И. Бурдонов, А. Косачев
Семантики взаимодействия с отказами, дивергенцией и разрушением. Часть 2
ИСПРАН
27 слайдов

Тестирование и верификация систем на основе формальных моделей.

6. Проблемы практического тестирования

Слайд 1. Сегодня у нас будет продолжение с переходом от теории к практике.

Слайд 2. Вот план доклада.

Слайд 3. 6. Проблемы практического тестирования.
Проблема конечности

Для практического применения конечными по времени должны быть как генерация тестов, так и тестирование по этим тестам.

►Отсюда вытекает конечность полного набора тестов (или единого теста с рестартом). Для этого «почти» необходимы, хотя и недостаточны, конечность алфавита внешних действий $\boldsymbol{L}$ и конечность LTS-спецификации (числа ее переходов), что на практике вполне приемлемо.

►Конечность тестирования опирается на конечность полного набора тестов, конечность времени прогона каждого теста и конечность требуемого числа прогонов каждого теста.

►Конечность времени прогона теста гарантируется для конечного теста «практическими предположениями» о семантике: 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) в реализацию и наблюдения от реализации выполняются за конечное время. Эти предположения гарантируют наблюдение

внешнего действия, выполняемого реализацией, через конечное время после нажатия кнопки, разрешающей это действие.

►Таким образом, остаются две основные проблемы: 1) конечность полного набора тестов и 2) конечность требуемого числа прогонов теста.

Эти проблемы не имеют решения в общем виде, поэтому такие решения приходится искать в частных случаях: либо ограничивая классы рассматриваемых спецификаций и/или реализаций, либо предполагая наличие дополнительных тестовых возможностей, либо сочетая одно с другим.

Слайд 4. 6. Проблемы практического тестирования.
Тестирование с открытым состоянием

Одной из таких возможностей является опрос текущего состояния реализации. Если можно «подсматривать» состояния реализации, говорят о тестировании с открытым состоянием, в противном случае – о тестировании с закрытым состоянием. Дальше мы рассмотрим эти два вида тестирования и соответствующие ограничения на классы реализаций и/или спецификаций.

Слайд 5. 6. Проблемы практического тестирования.
Ограничения на недетерминизм реализации

Проблема конечности требуемого числа прогонов теста общая для этих двух видов тестирования. Гипотеза о глобальном тестировании дает только теоретическую возможность обнаружить любую ошибку в любой неконформной реализации. На практике нам, прежде всего, требуется конечность различимых «погодных условий», а также либо какие-то способы «управления погодой», либо гипотезы, ограничивающие возможные проявления недетерминизма реализации.

►Первое возможно в каких-то частных случаях, например, когда недетерминизм является следствием псевдопараллелизма, то есть псевдопараллельного выполнения нескольких детерминированных

процессов на одном процессоре. Если мы можем вмешиваться в работу планировщика, мы тем самым можем «управлять погодой».

►Второй способ используется чаще всего в его экстремальном виде, когда ограничиваются только детерминированными реализациями (при недетерминированной спецификации).

►Здесь мы должны уточнить понятие детерминизма. Обычно LTS-реализация определяется как детерминированная, если каждая ее трасса заканчивается ровно в одном состоянии. Однако в общем случае для тестирования в ***R/Q***-семантике дополнительно требуется, чтобы в каждом достижимом состоянии для каждой кнопки P было определено не более одного перехода по действию $z \in P$. В терминах тестов это можно сформулировать так: безопасно-тестируемая LTS-реализация детерминирована в ***R/Q***-семантике для заданной спецификации, если каждый тест при любом его прогоне либо заканчивается только с вердиктом ***fail***, быть может, в разных состояниях реализации, либо только с вердиктом ***pass*** и только в одном состоянии реализации.

