Бурдонов И.Б., Косачев А.С.

ТЕСТИРОВАНИЕ БЕЗОПАСНОЙ СИМУЛЯЦИИ

1. ВВЕДЕНИЕ

Безопасная симуляция введена в нашей статье «Симуляция систем с отказами и разрушением» [1]. Мы будем использовать введенные там основные понятия и обозначения без повторения их определения.

Тестирование – это проверка соответствия тестируемой системы заданным требованиям в процессе эксперимента, когда тест подменяет собой окружение системы. Симуляция, как любая конформность, определяется как соответствие двух формальных моделей: реализации (модель системы) и спецификации (модель требований). Спецификация считается заданной. Если реализация (модель системы) известна, можно применять аналитические методы верификации. Однако даже в этом случае соответствие верифицируемой модели (реализации) и «реальной» системы остается за скобками самой верификации. Такое соответствие может обеспечиваться двумя способами. 1) Модель системы (реализация) предшествует созданию самой системы и, если применяются методы автоматической генерации программы по модели, то гарантируется требуемое соответствие, а для верифицированной модели системы – также конформность: соответствие системы требованиям. 2) Модель системы получается также автоматически из текста программы специальными методами анализа программ.

При тестировании, однако, реальная система неизвестна, но *предполагается*, что она имеет модель (данного класса, в статье – LTS), которая при тестировании ее ведет себя так же, как реальная система. В литературе это называется *тестовой гипотезой* [2]. Тестовая гипотеза не проверяется при тестировании, целью которого является проверка соответствия реализации (как модели системы) и спецификации. Если тестовая гипотеза верна, то поведение реальной системы при тестировании будет таким же, как и ее модели (реализации). Поэтому реальная система удовлетворяет требованиям, формализуемым в спецификации, тогда и только тогда, когда ее модель (реализация) конформна спецификации.

Ставится задача генерации по спецификации *полных тестов*, которые однозначно определяли бы конформность или неконформность любой реализации (быть может, из некоторого класса). В разделе 3 утверждается, что

для некоторых спецификаций не существует полного теста на классе всех реализаций, которые можно безопасно тестировать для проверки симуляционной конформности заданной спецификации. Это существенно отличается от трассовой конформности, для которой полный тест всегда существует. Сформулированы ограничения на спецификацию, достаточные для существования полного теста. В следующем разделе 4 рассматривается практическое тестирование, которое требует конечности полного теста. Сформулированы более сильные ограничения на спецификацию, а также семантику взаимодействия и реализацию (в том числе ограничение на ее недетерминизм), которые достаточны не только для существования, но и для конечности полного теста. Приводится алгоритм работы такого теста.

2. ПОЛНОТА ТЕСТИРОВАНИЯ

Тестирование – это проверка конформности в процессе эксперимента. Для симуляции требуется операция опроса текущего состояния реализации, применяемая в начале тестирования и после каждого наблюдения (тестирование с открытым состоянием).

Тест – это инструкция, каждый пункт которой описывает либо требуемый рестарт системы[1], либо тестовое воздействие (кнопку) и в зависимости от полученных наблюдения и постсостояния – следующий пункт или вердикт (***pass*** или ***fail***). Реализация *проходит* тест, если ее тестирование всегда (при любом проявлении недетерминизма) не заканчивается с вердиктом ***fail***. Тест *значимый*, если каждая конформная реализация его проходит; *исчерпывающий*, если каждая неконформная реализация его не проходит; *полный*, если он значимый и исчерпывающий.

Для полноты тестирования симуляции ***ss*** нужно для каждого достижимого при безопасном тестировании состояния `i` реализации и каждой `H`-безопасной в нем кнопки `P` верифицировать каждое имеющееся в реализации наблюдение `u`$\in$`P`$\cup${`P`} и постсостояние `i`$`$. Предполагается, что любую пару (`u`,`i`$`$) можно получить за конечное число нажатий кнопки `P` в состоянии `i`. Также предполагается, что любое состояние $i_0` \in I_0 = (i_0$ ***after*** $\langle\rangle)$ можно получить за конечное число рестартов. Это называется гипотезой о *глобальном тестировании*.

[1] Тест с рестартом эквивалентен набору тестов, который обычно рассматривается.

3. ТЕОРЕТИЧЕСКОЕ ТЕСТИРОВАНИЕ

Если $s_0=\langle\gamma\rangle\Rightarrow$, то все реализации конформны, и тестирование не требуется. Если $s_0=\langle\gamma\rangle\not\Rightarrow$, определим минимальное множество N *неконформных пар* состояний (i,s), порождаемое следующими правилами вывода: $\forall (i,s)\in H\ \forall P\ H\text{-}\mathit{safe}\ i\ \forall u\in P\cup\{P\}$

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

Конформность $\mathbf{I}\ \mathit{ss}\ \mathbf{S}$ эквивалентна условию $(i_0,s_0)\notin N$. Правила вывода определяют граф вывода, вершины которого – пары $(i,s)\in N$, полученные применением 1-го (1-вершина) или 2-го (2-вершина) правила вывода. Помеченная дуга $(i,s)\text{—}(u,i`)\rightarrow(i`,s`)$ соответствует правилу вывода 2 для $s`\in(s\ \mathit{after}\ \langle u\rangle)$.

