*Модел. и анализ информ. систем.* Т. **18**, № **4** (2011) 118–130

УДК 004.4'22

# Формальная модель требований, используемая в процессе генерации кода приложения и кода тестов[1]

Баранов С.Н., Котляров В.П.

*Санкт-Петербургский государственный политехнический университет*

*e-mail: SNBaranov@gmail.com, vpk@spbstu.ru*

*получена 21 ноября 2011 года*

**Ключевые слова:** модель требований, поведенческие трассы, базовые протоколы, область определения модели

Рассматривается модель, используемая при ручной разработке спецификаций приложения, созданная на основе теории базовых протоколов А.А. Летичевского и поддерживающего теорию инструментария символьной верификации. Обсуждаются способы ограничения поведенческих характеристик модели при условии сохранения соответствия исходным требованиям. По успешно верифицированной модели генерируется код приложения и код тестов. Приводится описание методики применения разработанной модели.

## 1. Введение

Основной проблемой современной индустрии производства программного продукта является трудоемкость разработки качественного продукта, плотность остаточных дефектов в котором не превышает нескольких дефектов на миллион строк исходного кода. Получение продукта подобного качества обеспечивается технологией генерации кода по верифицированным и оттестированным формальным моделям.

Обязательными требованиями к подобным моделям являются:

- соответствие семантики требований и генерируемого кода,
- корректность, непротиворечивость и полнота формальной модели,
- надежность или соответствие генерируемого кода поведенческому графу верифицированной модели.

В настоящей работе применен подход, позволяющий формализовать исходные требования и на их основе строить корректную и непротиворечивую модель приложения. Решение проблемы основано на результатах работ школы А. Летичевского [1].

[1]Работа поддержана грантом РФФИ 11-07-90412_Укр_ф_а.

## 2. Формальная модель базовых протоколов

Основным элементом формальной модели является базовый протокол (БР). Базовый протокол [2] кодирует минимальный наблюдаемый шаг поведения системы. БР — это аналог тройки Хоара, содержащий предусловие, постусловие и процесс (наблюдаемое действие, последовательность действий). Пред- и постусловия — это логические формулы с неравенствами и арифметическими операциями, которые описывают подмножество состояний системы перед и после действий процесса, заключающегося в посылке сигналов или изменений значений переменных приложения. БР могут содержать символьные или конкретные значения параметров (переменных, входящих в пред- и постусловия, в выражения для передаваемых сигналов, в описание состояний и т.д.), причем для символьных задается диапазон их возможных значений. Фиксация конкретных значений параметров БР называется его конкретизацией.

Один БР может представлять одно или несколько требований, равно как и несколько БР или даже несколько упорядоченных множеств БР (цепочек БР) могут относиться к одному требованию.

Множество БР представляет собой формализацию исходных требований, с которой уже может работать автоматизированная система. Объединяя непротиворечивые пред- и постусловия различных БР, можно построить сценарии или трассы, позволяющие наглядно отображать возможные варианты поведения проектируемой системы. Трассы, использующие символьные параметры, называются символическими. Корректность варианта поведения, фиксируемого конкретной или символической трассой, доказывается верификатором [3, 4].

Совокупность трасс, покрывающая все цепочки всех требований, формирует модель требований разрабатываемого приложения. Эта модель может использоваться для генерации полного набора тестов, проверяющего функциональное покрытие всех конструктивно заданных требований.

Зафиксируем некоторое начальное состояние системы $s_0$ , которое включает в себя состояние среды и состояния всех погруженных в нее агентов [1]. Тогда можно получить все возможные трассы (истории функционирования системы) как последовательности

$$s_0 \xrightarrow{b_1(n_1,m_1)} s_1 \xrightarrow{b_2(n_2,m_2)} \dots \ ,$$

где $b_1(n_1, m_1), b_2(n_2, m_2), \dots$ — конкретизированные базовые протоколы. Здесь $b_1$, $b_2$, ... — имена соответствующих базовых протоколов, $n_1, n_2, \dots$ — имена ключевых агентов (ключевой агент — это фиксированный агент БР, состояние которого в данном БР разрешено изменять), а $m_1, m_2, \dots$ — наборы значений остальных параметров, удовлетворяющих предусловиям базовых протоколов. Поскольку постусловия определяют детерминированные преобразования, то состояния $s_1, s_2, \dots$ – однозначно определяются начальным состоянием системы и конкретизацией БР.

Опишем систему $\mathbf{S}(P)$, которая реализует систему базовых протоколов $P$ в форме атрибутной транзиционной системы (иначе: помеченной системы переходов — labelled transition system). Определим поведение системы $\mathbf{S}(P)$ в ее различных состояниях. Состояния системы $\mathbf{S}(P)$ отождествляются с формулами базового языка,

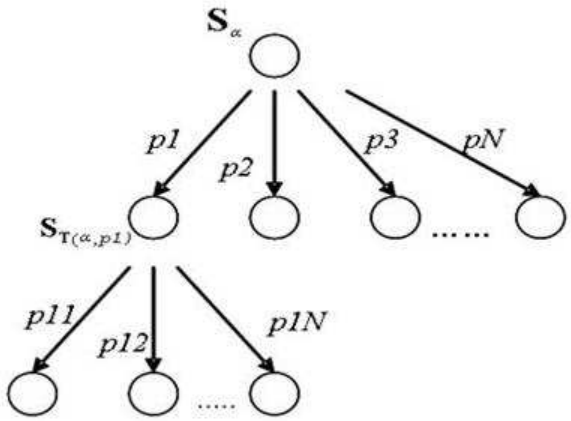


Рис. 1. Графическая интерпретация поведения системы

к которым добавляются промежуточные состояния, соответствующие выполнению ВР [5].