►В дальнейшем мы исходим из следующей гипотезы о *t-недетерминизме*: если в любом состоянии i реализации любую кнопку P нажимать t раз, то реализация продемонстрирует все возможные варианты поведения, то есть будут получены все возможные пары (наблюдение, постсостояние). При t=1 мы имеем детерминированную реализацию.

Слайд 6. 6. Проблемы практического тестирования.
Ограниченное отношение безопасности кнопок ***safe by***

Остановимся на одном независимом аспекте проблемы конечности полного набора тестов: проблеме определения безопасных трасс в спецификации. Дело в том, что правила для отношения безопасности кнопок ***safe by*** позволяют после различных трасс, заканчивающихся в LTS-спецификации в одном множестве состояний (в детерминированной RTS-спецификации – в одном состоянии), по-разному определять безопасные кнопки.

►При наличии циклов мы получаем бесконечную цепочку конечных трасс с произвольным распределением безопасных кнопок после этих трасс.

►Отношение ***safe by***, при котором трассы, заканчивающиеся в одном множестве состояний LTS-спецификации (в одном состоянии RTS-спецификации), имеют одинаковые множества безопасных после них ***Q***-кнопок (и, тем самым, всех кнопок, поскольку безопасность ***R***-кнопок одинакова для всех таких трасс), будем называть *ограниченным*.

►Это дает нам возможность говорить о безопасности кнопки (и соответствующих наблюдений) во множестве состояний LTS-спецификации (или в одном состоянии RTS-спецификации).

Множество безопасных трасс спецификации ***S*** будем обозначать ***Safe***(***S***).

►Важно, что для LTS-спецификации с конечным числом состояний конечно число множеств состояний (и состояний соответствующей RTS-спецификации).

7. Тестирование с закрытым состоянием

Слайд 7. 7. Тестирование с закрытым состоянием.
Ограничение на семантику и спецификацию

Решение проблемы конечности полного набора тестов будем искать двумя способами: сужая классы рассматриваемых спецификаций или реализаций. Для конечной семантики по каждой трассе генерируется конечное число примитивных тестов (тестов, сгенерированных по одной трассе спецификации). Поэтому достаточно конечности полного набора трасс, что для конечной семантики эквивалентно ограниченности длины трасс полного набора.

►**Ограничение на спецификацию.** Если число ***R***-кнопок конечно, а в конечной спецификации нет циклов, то число трасс спецификации конечно, тем более конечен полный набор трасс. В то же время не любой цикл приводит к бесконечности полного набора трасс.

►Для RTS-спецификации будем называть *демоническим* состояние *s*, после которого не бывает ошибок, то есть любая актуальная (встречающаяся в безопасно-тестируемых реализациях) тестовая трасса $\mu\cdot\lambda$, где трасса μ заканчивается в *s*, не является ошибкой.

►Для того, чтобы для конечной спецификации в конечном алфавите существовал полный набор трасс ограниченной длины необходимо и достаточно, чтобы в RTS-спецификации любой маршрут с безопасной трассой не содержал цикла, проходящего через недемонические состояния. Длина трасс такого полного набора имеет точную верхнюю оценку *K*, где *K* число состояний RTS-спецификации.

Слайд 8. 7. Тестирование с закрытым состоянием.
Ограничение на размер реализации

Ограничение на размер реализации. Путь задана RTS-спецификация ***S*** и рассмотрим некоторую произвольную безопасно-тестируемую LTS-реализацию ***I***.

Пусть σ безопасная трасса спецификации, а *P* - кнопка, безопасная в спецификации после трассы σ.

Обозначим через s_σ (единственное) состояние спецификации, в котором заканчивается трасса σ, а через i_σ – одно из состояний реализации ***I***, в которых заканчивается трасса σ.

►Если для каждой тройки (s_σ,i_σ,P) в наборе тестов будет примитивный тест, сгенерированный по какой-нибудь трассе $\sigma`$ такой, что $s_\sigma= s_{\sigma`}$, в котором после трассы $\sigma`$ нажимается кнопка *P*, то такой набор тестов, очевидно, будет полным для ограниченного отношения ***safe by***.

