.

Зато семантика меняется при факторизации. Но в этом случае используется фактор-гипотеза об эквивалентности, а для медиаторного тестирования она не нужна.

При тестировании с открытым состоянием появляется ещё и преобразование состояний: состояние реализации – в состояние спецификации.

Для тестирования с медиатором соответствующим образом модифицируются гипотеза о безопасности и конформность **saco**. Предлагается модификация алгоритма полного тестирования с открытым состоянием ограниченно недетерминированных реализаций. Но здесь уже ограничение на число повторений тестового воздействия перестаёт быть константой, поскольку зависит от медиаторного преобразования. Вместо него вводится ответ медиатора «все наблюдения получены».

Слайд 24. Расширение *RQ*-модели: приоритеты (1)**

Второе расширение ***R***/***Q***-модели – это приоритеты.

В теории приоритетов обычно нет: любое действие, разрешённое тестовым воздействием и определённое в текущем состоянии реализации, может быть выполнено независимо от того, какие ещё действия разрешены и определены.

Тем самым, вносится излишний недетерминизм в описание систем. Почему лишний? Потому что на практике приоритеты используются достаточно часто. Вот лишь несколько примеров:

1) Приоритетная обработка запросов или сообщений, в том числе аппаратных прерываний. Здесь имеются взаимные приоритеты разных стимулов.

2) Выход из дивергенции, когда запрос, поступающий извне, прерывает внутреннюю активность системы, хотя без этого запроса она могла бы продолжаться бесконечно. Этому соответствует приоритет стимула над дивергенцией.

3) Прерывание цепочки действий – операция cansel. Такая операция должна быть приоритетнее действий в цепочке, иначе она может ничего не прервать.

Как ввести приоритеты в LTS?

Мы предлагаем переход по действию помечать дополнительно тестовым воздействием, разрешающим это действие. Только при таком тестовом воздействии этот переход будет выполняться.

Для сокращения записи можно ввести булевские переменные, соответствующие всем действиям. Вместо тестового воздействия на переходе пишется предикат, а именно: конъюнкция, в которую входят переменные, разрешаемые этим тестовым воздействием, и отрицания остальных переменных. Тогда кратные переходы по одному действию заменяются на один переход: на нём пишется дизъюнкция конъюнкций на кратных переходах.

Слайд 25.Расширение *R\Q*-модели: приоритеты (2)

Что меняется после введения приоритетов?

Во-первых, понятия дивергенции и стабильности становятся условными. При одном тестовом воздействии имеется дивергенция или стабильность состояния, а при другом – нет.

Во-вторых, меняется понятие трассы. Без приоритетов нам не важно, каким именно тестовым воздействием вызвано данное наблюдение. При наличии приоритетов это уже становится важным. Поэтому используются трассы, содержащие как наблюдения, так и тестовые воздействия.

В-третьих, меняется оператор параллельной композиции LTS. Предикат синхронного перехода композиции вычисляется по предикатам переходов-операндов. Кроме того, предикаты всех переходов композиции приводятся к алфавиту композиции.

Слайд 26.Расширение *RQ*-модели: слабая симуляция (1)**

Третье расширение ***R*/*Q***-модели – это симуляция вместо редукции.

Выбор симуляции в качестве конформности наиболее естественен, когда состояния реализации доступны для наблюдения, т.е. при тестировании с открытым состоянием.

Мы исследовали наиболее практический вариант симуляции – слабую или наблюдаемую симуляцию. Такая симуляция основана, как и трассовая конформность, на принципиальной ненаблюдаемости τ-переходов.

Слабая симуляция – это существование такого соответствия состояний реализации и спецификации, которое согласовано с наблюдаемыми в этих состояниях действиями.

В ***R*/*Q***-семантике слабая симуляция, прежде всего, меняется за счёт того, что наблюдаться могут не только действия, но и отказы.

Слайд 27.Расширение *RQ*-модели: слабая симуляция (2)**

Кроме того, для безопасного тестирования нам нужна соответствующая гипотеза о безопасности.

Для **saco** безопасность тестового воздействия определяется после трассы, т.е. в каждом состоянии после трассы.

Если состояние наблюдается, то достаточно, чтобы тестовое воздействие было безопасно в наблюдаемом состоянии реализации.

Гипотеза о безопасности использует соответствие состояний, которое мы обозначили буквой *H*. Состояния *H*-соответствуют друг другу, если они достижимы с помощью тестовых воздействий, которые безопасны в состояниях, через которые проходит трасса, как в реализации, так и в спецификации.