В графе вывода существует дерево маршрутов (быть может, не единственное), которое назовем деревом вывода. Каждый маршрут дерева начинается в (i_0,s_0), корень дерева – пустой маршрут, а листья – маршруты, заканчивающиеся в 1-вершинах. Маршрут дерева, заканчивающийся в 2-вершине, продолжается в дереве теми и только теми дугами, которые помечены одной меткой $(u,i`)$, что соответствует 2-му правилу вывода.

Если $(i_0,s_0)\in N$, но все деревья вывода бесконечны, то никакой тест не сможет за конечное время определить неконформность. Поэтому на классе всех H-безопасных реализаций могут существовать только значимые тесты. Один из таких тестов полный на подклассе реализаций, в которых либо не существует дерева вывода (реализация конформна), либо существует конечное дерево вывода (реализация неконформна). Кроме глобального тестирования, требуется перечислимость множества $S_0=(s_0\ \mathit{after}\ \langle\rangle)$, множества безопасных в каждом безопасно достижимом состоянии s кнопок $P(s)=\{P\in\mathbf{R}\cup\mathbf{Q}\mid P\ \mathit{safe}\ s\}$, и множества постсостояний $S(s,u)=(s\ \mathit{after}\ \langle u\rangle)$ для каждого безопасного в s наблюдения u.

Дерево вывода конечно тогда и только тогда, когда оно имеет конечное ветвление, что эквивалентно конечности каждого множества $s\ \mathit{after}\ \langle u\rangle$. Для этого достаточно, чтобы в каждом безопасно достижимом состоянии s спецификации были конечны 1) число переходов по каждому безопасному действию, и 2) множество $s\ \mathit{after}\ \langle\rangle$.

3. ПРАКТИЧЕСКОЕ ТЕСТИРОВАНИЕ

На практике требуется, чтобы тест заканчивался за конечное время. Без каких-либо ограничений полный тест обнаруживает ошибку за конечное время, но при отсутствии ошибок может выполняться бесконечно долго. Достаточно следующих ограничений. 1) Число кнопок *конечно* и каждая кнопка *разрешима* относительно алфавита действий. 2) Спецификация *конечна*: конечно число состояний и переходов. 3) «Часть» **I`** реализации **I**, «проходимая» при безопасном тестировании, *конечна* и *t-недетерминирована*: все возможные пары (наблюдение, постсостояние) могут быть получены не просто за конечное число нажатий данной кнопки в данном состоянии, как при глобальном тестировании, а не более, чем за t нажатий. При t=1 реализация детерминирована.

Условия конечности позволяют сначала исследовать реализацию **I**, то есть построить **I`**, а потом провести верификацию. Переход i—u→i` добавляется в **I`** тогда, когда после опроса состояния i нажимается кнопка P, получается наблюдение u и снова опрашивается постсостояние, которым оказывается состояние i`. Эту кнопку P будем называть *управляющей* кнопкой и обозначать P(i—u→i`). Заметим, что мы добавляем не только переходы по внешним действиям, но также виртуальные переходы по наблюдаемым отказам. Если P – это **R**-кнопка и u=P, то добавленный переход i—P→i` означает, что в реализации либо есть виртуальная петля по отказу i—P→i, если i=i`, либо в реализации есть τ-маршрут из i в i` и виртуальная петля по отказу i`—P→i`.

Для каждого пройденного состояния i хранится множество H(i) соответствующих ему по H спецификационных состояний, которое будет постепенно расти, и множество безопасных кнопок P(i)=∪{P(s)|s∈H(i)}. Для каждой кнопки P∈P(i) хранится счетчик C(i,P) числа нажатий кнопки P в состоянии i. Кнопка P∈P(i) *полна в состоянии* i, если c(P,i)=t. Состояние i *полно*, если все кнопки из P(i) полны.

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

Сначала проверяем полноту текущего состояния i из **I`** (рис. 1). Если состояние неполное, то есть неполная кнопка P∈P(i). Тогда нажимаем

кнопку P и получаем переход i—⟨u⟩→i`, постсостояние i` становится новым текущим состоянием. Увеличиваем счетчик c(P,i):=c(P,i)+1. Если переход i—⟨u⟩→i` новый, добавляем его в **I`**, кнопку P запоминаем как управляющую P(i—u→i`), и обновляем H(i`):=H(i`)∪S(s,u) для каждого s∈H(i) при условии P ***safe*** s.