►При однократном прогоне теста на реализации ***I*** мы получаем последовательность пар состояний (s_μ,i_μ), где μ - префикс трассы σ.

►Будем говорить, что тест *простой* для данной реализации ***I***, если хотя бы при одном его прогоне в полученной последовательности пар состояний нет одинаковых пар.

Очевидно, что для полноты тестирования данной реализации ***I*** достаточно тестов, которые простые для этой реализации. Для полноты тестирования всех реализаций достаточно, чтобы в набор тестов входили все тесты, которые просты для той или иной реализации.

Слайд 9. 7. Тестирование с закрытым состоянием.
Оценка длины простого теста

Оценим сверху длину N простого теста.

Если число состояний реализации не превосходит n, то, очевидно, $N \le nK$.

Если набор тестов состоит из всех тестов длины не больше nK, то такой набор тестов содержит все простые тесты для всех реализаций с числом состояний не больше n.

►Эта верхняя оценка является точной по порядку: для любых k и n существует семантика с O(k) действиями, LTS-спецификация с O(k) состояниями и неконформная реализация с O(n) состояниями, ошибка в которой не может быть обнаружена тестом длины меньше $O(n2^k)$.

►Можно показать, что даже для семантики с ограниченным число действий оценка остается суперполиномиальной.

В частности, существует семантика с двумя действиями, для которой в точной верхней оценке показатель степени k заменяется на $c(\ln k)^2$.

►LTS-спецификации, на которых достигаются эти оценки, существенно недетерминированы.

Понятно, что для детерминированной LTS-спецификации $K=k$ (а не 2^k по порядку).

Однако даже минимальный недетерминизм LTS-спецификации, когда нет τ-переходов и только в одном состоянии только по одному действию имеются два перехода в разные состояния, оставляет оценки суперполиномиальными.

Слайд 10.7. Тестирование с закрытым состоянием.
Ограничение на недетерминизм реализации

►**Ограничение на недетерминизм реализации.** При тестировании с закрытым состоянием нужна усиленная гипотеза о t-недетерминизме: нас интересует число $T(\sigma,P)$ возможных поведений реализации при нажатии кнопки P не в состоянии реализации (которого мы не видим), а после трассы σ.

Если число состояний реализации ограничено числом n, то $T(\sigma,P)\leq t^n$.

Но можно использовать и независимую от n гипотезу о том, что $T(\sigma,P)$ ограничено некоторым числом T.

Такую гипотезу мы вынуждены использовать, если ограничения налагаются только на спецификацию, но не на размер реализации.

►Число прогонов примитивного теста с N кнопками равно $O(T^N)$.

Эта оценка достижима для некоторых реализаций, но, конечно, для каких-то реализаций она может оказаться гораздо меньше.

В частности, для детерминированных реализаций оценка $O(N)$.

8. Тестирование с открытым состоянием

Слайд 11.8. Тестирование с открытым состоянием.
Один адаптивный тест с рестартом

Для тестирования с открытым состоянием мы будем строить не набор тестов, а один адаптивный тест с возможными рестартами в середине теста.

►Для полноты тестирования нужно проверить все переходы реализации, лежащие на маршрутах с трассами, безопасными в спецификации.

►Для этого, поскольку у нас только один тест, LTS-реализация должна быть сильно-связной: из каждого состояния достижимо по переходам каждое другое.

Рестарт понимается как одно из внешних действий, отличающихся только тем, что гарантированно переводит реализацию в начальное состояние.

Переход по рестарту делает трассу пустой.

Переходы по рестарту дополнительно учитываются при определении сильно-связности.

►Также требуется, чтобы начальное состояние реализации было стабильным или хотя бы в одном состоянии, достижимом по безопасной трассе спецификации, был определен рестарт.