Такую гипотезу мы назвали *H*-гипотезой и обозначили **H-safe**. Она требует следующее: если тестовое воздействие безопасно в состоянии спецификации, то оно должно быть безопасно в *H*-соответствующем состоянии реализации.

Заметим, что эта гипотеза сильнее, чем трассовая гипотеза о безопасности **safe_for**.

Слайд 28. Расширение *RQ*-модели: слабая симуляция (3)**

Безопасную симуляцию мы обозначили как **ss**. Она требует выполнения *H*-гипотезы. Кроме этого, она отличается от обычной симуляции тем, что учитываются только те тестовые воздействия, которые безопасны в спецификации. По *H*-гипотезе они будут безопасны и в реализации.

Можно также рассматривать симуляцию не с *H*-гипотезой, а с более слабой трассовой гипотезой **safe_for**. Мы обозначили её как **sst**. Она занимает промежуточное положение между **ss** и **saco.**

Слайд 29. Расширение *RQ*-модели: слабая симуляция (4)**

Как тестировать симуляцию?

Для редукции неконформность реализации можно определить за конечное время. Но для симуляции это не так в общем случае.

Тем не менее, для безопасной симуляции мы придумали общий алгоритм *значимого* тестирования, т.е. тестирования, которое не находит ложных ошибок.

На некотором подклассе спецификаций алгоритм делает полное тестирование. Этот подкласс включает все конечные спецификации в конечных алфавитах.

Алгоритм сначала выполняет обход графа реализации, точнее, подграфа, покрываемого трассами, безопасными в спецификации.

А затем без тестирования строится соответствие состояний.

Если его удалось простроить, реализация конформна, иначе – неконформна.

Слайд 30. Критика R\Q-модели (1)

Мы потратили на R/Q-семантику несколько лет жизни. И казалось, что всё хорошо: эта семантика обобщила многие известные конформности, что позволило единообразно ставить и решать многие общие проблемы. Основными из них являются проблемы оптимизации генерации тестов, композиции систем и учёта приоритетов. В то же время в решении этих проблем выявились серьёзные трудности: алгоритмы пополнения спецификации, удаления из спецификации неконформных трасс и монотонного преобразования оказались довольно сложными и громоздкими, что затрудняет их применение на практике.

Анализ этих трудностей показал, что их причины лежат достаточно глубоко: в самой семантике взаимодействия и выбранных моделях реализации и спецификации. При всей их общности R/Q-семантика и соответствующие ей модели недостаточно общи. Дело в том, что в них существуют внутренние зависимости, которые и приводят к отмеченным усложнениям и трудностям.

Во-первых, нет «просто» наблюдений. Есть действия и отказы с разной семантикой. После отказа P не может наблюдаться действие из P. Отказ может быть только в стабильном состоянии. И тому подобное.

Во-вторых, нет «просто» тестовых воздействий. Тестовое воздействие P разрешает выполнение и, следовательно, наблюдение только действий из P или отказа P, если он наблюдаемый.

В-третьих, представление спецификационных требований в виде модели того же типа, что модель реализации, обосновано только исторически – оно возникло из идеи об эквивалентности автоматов. Но такое представление затрудняет генерацию тестов.

Слайд 31.Критика R\Q-модели (2)

Возможность и необходимость оптимизации тестов – это следствие ***зависимостей между ошибками***.

Спецификация задаёт *неявно* множество A всех ошибок: реализация конформна, если в ней ни одной ошибки из A.

Зависимость между ошибками означает, что существует B строго вложенное в A такое, что любая реализация, содержащая ошибку из A, содержит и ошибку из B.

А тогда достаточно тех тестов, которые ловят ошибки только из множества B, а не из большего множества A.

Следует только отметить, что, поскольку мы имеем дело с трассами и ошибка – это тоже трасса, существуют две неустранимые тривиальные зависимости между ошибками.

1. Тестовое воздействие можно делать всегда, хотя не всегда безопасно. Поэтому если тестовое воздействие после трассы – это ошибка, то и сама трасса ошибочна.

2. Если префикс трассы ошибка, то сама трасса тоже ошибка.

Однако в ***R***/***Q***-семантике есть и ***НЕ***тривиальные зависимости между ошибками. Мы уже можем оптимизировать тесты, удаляя неконформные трассы. Однако в полном объёме проблема зависимостей между ошибками, не решена.

Слайд 32.Критика R\Q-модели (3)

Четвёртым недостатком ***R***/***Q***-модели является то, что спецификации приходится компоновать по тем же правилам, что и реализации, поскольку они относятся к одному типу моделей. А такая композиция приводит к несохранению конформности.

