Модел. и анализ информ. систем. Т. **19**, № **6** (2012) 79–91

УДК 004.415

Подход к конкретизации тестовых сценариев в рамках технологии автоматизации тестирования промышленных программных проектов

Колчин А.В.*, Летичевский А.А.*, Песчаненко В.С.*, Дробинцев П.Д.**, Котляров В.П.**

** Институт кибернетики им. В.М. Глушкова НАН Украины,*
*** Санкт-Петербургский государственный политехнический университет*

*e-mail: * let@cyfra.net, ** vpk@spbstu.ru*

получена 22 июля 2012

Ключевые слова: символьная верификация, автоматизация тестирования, конкретизация тестовых сценариев, предикатный трансформер

Предложен подход к созданию эффективной технологии автоматизации тестирования промышленных программных проектов, который использует формальную модель системы, выполняет автоматически символьную верификацию, генерацию и конкретизацию символьных трасс, генерацию тестовых наборов по конкретизированным трассам, а также включает средства анализа результатов исполнения тестов, позволяя автоматизировать полный цикл тестирования. Особый акцент сделан на изложении алгоритма конкретизации и настройки тестовых сценариев.

Введение

Ключевыми проблемами современной технологии тестирования является обеспечение качества программного продукта (ПП) и минимизация трудозатрат на производство ПП с фиксированным ограничением на плотность дефектов. Для обеспечения заданного уровня качества применяются статические (статическая верификация) и динамические (тестирование) методы контроля качества ПП, начиная с самых ранних этапов разработки. В данной работе рассматривается подход к созданию эффективной технологии автоматизации тестирования больших и средних промышленных программных проектов, обеспечивающий минимизацию трудозатрат за счет практически «бесшовной» автоматизации всех фаз разработки.

Работа поддержана грантом РФФИ 11-07-90412-Укр_ф_а (Россия) и научно-исследовательской работой № Ф40\51-2011(Украина).

Рис. 1. Свойства инструментария автоматизации тестирования

1. Текущее состояние инструментальных средств автоматизации тестирования

В промышленных инструментальных средствах поддержки технологий автоматизации тестирования для снижения трудоемкости тестирования важна поддержка следующих свойств (рис. 1):

1. Автоматический прогон тестового набора в соответствии с планом тестирования, локализация и анализ ошибок с помощью логирования.

2. Автоматическая генерация оптимизированного кода тестовых наборов (test-suite) на целевую платформу по созданным вручную поведенческим сценариям (трассам).

3. Использование созданной вручную поведенческой модели для автоматической генерации сценариев, удовлетворяющих заданному критерию покрытия.

4. Автоматическая генерация множества поведений, в совокупности образующих поведенческую модель ПП, на основе созданных вручную формальных моделей отдельных исходных требований (например, в виде троек Хоара [1], базовых протоколов Летичевского [2] и т.п.).

Среди существующего инструментария тестирования наиболее распространены системы, поддерживающие первое свойство, например, [3–4]; в качестве примеров систем поддерживающих второе свойство можно назвать [5–7]; третье свойство поддерживается в системах, использующих мощные CASE-технологии (например, [8]), где трассы генерируются на основе поведенческих моделей, созданных вручную. Настоящая работа посвящена развитию технологии тестирования для инструментария VRS/TAT [9], поддерживающего все четыре свойства и обеспечивающего автоматическую генерацию тестовых сценариев по спецификациям требований, формализованных в нотации базовых протоколов [2].

2. Основные проблемы существующих подходов к автоматизации тестирования

В инструментальных системах, где в основном решены проблемы автоматизации формализации требований, создания поведенческих моделей, верификации сгенерированных по модели символьных сценариев (трасс) и анализа степени покрытия исходных требований [19–22], при генерации поведенческих сценариев промышленных систем встает проблема комбинаторного взрыва вариантов поведения, подлежащих тестированию [23–25]. Например, количество состояний системы растет экспоненциально с числом процессов.