►Действительно, в примере реализация в начале тестирования может перейти в состояние 1. После этого без рестарта она попадет в состояние 0 только после трассы $\langle x \rangle$, после которой в спецификации кнопка $\{y\}$ опасна: ее нельзя нажимать при тестировании. Тем самым ошибка в реализации не будет обнаружена.

Слайд 12.8. Тестирование с открытым состоянием.
Структуры данных для RTS-спецификации ***S***

Предварительно строятся структуры для спецификации ***S***, которые не зависят от реализации и используются без модификации для верификации любой реализации в той же ***R***/***Q***-семантике.

Будем предполагать, что спецификация ***S*** задана как RTS.

►Рассматриваются состояния *s* в конце безопасных трасс, для каждого из которых определяем:

$\boldsymbol{A}(s)$ — множество кнопок, безопасных в состоянии s;

$\boldsymbol{B}(s)$ — множество безопасных наблюдений плюс символ τ;

$\boldsymbol{C}(s,u)$ – постсостояние после перехода по наблюдению u из состояния s.

Если таких переходов нет, $\boldsymbol{C}(s,u)=*$, где $*$ не совпадает ни с одним состоянием спецификации.

Доопределим $\boldsymbol{C}(s,\tau)=s$.

Слайд 13.8. Тестирование с открытым состоянием.
Построение модели реализации $\boldsymbol{I}$ *в процессе тестирования*

Получая наблюдения и опрашивая состояния реализации, мы будем поэтапно строить LTS-реализацию с одновременной проверкой тестируемого условия.

Более точно: строится LTS-модель, имеющая такое же как в реализации множество трасс, безопасных в спецификации, и такое же множество состояний, достижимых по этим трассам.

►В начале тестирования и после каждого перехода опрашиваем состояние реализации.

Переход $i—z\rightarrow i`$ по внешнему действию $z$ добавляется, когда после опроса состояния $i$ нажимается кнопка $P$, после чего наблюдается действие $z\in P$, а затем опрашивается постсостояние $i`$.

Если наблюдается отказ с *тем же самым* постсостоянием $i`=i$, то добавляется виртуальный переход-петля по отказу $i—P\rightarrow i$.

Если отказ P наблюдается с *другим* постсостоянием $i`\neq i$, то добавляются переходы $i—\tau\rightarrow i`—P\rightarrow i`$.

Вместе с каждым переходом по внешнему действию $i—z\rightarrow i`$ будем хранить кнопку, нажатие которой вызвало этот переход.

Слайд 14.8. Тестирование с открытым состоянием.
Структуры данных для LTS-реализации ***I*** *(1)*

При построении реализации с каждым построенным состоянием i будем связывать множество $\boldsymbol{S}(i)$. Это состояния спецификации в конце трасс, безопасных в спецификации, имеющихся в реализации и заканчивающихся там в состоянии i.

В процессе тестирования мы будем строить эти множества $\boldsymbol{S}(i)$, постепенно добавляя в них новые состояния.

►Кнопка P *допустима в* i, если она безопасна хотя бы в одном состоянии $s \in \boldsymbol{S}(i)$.

Только допустимые кнопки будут нажиматься в состоянии i.

Для каждой допустимой кнопки P определим счётчик $c(P,i)$ числа ее нажатий в состоянии i.

Кнопка P *полна в состоянии* i, если
1) счетчик=1 и в состоянии i есть виртуальный переход-петля по отказу P, или
2) счетчик=t.

В обоих случаях уже получены все возможные переходы из состояния i при нажатии кнопки P.

Состояние *полно*, если каждая допустимая в нем кнопка полна.

Слайд 15.8. Тестирование с открытым состоянием.
Структуры данных для LTS-реализации ***I*** *(2)*

Кроме $\boldsymbol{S}(i)$ формируются следующие структуры данных для каждого состояния i:
$\boldsymbol{A}(i)=\cup\{\boldsymbol{A}(s)|s\in\boldsymbol{S}(i)\}$ — множество кнопок, допустимых в состоянии i;
$\boldsymbol{D}(i)=\{i—z\rightarrow i`\}$ — множество переходов из состояния i.