Наконец, приоритеты. Мы их добавили к ***R***/***Q***-модели, но это именно «добавка», она усложняет модель и алгоритмы. Кроме того, для систем с приоритетами не решена проблема монотонности.

Короче говоря, все эти трудности и сложности нам надоели. И возникла идея исследовать семантику более общего вида, в которой с самого начала есть приоритеты и нет внутренних зависимостей.

Слайд 33.Модель наблюдений: T/O-семантика (1)

В ***R***/***Q***-модели переход помечается действием или τ. Такую LTS можно назвать LTS действий, сокращённо ATS, от *action* – действие.

Теперь рассмотрим семантику, основанную на двух непересекающихся универсумах тестовых воздействий и наблюдений.

Переход помечается тестовым воздействием, или наблюдением, или τ.

Будем называть это LTS наблюдений или сокращённо OTS, от *observation* – наблюдение.

Идея в том, что в процессе взаимодействия с реализацией идёт поток наблюдений, а тестовое воздействие лишь регулирует этот поток.

Мы исходили из двух предположений, которые обычно считаются противоречащими друг другу.

Во-первых, приоритет тестовых воздействий над наблюдениями и дивергенцией.

Во-вторых, реализация всегда может выполнять τ-переходы независимо от тестовых воздействий.

Слайд 34.Модель наблюдений: T/O-семантика (2)

Разрешить это противоречие удалось с помощью вот такого протокола взаимодействия.

Если тестового воздействия нет, реализация выполняет цепочку переходов по наблюдениям и τ-переходов.

Если есть тестовое воздействие *P*, реализация не может выполнять переходы по наблюдениям. Это приоритет тестовых воздействий над наблюдениями.

Реализация выполняет цепочку τ-переходов, но только *конечной* длины, а затем обязательно – *P*-переход. Это приоритет тестового воздействия над дивергенцией.

Для того чтобы это было всегда возможно, отсутствие *P*-перехода в состоянии трактуется как наличие *P*-петли.

Второе новшество – это модель спецификации.

Мы уже говорили, что редукция разделяет все трассы на конформные и ошибочные. Поэтому вполне годится представление спецификации как множества всех ошибочных трасс.

С учётом тривиальных зависимостей достаточно только таких трасс-ошибок, которые, во-первых, не заканчивается тестовым воздействием и, во-вторых, не имеют ошибочных строгих префиксов. Такую спецификацию мы называем *нормализованной*. Алгоритм нормализации очевиден.

OTS-спецификация – это просто порождающий граф регулярного множества ошибок. Состояния, в которых заканчиваются трассы-ошибки, помечены как конечные.

Слайд 35.Модель наблюдений: T/O-семантика (3)

Для модели наблюдений есть естественная гипотеза о безопасности, состоящая из двух частей: лямбда-гипотеза и гамма-гипотеза.

Лямбда-гипотеза утверждает, что время ожидания наблюдения конечно. Если в префикс-замыкании спецификации

после трассы есть наблюдение, то в реализации после этой трассы тоже должно быть какое-то наблюдение.

Гамма-гипотеза утверждает, что при тестировании по трассам спецификации не возникает разрушение реализации

Лямбда-гипотеза сужает класс тестируемых реализаций и вводит новую зависимость между ошибками. А именно: если продолжение трассы каждым наблюдением ошибочно, то и сама трасса ошибочна. Очевидно, что такая трасса не может быть в конформных реализациях. Процедура лямбда-нормализации добавляет такую трассу в спецификацию, а все её продолжения наблюдениями удаляет.

Гамма-гипотеза, конечно, тоже сужает класс тестируемых реализаций, но не вводит новых зависимостей между ошибками.

Слайд 36.Модель наблюдений: T/O-семантика (4)

Через ***T***/***O***-семантику моделируются все другие семантики с трассовыми конформностями типа редукции, в том числе все ***R***/***Q***-семантики, включая ***ioco*** и $\boldsymbol{ioco}_{\beta\gamma\delta}$.

Как показано на слайде, для реализации применяется подходящее моделирующее преобразование, а в качестве спецификации берётся множество ошибок – как трасс тестовых воздействий и наблюдений – обнаруживаемых каким-нибудь полным набором тестов для исходной спецификации.

Слайд 37.Модель наблюдений: T/O-семантика (5)

Важно, однако, что моделируемая семантика ориентирована не на все реализации, допускаемые ***T***/***O***-семантикой, а на строгий подкласс таких реализаций, в том числе, определяемый гипотезой о безопасности для моделируемой семантики.