Для сокращения пространства поведений эффективно применяются методы символьной верификации. Символьным состоянием является формула, описывающая ограничения над параметрами трассы. В символьной трассе можно определить диапазон (Range) возможных значений параметров. Каждая символьная трасса представляет пучок конкретных трасс с эквивалентным поведением (т.е. с одинаковой последовательностью событий). А так как критерии покрытия определяются поведением модели, то это означает, что для обеспечения необходимого покрытия достаточно выбрать некоторых представителей из каждого класса поведенческой эквивалентности, заменив таким образом (без потери покрытия) потребность в переборе вариантов поведения для всех конкретных значений параметров.

Набор символьных трасс, удовлетворяющий критерию покрытия, после оптимизации (удаления трасс с повторным покрытием некоторых требований) существенно сокращает набор тестовых сценариев, но, к сожалению, получаемые символьные трассы в таком виде не годятся для генерации тестов, ориентированных на целевую платформу. Для генерации кода исполнимых тестов требуются трассы с конкретными значениями параметров, что ставит задачу создания инструмента автоматической подстановки конкретных значений параметров (конкретизации) символьных трасс.

Следует отметить, что в современных промышленных проектах количество тестов исчисляется тысячами, а зависимость между значениями параметров нетривиальна. Осуществлять вручную выбор и подстановку взаимосогласованных конкретных значений, ориентируясь по указанным в трассе диапазонам, практически невозможно. Поэтому процесс конкретизации должен быть полностью автоматическим.

Кроме того, реальная практика тестирования свидетельствует об уменьшении трудоемкости поиска дефектов при направленном выборе значений из разрешенного диапазона по сравнению со случайном выбором. При подстановках необходимо следовать плану тестирования, подготовленному заранее тестировщиком. Подобные планы должны быть гибкими, основная их часть должна генерироваться на основе стандартных шаблонов или переиспользовать отредактированные пользователем планы.

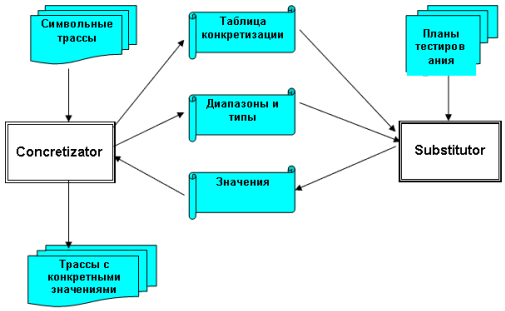

Рис. 2. Инструмент конкретизации

Var name	Signal name	Type	User option	Range	Value
Speed_Value	Current_Speed	I	L	-2147483648 <=Speed_Value &Speed_Value <=2	-2147483648
Speed_Value	Current_Speed	I	L	4<=Speed_Value&Speed_Value<=2147483647	4
Driving_Mode	MCS_Position	TRAIN_OPERATION_MODE	L	_AUTO,_CS,_RM_REV	_AUTO

Рис. 3. Таблица конкретизации

3. Метод автоматической конкретизации символьных трасс в системе VRS

Далее описан метод подстановки конкретных значений в символьные трассы и его реализация в системе верификации VRS (Verification Requirements System). VRS имеет символьный трассовый генератор STG [10, 12], который исследует пространство поведения формальных моделей, используя экзистенциальную абстракцию, и строит трассы — линейные последовательности событий модели. Так же в трассу записываются состояния модели, которые представлены формулами логики предикатов первого порядка. Инструментом подстановки конкретных значений в символьные трассы служит Trace Concretization Tool. Его структура представлена на рис. 2. Инструмент состоит из двух модулей Concretizator и Substitutor, взаимодействующих в режиме диалога.

Concretizator для каждой символьной трассы создает таблицу конкретизации (рис. 3), в которой заполняет столбцы имен параметров и сигналов, типов и разрешенных диапазонов значений, а затем при прямом обходе трассы обращается к модулю Substitutor за конкретным значением очередного параметра. Substitutor на основе команд и данных плана конкретизации трассы вычисляет по допустимому диапазону конкретное значение и возвращает его в Concretizator. План автоматической конкретизации тестового набора заносится в колонку User option и формируется в терминах команд управления: R, M, L, O, C.