►В начале тестирования после опроса состояния в ***I*** есть только одно состояние $i\in(\boldsymbol{I}\ \textbf{after}\ \varepsilon)$, где ε пустая трасса, и для этого состояния

$\boldsymbol{S}(i)=\{s_0\}$, $\boldsymbol{A}(i)=\boldsymbol{A}(s_0)$, $\boldsymbol{D}(i)=\varnothing$ и $c(P,i)=0$ для каждой допустимой кнопки P.

►Тестируемое условие эквивалентно следующему условию в терминах введенных обозначений. Мы будем проверять это условие на каждом шаге тестирования.

Слайд 16.8. Тестирование с открытым состоянием.
Общая схема работы алгоритма тестирования (1)

На рисунке изображена общая схема работы алгоритма.
Сначала проверяем полноту текущего состояния.

►Если текущее состояние полное, то для продолжения тестирования нужно перейти в неполное состояние.

►Если таких состояний нет, алгоритм заканчивается с вердиктом ***pass***.

►Рассмотрим переход в неполное состояние.

Слайд 17.8. Тестирование с открытым состоянием.
Общая схема работы алгоритма тестирования (2)

В графе LTS реализации всегда можно выбрать множество деревьев, покрывающих все состояния так, что из каждого состояния выходит не более одного перехода, принадлежащего деревьям, и все эти деревья ориентированы к своим корням, которыми являются все неполные состояния. Будем двигаться по переходам этих деревьев, нажимая управляющие кнопки этих переходов. Из-за недетерминизма мы можем оказаться не в постсостоянии перехода, а в другом состоянии. Если это не полное состояние, мы снова будем нажимать управляющую кнопку. За конечное число шагов мы гарантированно попадем в неполное состояние.

Слайд 18.8. Тестирование с открытым состоянием.
Общая схема работы алгоритма тестирования (3)

►Если текущее состояние неполное, то выбираем неполную кнопку, нажимаем ее и получаем один переход или два перехода. Постсостояние перехода становится новым текущим состоянием. Корректируем счетчик нажимавшейся кнопки.

►Переход, который мы получили, может быть новым или построенным ранее.

►Если получен старый переход, возвращаемся к началу работы алгоритма.

►Если получен новый переход, то корректируем множество переходов $\boldsymbol{D}(i)$.

После этого выполняется блок «Распространие с верификацией».

Слайд 19.8. Тестирование с открытым состоянием.
Общая схема работы алгоритма тестирования (4)

Для работы этого блока создается вспомогательный список $\boldsymbol{W}$ всех пар (состояние спецификации и переход реализации), где состояние спецификации сопоставлено пресостоянию перехода. В самом начале множество $\boldsymbol{W}$ содержит все такие пары для каждого нового перехода.

Опишем шаг работы блока.

Если список $\boldsymbol{W}$ не пуст, выбираем первый элемент из списка, удаляя его из списка.

►Проверяем для него тестируемое условие.

►Если условие не выполнено, фиксируется ошибка и алгоритм заканчивается с вердиктом ***fail***.

►Если ошибки нет, рассматриваем состояние спецификации в конце перехода из состояния s по наблюдению u – $\boldsymbol{C}(s,u)$.

Если $\boldsymbol{C}(s,u)$ уже сопоставлено постсостоянию $j`$ реализационного перехода по u, то выбираем следующий элемент из списка $\boldsymbol{W}$.

В противном случае сопоставляем состояния спецификации и реализации, добавляя $\boldsymbol{C}(s,u)$ в $\boldsymbol{S}(j`)$,
и помещаем в список $\boldsymbol{W}$ все требуемые пары.
Выбираем следующий элемент из списка $\boldsymbol{W}$.