Именно сужение класса реализаций порождает нетривиальные зависимости между ошибками, что и приводит к трудной проблеме оптимизации тестов.

Тем самым, такая оптимизация для различных семантик – это частный случай общей проблемы оптимизации тестов при сужении класса тестируемых реализаций.

Вот лишь несколько примеров такого сужения, не связанного с моделированием семантик или гипотезой о безопасности:

1. Реализации с ограниченным числом состояний.
2. Заданный (обычно конечный с точностью до изоморфизма) класс реализаций.
3. Заданный (обычно конечный) класс неконформных реализаций. То, что называют *классом неисправностей*.
4. Класс реализаций, где любая неконформная реализация содержит ошибку из заданного конечного множества. В этом случае говорят, что *тесты нацелены на поиск этих ошибок*.

Слайд 38.Модель событий: T/E-семантика (1)

Модель наблюдений всё ещё недостаточно хороша, потому что не решает проблему композиции. Дело в том, что наблюдения, даже самые общие, вообще говоря, ***не аддитивны***.

Поэтому предлагается новая модель событий, сокращённо ETS от *event* – событие. Вместо универсума наблюдений – универсум событий ***E***.

Наблюдения получаются из событий с помощью частично-определённой однозначной функции ***f***.

Переходы совершаются только по *наблюдаемым событиям*, т.е. событиям из домена функции ***f***. При переходе по событию x внешний наблюдатель видит как раз наблюдение $\boldsymbol{f}(x)$.

События придуманы для композиции.

Два перехода в операндах по одному тестовому воздействию выполняются синхронно и порождают композиционный переход по этому же тестовому воздействию.

γ- и τ-переходы в одном операнде сохраняются в композиции, это асинхронные переходы.

События делятся на синхронные и асинхронные, что задаётся алфавитом синхронных событий.

Наконец, задана частично определённая коммутативная композиция синхронных событий.

На этой основе определяется композиция трасс событий и ETS, и доказывается свойство ***аддитивности***.

Слайд 39.Модель событий: T/E-семантика (2)

Это мы определили композицию реализаций. А теперь композиция спецификаций.

Спецификация для модели событий такая же, как для модели наблюдений: нормализованное множество ошибочных трасс наблюдений, или, для регулярных множеств, – OTS-спецификация.

Композиция таких спецификаций определяется на основе следующего достаточно очевидного утверждения:

Множество трасс наблюдений, которые НЕ генерируются с помощью функции ***f*** по трассам событий, встречающимся в композициях конформных реализаций, является композицией спецификаций – как множество ошибок.

Для регулярных множеств применяется аналогичный алгоритм, который по OTS-спецификациям операндов строит OTS-спецификацию композиции.

Слайд 40.Модель событий: T/E-семантика (3)

В ***T/E***-семантике моделируются те же семантики, что для ***T***/***O***-семантики. А также такие семантики, как, например, семантика трасс готовности (ready trace semantics).

Это моделирование согласовано с композицией: Композиция LTS событий, которые получаются в результате моделирования исходных LTS действий с исходными семантиками, совпадает с LTS событий, которая получается в результате моделирования композиции этих LTS действий.

Это изображено на рисунке.

Слайд 41.Параллельное тестирование

В заключение я хочу сказать несколько слов про обход графа. Это самое главное, потому что ещё сорок с лишним лет назад темой моей дипломной работы был обход графа конечным автоматом.

Ну, а если серьёзно, то, как вы могли заметить, почти все теоретические и практические алгоритмы тестирования, которые мы использовали, основаны на обходе графа.

Но сегодня в связи с ростом размера используемых систем и сетей и, следовательно, размера исследуемых графов, возникает задача **распределённого и параллельного обхода графа.**

Граф просто не помещается в оперативной памяти одного компьютера или слишком долго обходится одним компьютером.

К сегодняшнему дню у нас уже разработаны алгоритмы трёх типов для коллектива автоматов, обменивающихся сообщениями:

1. Описание графа строится в памяти каждого компьютера. Тут проблема не в памяти, а во времени – несколько параллельно работающих компьютеров обходят граф быстрее, чем один.
2. Автоматы-движки ползают по графу. Об этом на конференции делал доклад Александр Сергеевич Косачева.

   Описание графа строится в распределённой памяти автоматов-регуляторов.
3. Автоматы сидят в вершинах графа и посылают сообщения по дугам графа. Описание графа строится в распределённой памяти автоматов. Этой задачей мы сейчас занимаемся. В этом году должны написать статью.

Слайд 42.Спасибо за внимание!