Команда R управляет подстановкой значения, равного правой границе диапазона соответствующего параметра, команда L управляет подстановкой значения, рав-

ного левой границе, команда M предусматривает вычисление и подстановку среднего значения диапазона, команда O предусматривает подстановку значения, находящегося за границами диапазона, а команда C управляет подстановкой конкретного значения, определенного пользователем.

Следует отметить возможность вычисления Concretizator не непрерывного диапазона допустимых значений параметров, в этом случае в плане предусматривается анализ каждого из подобных поддиапазонов в отдельности.

Таким образом, средства управления конкретизацией достаточно гибки для тестирования любых режимов работы ПП.

4. Краткое описание алгоритма конкретизации

Так как исходная символьная трасса получена применением экзистенциального моделирования, перед ее конкретизацией необходимо вычислить фактические ограничения ее параметров. Это обеспечивается применением обратного предикатного трансформера [13]. Далее процесс конкретизации происходит итеративно, на каждом шаге конкретизируется значение очередного параметра, подстановка которого обеспечивается прямым предикатным трансформером [12]. Процесс конкретизации заканчивается, когда подставлено конкретное значение последнего параметра трассы.

Перед описанием алгоритма необходимо привести определения переходов модели и трасс. Переходом формальной модели в системе VRS служит «базовый протокол» [10], представляющий атомарный параметризированный переход из одного состояния модели в другое. Базовый протокол $B(x)$ представляет собой выражение вида: $\forall x(\alpha(x) \rightarrow < P(x) > \beta(x))$, где x — список параметров протокола, $\alpha(x), \beta(x)$ — формулы базового логического языка, которые называются предусловием и постусловием соответственно, $P(x)$ — процесс базового протокола (в рассматриваемом случае — последовательность параметризированных сигналов в формате MSC). Параметрами трассы будем называть параметры ее сигналов. В качестве базового языка используется язык многосортного исчисления предикатов первого порядка. Формула базового языка может содержать переменные и константы, массивы элементов простых типов, функциональные типы, списки элементов простых типов. В роли переменных, меняющих свои значения в процессе функционирования системы, выступают атрибуты и атрибутные выражения. Атрибутными выражениями являются операторы доступа к элементу массива по индексам, функции доступа к спискам, выражения атрибутов функциональных типов.

Трассой называется последовательность вида $S_0 \xrightarrow{B_0(x_0)} S_1 \xrightarrow{B_1(x_1)} ...S_n$, в которой S — состояния, B — базовые протоколы, x — списки их параметров.

Алгоритм конкретизации.

Шаг 1. Восстановление исходной символьной трассы.

Шаг 2. Получение области допустимых значений параметров базового протокола.

Шаг 3. Интерактивная конкретизация параметров трассы.

Шаг 4. Запись конкретизированной трассы.

Ниже представлено подробное описание шагов алгоритма.

4.1. Восстановление исходной символьной трассы

Процесс восстановления трассы начинается с загрузки исходной трассы и начального состояния. Результатом загрузки трассы становится список двоек $L =< B, R_B >$, где B — базовый протокол, R_B — ограничения на параметры протокола B, которые возникли в процессе применения этого протокола. Для восстановления трассы загружается начальное состояние среды и далее последовательно применяются протоколы прямым предикатным трансформером, двигаясь из начала в конец списка L. Для вычисления фактических ограничений необходимо выполнить применение базовых протоколов из списка L в обратном порядке, применяя обратный предикатный трансформер [13]. Прямой предикатный трансформер позволяет осуществить применение (выполнение) базового протокола $B(x) = \forall x(\alpha(x) \rightarrow < P(x) > \beta(x))$ к состоянию модели E и вычислить следующее состояние $E' : E' = pt(E \wedge \alpha(x), \beta(x))$. Обратный предикатный трансформер на основании состояния E' и перехода $B(x)$ позволяет вычислить предшествующее ему состояние $E : E = pt^{-1}(E' \wedge \alpha(x), \beta(x))$. При применении каждого протокола B учитываются соответствующие ограничения R_B. В результате выполнения первого шага алгоритма будут вычислены фактические ограничения на все состояния трассы, включая начальное состояние.