►Когда список $\boldsymbol{W}$ становится пустым, переходим к началу работы.

Слайд 20.8. Тестирование с открытым состоянием.
Общая схема работы алгоритма тестирования (4)

Отметим, что при таком тестировании верифицируются не только наблюдения, полученные после *реальных* трасс, пройденных при тестировании, но и возможные наблюдения после *потенциальных* трасс, то есть наблюдения и трассы, про которые установлено, что они есть в реализации. Это даёт существенную экономию числа тестовых воздействий, необходимых для проверки конформности: мы выполняем множество проверок без реального тестирования, основываясь на полученном знании о поведении реализации.

Например, если при тестировании получены две трассы $\mu_1 \cdot \lambda$ и μ_2, где трассы μ_1 и μ_2 заканчиваются в реализации в одном состоянии i, то мы можем проверить обе трассы: как $\mu_1 \cdot \lambda$, которую реально прошли при тестировании, так и потенциальную трассу $\mu_2 \cdot \lambda$.

Это преимущество дает дополнительная тестовая возможность опроса состояния реализации.

Слайд 21.8. Тестирование с открытым состоянием.
Оценки сложности алгоритма. Число тестовых воздействий

При тестировании обычно наиболее важным считается число тестовых воздействий.
Оценка экспоненциальная в общем случае и полиномиальная для детерминированных реализаций.

►Такая полиномиальная оценка достигается не только для детерминированной реализации, но и во всех случаях, когда переход в неполное состояние можно гарантированно выполнить, проходя

путь (маршрут без самопересечений), длина которого ограничена *n*. Приведем три таких случая:

►1. Упомянутый выше случай недетерминизма как следствия псевдопараллелизма, когда мы можем вмешиваться в работу планировщика процессов, тем самым «управляя погодой».

►2. Сильно-Δ-связные LTS-реализации. Это такие LTS, в которых для любой пары состояний *a* и *b* можно в каждом состоянии *i* найти такую кнопку *P*(*i*), что, нажимая только такие кнопки, мы гарантированно окажемся в состоянии *b*, хотя путь из *a* в *b*, который мы проходим, зависит от недетерминированного поведения LTS. Детерминированные LTS — это частный случай сильно-Δ-связных LTS.

►3. Недетерминизм может быть следствием повышения уровня абстракции при моделировании детерминированной исследуемой системы такой реализацией, которая оказывается недетерминированной в той семантике взаимодействия, которая используется в спецификации. При тестировании связь уровней абстракции осуществляется промежуточной программой (медиатором). В этом случае при нажатии в состоянии *i* кнопки *P* медиатор может сообщить тесту некий дополнительный параметр, от которого абстрагируется модель. Если есть возможность при повторном нажатии в состоянии *i* кнопки *P* сообщить медиатору этот параметр, то гарантированно выполнится тот же реализационный переход.

Слайд 22.8. Тестирование с открытым состоянием.
Оценки сложности алгоритма. Объем вычислений

Оценка объема вычислений, кроме тестовых воздействий, содержит три слагаемых:
1) вычисления, необходимые для поиска опрошенного состояния среди пройденных при каждом тестовом воздействии, имеют экспоненциальную оценку в общем случае и полиномиальную для детерминированных реализаций;
2) построение множества деревьев имеет полиномиальную оценку;

3) вычисления в блоке «Распространение с верификацией» имеют O от m K большое.

9. Развитие теории конформности

Слайд 23.9. Развитие теории конформности.
Проблема монотонности

Теперь я хочу сказать несколько слов о дальнейшем развитии теории безопасной конформности. Какие-то проблемы здесь уже решены, а какие-то ещё предстоит решить.

Первой такой проблемой является проблема монотонности конформности. Эта проблема возникает в связи с композицией. Композиционная система – это составная система, собранная из компонентов с помощью применения правил параллельной композиции, определенных для LTS-моделей.