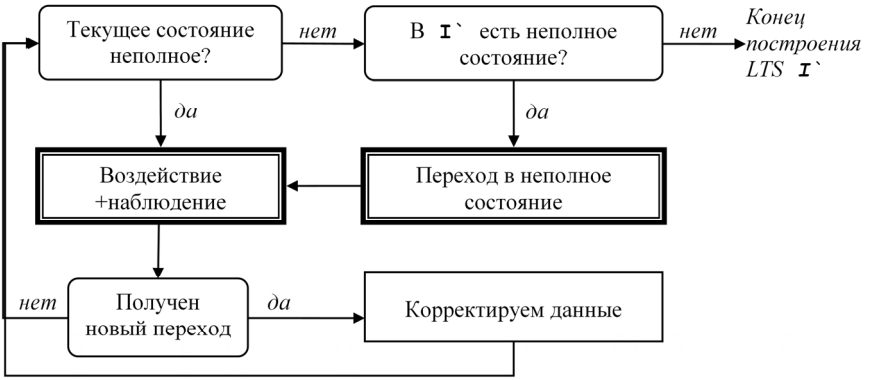

Рис. 1. Схема алгоритма исследования реализации

Когда новое состояние s добавляется в H(i), корректируем P(i):=P(i)∪P(s), и для каждого полученного ранее перехода i—⟨u⟩→i` обновляем H(i`):=H(i`)∪S(s,u) при условии P(i—⟨u⟩→i`) ***safe*** s, отмечая вновь добавленные состояния. Эта рекурсивная процедура повторяется пока возможно. Условия конечности гарантируют, что процедура закончится за конечное число шагов.

Если текущее состояние i полное, то переходим в любое неполное состояние (если таких состояний нет, построение **I`** заканчивается). Для этого выбираем в LTS **I`** лес деревьев, покрывающих все состояния и ориентированных к своим корням, которыми являются все неполные состояния. С каждым переходом i—u→i` этого леса свяжем управляющую кнопку A(i)=P(i—u→i`). Будем двигаться, нажимая в каждом текущем состоянии i кнопку A(i). Из-за недетерминизма мы можем оказаться не в состоянии i`, а в другом состоянии i``, где будем нажимать кнопку A(i``). Переход в неполное состояние гарантируется t-недетерминизмом реализации.

Исследование реализации заканчивается за конечное время. Число тестовых воздействий равно $O(bt^n)$ для $t>1$ и $O(bn^2)$ для $t=1$, а объем вычислений равен $O(bnt^n)+O(bnm)+O(mk)$ для $t>1$, для $t=1$ первое слагаемое заменяется на $O(bn^3)$, где b – число кнопок, n – число состояний реализации, $m=O(bnt)$ – число переходов реализации, k – число состояний спецификации.

Верификация симуляции после построения LTS **I`** пытается построить конформное соответствие $R_2$, выдавая вердикт ***pass***, если такое соответствие существует и построено, или вердикт ***fail***, если такого соответствия быть не может. Сначала строится двудольный граф. Вершины 1-го типа – пары $(i,s)$, где $i$ – состояние **I`**, а $s \in H(i)$ состояние **S**. Вершины 2-го типа – пары $(i—u→i`,s)$, где $i—u→i`$ – переход **I`**, а $s \in H(i)$. В каждую вершину 2-го типа входит одна дуга 1-го типа $(i,s) \to (i—u→i`,s)$. Дуга 2-го типа $(i—u→i`,s) \to (i`,s`)$ проводится, если $s` \in S(s,u)$. Одновременно составляется список терминальных вершин 2-го типа.

После построения двудольного графа каждая терминальная вершина v_2 2-го типа удаляется вместе с входящей в нее дугой $v_1 \to v_2$, начальной вершиной v_1 этой дуги, и каждой входящей в нее дугой $v_1` \to v_1$. Одновременно для $v_1=(i,s)$ состояние $s$ удаляется из множества $H(i)$. Эти операции повторяются, пока не будет удалена вершина 1-го типа $(i_0`,s_0)$, где $i_0` \in I_0$, или пока не будут удалены все терминальные вершины 2-го типа. В первом случае алгоритм заканчивается с вердиктом ***fail***, поскольку $(i_0`,s_0) \in N$ влечет $(i_0,s_0) \in N$, а во втором случае – с вердиктом ***pass***. Во втором случае строится соответствие $R_2=\{(i,s) \mid i \in V_{I`} \ \& \ s \in H(i)\}$. Если при построении двудольного графа каждое $H(i)$ заменить на множество всех состояний спецификации, то алгоритм будет строить наибольшее конформное соответствие R_1.

Объем вычислений при верификации равен $O(mk^2)$.

СПИСОК ЛИТЕРАТУРЫ

1. **Бурдонов И.Б., Косачев А.С.** Симуляция систем с отказами и разрушением. – настоящий сборник.
2. **Bernot G.** Testing against formal specifications: A theoretical view. In S. Abramsky and T.S.E. Maibaum, editors, TAPSOFT'91, Volume 2, pp. 99-119. Lecture Notes in Computer Science 494, Springer-Verlag, 1991.