4.2. Получение области допустимых значений параметров базового протокола

Отметим тот факт, что обратный предикатный трансформер позволяет корректно уточнить (сузить) предшествующее состояние, т.к. изначально оно было построено прямым предикатным трансформером с экзистенциальной абстракцией. Такой подход гарантирует корректность трассы для любых значений из уточненного состояния. Рассмотрим пример:

$E : x > 0$

$B_1 : (x < 5) \rightarrow < P_1 > 1$

$B_2 : \forall(a, b)((a = b) \rightarrow < P_2(a, b) > (x := a) \wedge (y := b))$

$B_3 : (x > 6) \rightarrow < P_3 > 1$

Восстановим трассу B_1, B_2, B_3. Начнем восстановление трассы прямым предикатным трансформером:

- $B_1(x > 0) = 0 < x \wedge x < 5 = E_1$;
- $B_2(0 < x \wedge x < 5) = \exists(x', y', a, b)(0 < x' \wedge x' < 5 \wedge a = b \wedge x = a \wedge x = b) =$ $= (x = y) = E_2$;
- $B_3(x = y) = (x > 6 \wedge x = y) = E_3$ [12].

Продолжим восстановление трассы обратным предикатным трансформером:

- $B_3(E_3) = B_3(x > 6 \wedge x = y) = (x > 6 \wedge x = y) = E_2'$;
- $B_2(E_2') = B_2(x > 6 \wedge x = y) = \exists(x', y', a, b)(x' > 6 \wedge x' = y' \wedge a = b \wedge x' = a \wedge y' = b) =$ $= \exists(a, b)(a > 6 \wedge a = b) = 1 = E_1'$;
- $B_1(E_1') = B_1(1) = (x < 5) = E'$.

Построим область допустимых значений для параметра b протокола B_2. При прямом моделировании параметр не имеет ограничений, однако, например, значение 0 не корректно для данной трассы — предусловие перехода B_3 окажется невыполнимым. Для вычисления корректного диапазона мы используем обратный предикатный трансформер. В данном случае достаточно взять конъюнкцию $R_{B_2} \wedge R'_{B_2}$, т.е. $(a > 6 \wedge a = b)$, откуда следует $b > 6$. Для более общего случая область допустимых значений следует искать в конъюнкции $E_B \wedge E'_B \wedge R_B \wedge R'_B$, где E_B (E'_B) — состояние среды до (соответственно после) применения протокола B прямым (обратным) предикатным трансформером, R_B (R'_B) — ограничения на параметры протокола полученные после применения прямого (обратного) предикатного трансформера.

Для получения области допустимых значений по заданной формуле используется следующая последовательность действий:

1. Элиминируются все функциональные атрибутные выражения, включая массивы методом Шостака [12], для того чтобы правильно получить ограничения на атрибутные выражения в параметрах функциональных атрибутных выражений.

2. Строится дизъюнктивная нормальная форма, при этом используется специальный теоретико-множественный алгоритм для перечислимого типа.

3. Каждый конъюнкт построенной дизъюнкции разделяется по типам подформулы: булева, числовая, перечислимая, произвольная символьная.

4. Строится соответствующая область допустимых значений для конъюнкции всех подформул, которые зависят от данного атрибута.

Поскольку булевы атрибутные выражения рассматриваются как перечислимые с двумя предопределенными константами, то алгоритм из пункта 2 для атрибутных выражений обоих типов даст области в виде множеств $a = \{a_1, ..., a_n\}$. Более сложной является задача нахождения области допустимых значений для атрибутов, представленных произвольными термами — в общем случае нельзя выделить область допустимых значений, не зависящую от других атрибутных выражений. Эта область состоит из всех областей атрибутных выражений, от которых зависит данных символьный терм.

При вычислении области допустимых значений для числовых формул с целочисленными и вещественными атрибутными выражениями используется алгоритм построения выпуклой оболочки в n-мерном векторном пространстве, основанный на методе трапецоидов [14].