Неформально проблема звучит так: если компоненты работают правильно, то почему система в целом работает неправильно?

Формально это означает, что композиция реализаций компонентов, которые конформны спецификациям этих компонентов, не конформна композиции спецификаций компонентов. Это вызвано разными уровнями абстракции, используемыми в определениях конформности и прямой композиции. Конформность основана на трассах наблюдений над поведением реализационной модели, а композиция, кроме того, на ненаблюдаемых напрямую состояниях и ненаблюдаемых действиях (τ-действиях).

Особым случаем композиции является тестирование в контексте, которое можно рассматривать как тестирование системы из двух компонентов, один из которых – реализация, а другой – фиксированная среда взаимодействия. Здесь возможно несохранение конформности, когда ловится «ложная» ошибка, что является частным случаем общей проблемы монотонности.

Решением является такое эквивалентное преобразование спецификаций компонентов, что композиция конформных

реализаций компонентов конформна композиции преобразованных спецификаций компонентов. Такое преобразование называется монотонным. Преобразование пополнения, о котором мы говорили, является первым шагом к монотонному преобразованию, но оно недостаточно для монотонности.

При тестировании в контексте предполагается, что в среде нет ошибок и она известна. Поэтому монотонное преобразование нужно применять только к спецификации реализации, а среда остаётся неизменной. При решении проблемы монотонности учитывается и этот случай.

Если спецификация системы задана независимым образом, то она должна быть *согласованной* со спецификациями компонентов так, чтобы композиция конформных реализаций компонентов была конформна этой спецификации системы. Композиция монотонно преобразованных спецификаций оказывается самой сильной согласованной спецификацией системы, то есть предъявляющей к системе наибольшие требования среди всех согласованных спецификаций.

Здесь возникает вопрос: в какой семантике следует рассматривать композицию монотонно преобразованных спецификаций? Дело в том, что разные компоненты могут быть определены в разных алфавитах и разных семантиках. По счастью, этот вопрос снимается тем, что композиция монотонно преобразованных спецификаций оказывается самой сильной согласованной спецификацией *в любой* семантике. Естественно, для одного и того же композиционного алфавита, однозначно определяемого алфавитами компонентов и не зависящего от семантик компонентов при тех же алфавитах компонентов.

Монотонное преобразование позволяет решить две основные задачи: 1) верификации декомпозиции системных требований, то есть верификация согласованности имеющейся спецификации системы со спецификациями компонентов, и 2) при отсутствии спецификации системы её генерация по спецификациям компонентов.

Я здесь не буду останавливаться на технических деталях: как именно определяется монотонное преобразование, как его сделать алгоритмическим и тому подобное. Это довольно сложное преобразование и рассказ о нем потребовал бы очень много времени.

Слайд 24.9. Развитие теории конформности.
Приоритеты

Теория, которую я здесь рассказывал, да и вообще все существующие теории конформности работают с системами без приоритетов, когда любое действие, разрешённое оператором и определённое в реализации, может быть недетерминированным образом выбрано на выполнение. В то же время на практике часто встречаются системы с приоритетами. Вот только несколько примеров, показывающих важность и полезность приоритетов.

Выход из дивергенции. Если нет приоритетов, то при возникновении дивергенции дальнейшее взаимодействие с системой проблематично, поскольку она может бесконечно долго выполнять только внутренние действия. В то же время дивергенция, как бесконечная внутренняя работа, встречается во многих реальных системах, но при возникновении внешнего воздействия система сразу же на него реагирует. Это происходит из-за того, что внешние действия имеют больший приоритет, чем внутренние τ-действия.

Выход из осцилляции. Под осцилляцией понимается бесконечная цепочка выдачи сообщений системой. Для того, чтобы такую цепочку можно было прервать, заставив систему обрабатывать поступающий извне запрос, последний должен иметь больший приоритет, чем выдача сообщений.