4.3. Интерактивная конкретизация параметров трассы

Конкретизация параметров трассы использует области допустимых значений каждого параметра, вычисленного на предыдущем шаге. Далее выполняется итеративный процесс подстановки конкретных значений:

1. Сообщить модулю Substitutor диапазон допустимых значений очередного параметра.

2. Получить конкретное значение от модуля Substitutor (значение вычисляется автоматически или задается вручную).

3. Проверить, входит ли заданное конкретное значение в область допустимых значений, в противном случае вернуться в пункт 1 для получения нового значения.

4. Конъюнктивно добавить к ограничениям R_B на параметры текущего протокола B равенство вида: $<$*атрибутное выражение*$> = <$*полученное значение*$>$.

5. После того как все конкретные значения параметров очередного протокола B заданы, следует обновить состояние трассы посредством выполнения прямого предикатного трансформера для протокола B с учетом новых значений параметров протокола:

   $E' = pt(E \wedge R_B^* \wedge \alpha_B, \beta_B)$,

   $B = \forall x(\alpha_B(x) \rightarrow < P(x) > \beta_B(x))$, $R_B^* = (a_1 = v_1) \wedge ... \wedge R_B$, $\{B, R_B\} \in L$,

   $a_1, ... \in x$, $v_1, ...$ — новые значения каждого из параметров протокола B.

4.4. Запись конкретизированной трассы

После того, как последний параметр трассы получил свое конкретное значение, трасса может быть записана на диск. Теперь в трассе все ее сигналы имеют конкретные значения всех параметров. Построенная описанным методом трасса корректна, т.е. все значения ее параметров согласованы между собой и не противоречат исходной символьной трассе. Такая трасса служит входом для очередного этапа технологии — построения теста.

5. Простой пример конкретизации

В требованиях рассматриваемого примера описана задача сложения двух положительных целых, не превышающих 10. Спецификация состоит из 4 базовых протоколов (рис. 4):

- Protocol1 осуществляет получение на вход значений двух переменных “a” и “b” для сложения.

- Protocol2 выполняет анализ значений переменных на предмет вхождения в диапазон (0;10) и вычисляет сумму, если значения лежат внутри диапазона.

- Protocol2_1 выполняет анализ значений переменных на предмет выхода из диапазона (0;10) и запись сообщения об ошибке res=999.

- Protocol3 выполняет вывод результатов вычисления.

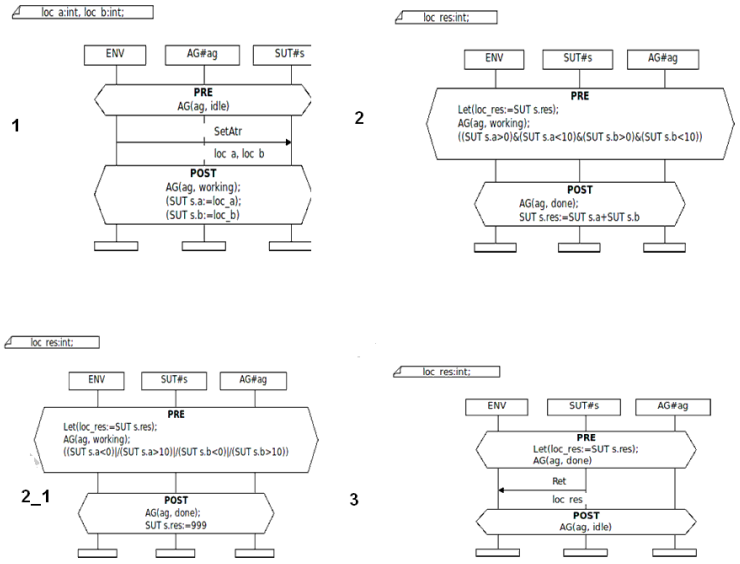

Рис. 4. Базовые протоколы простого примера

При проведении верификации были проверены 2 варианта поведения системы, в первом случае были использованы протоколы, в которых переменные входят в диапазон, и во втором случае, когда не входят.