Прерывание цепочки действий. Команда «отменить» (cancel) должна останавливать выполнение последовательности действий, инициированной предыдущим запросом, и вызывать цепочку завершающих действий. Если команда «отменить» не имеет наивысшего приоритета, она может быть выполнена уже после того, как вся обработка закончится, то есть, фактически, ничего «не отменяет».

Приоритетная обработка входных воздействий. Если в систему поступает одновременно несколько запросов, то часто требуется их обработка в соответствии с некоторыми приоритетами между ними. Это реализуется в виде очереди запросов с приоритетами или в виде нескольких очередей запросов с приоритетами между очередями. К этому типу приоритетов относится и обработка аппаратных прерываний в операционной системе.

Нам удалось ввести в теорию приоритеты. Конкретно мы ввели приоритеты в LTS-модель и соответствующим образом модифицировали конформность, генерацию тестов. Также удалось определить параллельную композицию систем с приоритетами. Нерешенными остаются проблемы пополнения и монотонного преобразования для систем с приоритетами.

Слайд 25.9. Развитие теории конформности.
Симуляции

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

Разумеется, тестирование симуляции возможно только в том случае, когда мы можем опрашивать текущее состояние реализации, то есть это всегда тестирование с открытым состоянием.

Нам удалось определить слабую симуляцию с учетом отказов (как наблюдаемых, так и ненаблюдаемых), дивергенции и разрушения. Мы назвали ее безопасной симуляцией. Установлена связь безопасной симуляции с трассовой конформностью ***saco*.**

Теоретически исследовано полное тестирование безопасной симуляции. В отличие от трассовой конформности полное тестирование не всегда возможно. Найдено достаточное условие полноты тестирования и предложен общий алгоритм тестирования безопасной симуляции.

Для практического использования предложена модификация общего алгоритма, которая аналогична алгоритму тестирования с открытым состоянием, о котором я рассказывал, но только верификация выполняется не по ходу исследования реализация, а после завершения такого исследования.

Нерешенными остаются проблемы пополнения и монотонного преобразования для безопасной симуляции.

Слайд 26.9. Развитие теории конформности.
Тестирование с преобразованием семантик

В теории конформности обычно предполагается, что реализация и спецификация заданы в одной семантике. Однако на практике часто требуется некоторое преобразование спецификационных тестовых воздействий в реализационные тестовые воздействия и обратное преобразование реализационных наблюдений в спецификационные наблюдения. Программа, осуществляющая эти преобразования, называется медиатором. По сути, это означает, что реализация может быть задана в другой семантике взаимодействия, и медиатор осуществляет преобразование семантик.

В простейшем случае различие семантик только в разных способах представления одних и тех же действий в реализации и спецификации. Достаточно взаимно-однозначного преобразования алфавитов реализации и спецификации. В общем случае такого преобразования недостаточно, поскольку спецификация, как правило, определяется на более высоком уровне абстракции. Медиатор может оказаться довольно сложной программой, осуществляющей медиативные функции между реализацией в одной семантике и тестом, генерируемым по спецификации в другой семантике.

Кроме тестовых воздействий и наблюдений, медиатор может преобразовывать также и состояния: из реализационного в спецификационное. Это означает, что на множестве состояний реализации предполагается заданным отношение эквивалентности, и состояние спецификации соответствует классу эквивалентных состояний реализации. Это соответствие является не предусловием тестирования, а частью проверяемого условия.

Предложена формализация устройства теста и медиатора при тестировании с преобразованием семантик. Соответствующим образом модифицированы гипотеза о безопасности и конформность ***saco***. Дополнительная модификация предложена для медиатора, осуществляющего преобразование состояний.

Для практического использования модифицирован алгоритм конечного полного тестирования с открытым состоянием, в том числе и для случая преобразования состояний в медиаторе.

Не до конца исследованы проблемы пополнения и монотонного преобразования для этого случая.