На рис. 5 представлены полученные в результате верификации 2 символьные трассы. В данных трассах видно, что в сигналах установки значений переменных и в сигнале возврата результата вычисления находятся символьные параметры “loc_a”, “loc_b”, “loc_res”.

На рис. 6 представлены конкретизированные трассы, в которых подстановка осуществлялась по команде “M” (то есть из допустимого диапазона для переменной в каждом случае выбиралось среднее значение). В представленных трассах все символьные параметры заменены на конкретные значения, и, следовательно, эти трассы могут быть использованы для генерации исполнимого кода тестов.

6. Результаты апробаций

Рассмотренный подход был применен для подготовки тестов для 5 промышленных проектов. Символьные трассы, описывающие различные варианты поведения исследуемых проектов, содержали от десятков до нескольких сотен базовых протоколов. Очевидно, что для проведения тестирования необходимо в символьных трассах, применяемых для сокращения количества поведений системы, провести замену каждого символьного параметра на конкретное значение, что без использования средств автоматизации приводит к значительному замедлению процесса тестирования.

Например, применение рассмотренного подхода в небольшом проекте из 11 базовых протоколов с трассами, в которых конкретизировался только один параметр, позволило решить задачу автоматической конкретизации символьных трасс за 2

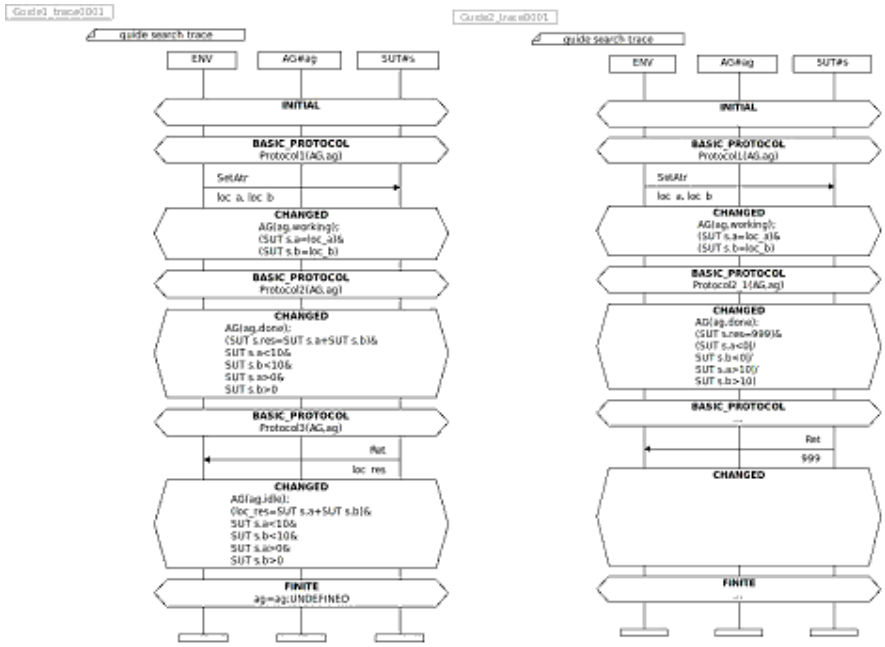

Рис. 5. Символьные трассы простого примера

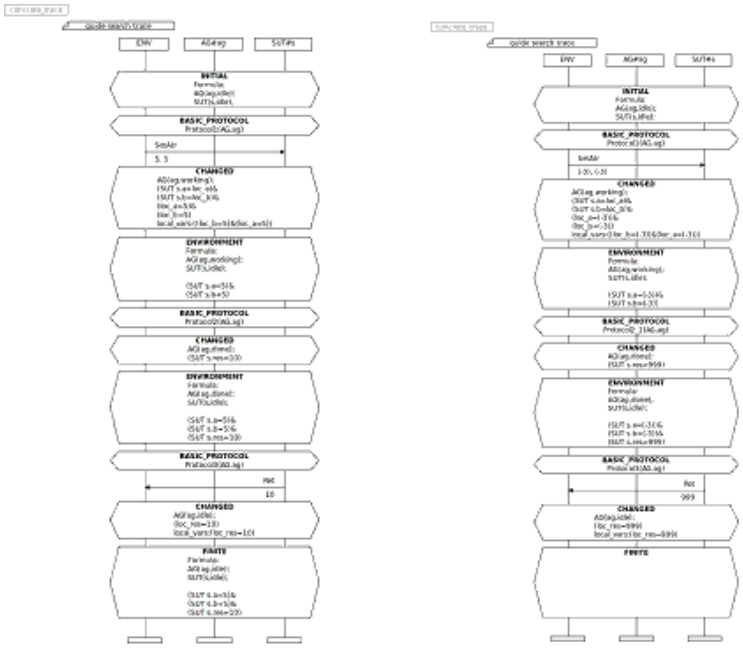

Рис. 6. Конкретизированные трассы

минуты. В проекте с 151 базовым протоколом и 4 параметрами конкретизация заняла менее получаса. Оценка времени ручной конкретизации для подобного проекта составляет 2–3 рабочих дня.

Понятно, что в проектах размером 10^3–10^4 базовых протоколов реализовать конкретизацию набора символьных трасс, удовлетворяющих заданному критерию тестирования, а значит, обеспечить требуемый уровень качества ПП, без инструментария автоматической конкретизации невозможно.

Заключение

Интеграция верификации и тестирования позволяет достичь требуемого уровня качества ПП за счет объединения результатов анализа верифицируемой модели с ограниченным числом экспериментальных результатов, полученных в процессе тестирования.

Для систем с типами данных, включающими большой диапазон значений, одна символьная трасса может покрывать множество конкретных трасс. Это свойство особенно эффективно используется при верификации моделей, использующих числовые типы данных, однако символьные трассы в исходном виде непригодны для создания по ним исполнимых тестов. Предложенный метод конкретизации трасс устраняет это ограничение, причем процесс конкретизации полностью автоматизирован. Более того, технология позволяет контролировать покрытие граничных значений параметров теста, что в результате дает возможность повысить качество создаваемого тестового набора.

Применение предложенной технологии в практике верификации и тестирования промышленных проектов обеспечило эффективную генерацию минимизированного набора тестовых сценариев, обеспечивающих необходимое покрытие исходных требований к разрабатываемым системам.

Список литературы

1. Hoare C.A.R. Communicating sequential processes, Prentice Hall, 1985.

2. Letichevsky J., Kapitonova A., Letichevsky Jr., Volkov V., Baranov S., Kotlyarov V., Weigert T. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Computer Networks. 2005. 47. P. 662–675.

3. Abbot framework for automated testing of Java GUI components and programs/Abbot., 2011, `http://abbot.sourceforge.net/doc/overview.shtml`.

4. Jameleon — An Automated Testing Tool — Overview / Jameleon, 2011, `http://jameleon.sourceforge.net/index.html`

5. Silk Software Test Management, Test Automation and Performance Testing / Borland, 2011, `http://www.borland.com/us/products/silkline/index.aspx`

6. Open Source Software Engineering Tools / Tigris, 2011, `http://maxq.tigris.org`.

7. Software Testing Tools and other Products / Parasoft, 2011, http://www.parasoft.com/jsp/products.jsp.

8. IBM Rational software/ IBM, 2011, http://www-01.ibm.com/software/rational/?pgel=ibmhzn&cm_re=masthead-_-products-_-sw-rational.

9. Baranov S.N., Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A. Implementation of an integrated verification and testing technology in telecommunication project. Proceedings // IEEE Russia Northwest Section. 110 Anniversary of Radio Invention conference. S.Petyersburg, 2005. 11 p.

10. Летичевский А.А., Капитонова Ю.В., Волков В.А., Летичевский А.А. (мл.), Баранов С.Н., Котляров В.П. Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ. 2005. №4. С. 256–268.

11. Потиенко С.В. Методы прямого и обратного символьного моделирования систем, заданных базовыми протоколами // Проблемы программирования. 2008. №4. С. 39–45.

12. Летичевский А.А., Годлевский А.Б., Летичевский О.А., Потиенко С.В., Песчаненко В.С. Свойства предикатного трансформера системы VRS // Кибернетика и системный анализ. 2010. №4. С. 3–16.

13. Годлевский А.Б., Потиенко С.В. Обратная трансформация формул в символьном моделировании: от результата к исходной формуле // Проблемы программирования. 2010. № 2–3. С. 363–368.

14. Львов М.С. Алгебраический подход к задаче решения систем линейных неравенств // Кибернетика и системный анализ. 2010. № 2. С. 175–188.

15. Goguen J., Meseguer J. Ordered-Sorted Algebra I: Partial and Overloaded Operations. Errors and Inheritance. SRI International, Computer Science Lab., 1987.

16. Моцкин Т.С., Райфа Х., Томпсон Дж.Л., Тролл Р.М. «Метод двойного описания», Матричные игры. М.:Физматгиз, 1961. С. 81–109.

17. Zeidler G.L. Lectures on convex polytopes. Springer Verlag, New York, 1994.

18. Черников С.Н. Линейные неравенства. М.: Наука, 1968. 490 с.

19. Баранов С.Н., Котляров В.П. Автоматизация формализации требований для получения сценариев тестирования программ, Перспективы систем информатики: труды семинара «Наукоемкое программирование», 15–19 июня 2009. Новосибирск, Академгородок, 2009. С. 27–35.

20. Baranov S., Kotlyarov V., Letichevsky A., Drobintsev P. The technology of Automation Verification and Testing in Industrial Projects // Proc. of St.Petersburg IEEE Chapter, International Conference, May 18-21. St.Petersburg, 2005. P. 81–86.

21. Никифоров И., Петров А., Юсупов Ю. Генерация формальной модели системы по требованиям, заданным в нотации USE CASE MAPS: Научно-технические ведомости СПбГПУ. Санкт-Петербург: Издательство Политехнического университета, 2010. № 4 (103). С. 191–195.

22. Baranov S., Kotlyarov V., Weigert Th. Verifiable Coverage Criteria for Automated Testing. SDL 2011: INTEGRATING SYSTEM AND SOFTWARE MODELING. Lecture Notes in Computer Science. 2012. Vol.7083/2012. P. 79–89.

23. Летичевский А.А., Колчин А.В. Генерация тестовых сценариев на основе формальной модели // Проблемы программирования. 2010. № 2–3. С. 209–215.

24. Utting M., Legeard B. Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann. 2007. 456 p.

25. Бурдонов И., Косачев А., Пономаренко В., Шнитман В. Обзор подходов к верификации распределенных систем. М.: ИСП РАН, 2006. 61 с.

An Approach to Concretization of Test Scenarios in Industrial Automation Testing of Software Projects

Kolchin A.V., Letichevsky A.A., Peschanenko V.S., Drobintsev P.D., Kotlyarov V.P.

Keywords: symbolic verification, testing automatization, concretization of test scenarios, predicate transformer

In this paper we propose an approach to efficient automating test technology for industrial software projects, that uses a formal model of the system, automatically performs a symbolic verification, generation and concretization of the symbolic traces, the generation of test suites for concretized traces, and also includes tools for analysis of the testing results, allowing users to automate the full cycle of testing. Particular emphasis is placed on the presentation of the algorithm concretization and setting of test scenarios.

Сведения об авторах:

Колчин Александр Валентинович, Институт кибернетики им. В.М. Глушкова НАН Украины, науч. сотр., канд. физ.-мат. наук;

Летичевский Александр Адольфович, Институт кибернетики им. В.М. Глушкова НАН Украины, зав. отделом теории цифровых автоматов ИК, академик, д-р физ.-мат. наук;

Песчаненко Владимир Сергеевич, Институт кибернетики им. В.М. Глушкова НАН Украины, доцент, канд. физ.-мат. наук;

Дробинцев Павел Дмитриевич, Санкт-Петербургский государственный политехнический университет, доцент, канд. техн. наук;

Котляров Всеволод Павлович, Санкт-Петербургский государственный политехнический университет, профессор, канд. техн. наук.