Поведение системы $\mathbf{S}$ в состоянии $\alpha$ обозначается как $\mathbf{S}_\alpha$. А. Летичевским доказано, что уравнение для $\mathbf{S}^\infty_\alpha$ имеет следующий вид:

$$\mathbf{S}^\infty_\alpha = \sum_{p \in P(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * \mathbf{S}^\infty_{\mathbf{T}(\alpha,p)}.$$

В этой формуле $P(\alpha) = \{p \in P_{inst} \mid \alpha \to \mathbf{pre}(p)\}$, $P_{inst}$ — множество конкретизированных протоколов, $\mathbf{proc}(p)$ — процесс базового протокола, $\beta$ — его постусловие $\mathbf{post}(p)$, $\mathbf{T}(\alpha, p) = \mathbf{Tr}(\alpha, \mathbf{post}(p))$. $\mathbf{Tr}(\alpha, \beta)$ — предикатный трансформер, $\to$ — импликация, $\Delta$ — заключительное состояние, в которое переходит система после выполнения последнего протокола из трассы $p$. На вход трансформеру $\mathbf{Tr}$ подаются две формулы $\alpha$ и $\beta$, а на выходе порождается новая формула $\gamma$, которая усиливает постусловие $\beta$, иными словами, формула $\gamma$ должна быть такой, что $\gamma \to \beta$. Такое усиление необходимо для возможности применения последующих ВР, предусловие которых является неявным следствием постусловия $\beta$.

Следующее определение включает в себя конечные трассы с успешным завершением. Для общности выделим множество $P^0$ заключительных протоколов и положим $P^1 = P \setminus P^0$. Предполагается, что заключительные протоколы могут завершать работу системы и не требуют ее обязательного продолжения. Уравнение для полной системы имеет вид:

$$\mathbf{S}_\alpha = \sum_{p \in P^1(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * \mathbf{S}_{\mathbf{T}(\alpha,p)} + \sum_{p \in P^0(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * (\mathbf{S}_{\mathbf{T}(\alpha,p)} + \Delta).$$

На рис. 1 приведена графическая интерпретация поведений системы.

# 3. Формализация критериев покрытия

Определяем множество событий $E = \{e\}$ (события могут иметь параметры), наблюдаемых при тестовых прогонах данной системы, и два его подмножества $E^{INIT} \subset E$ начальных событий и $E^{TERM} \subset E$ терминальных событий. Важно, что множество событий, определяемых в формализованной модели, является подмножеством $E$; т.е. при фактическом исполнении могут регистрироваться и другие «более мелкие» события, не используемые в формализации, но необходимые для взаимодействия системы с окружением и зависящие от него.

Далее определяем последовательности событий конечной длины — кортежи длины $n$, элементы которых могут повторяться: $E_n = \langle e_1, e_2, ..., e_n \rangle$, где $e_i = E_n^i \in E$.

Главным формальным определением является отношение вхождения одной последовательности событий в другую:

$$E_m \subset E_n \Leftrightarrow (m \leq n) \wedge (\forall i : (1 \leq i \leq m) \exists j_i : ((j_0 = 0) \wedge (j_i > j_{i-1})) \wedge (E_m^i = E_n^{j_i})).$$

Иными словами, $E_m$ входит в $E_n$, если $E_m$ или совпадает с $E_n$ , или может быть получена из $E_n$ путем удаления из $E_n$ одного или более элементов. Это отношение используется в последующих определениях критериев покрытия поведения исходных требований.

Наконец, множество поведенческих трасс (наблюдаемых поведений системы) определяется как некоторое множество таких последовательностей событий, длина которых не превосходит некоторого заданного $N$ (тем самым из рассмотрения исключаются сколь угодно длинные поведения), начинается с некоторого события из подмножества $E^{INIT}$ начальных событий и заканчивается некоторым событием из подмножества $E^{TERM}$ заключительных событий:

$$\{E_k : (k \leq N) \wedge (E_k^1 \in E^{INIT}) \wedge (E_r^k \in E^{TERM})\}.$$

Для поведенческой трассы возможен и более высокий уровень рассмотрения, при котором подпоследовательности из нескольких наблюдаемых событий рассматриваются как одно целое и вся трасса является конкатенацией таких подпоследовательностей. В предельных случаях вся трасса рассматривается как одно целое или каждая подпоследовательность состоит ровно из одного события.

Если для системы построена ее формализация в виде множества $\{BP_i\}$ базовых протоколов, то любая поведенческая трасса $T_k$ может быть представлена как кортеж базовых протоколов: $T_k = \langle BP_1, BP_2, ..., BP_n \rangle$ где $BP_i \in BP$, и в то же время как последовательность наблюдаемых событий, входящих в эти протоколы в том же порядке. В этом случае естественным «верхним уровнем» рассмотрения трассы является уровень базовых протоколов.

Если для системы построена поведенческая модель в виде графа из UCM-диаграмм [9], то различные поведения системы задаются как все возможные пути в данном графе, ведущие из какой-либо начальной точки в одну из конечных с учетом разветвлений на параллельное исполнение и циклических повторений некоторых участков. Отрезки пути между точками ветвления будем называть ветвями $(S_i)$. В ветвях присутствуют точки ответственности $RP(S_i) = \langle RP_1^i, RP_2^i, ..., RP_n^i \rangle$,

обычно задающие какие-либо действия и, следовательно, наблюдаемые события, в определенном порядке: $E(S_i) = \langle E(RP_1^i), E(RP_2^i), ..., E(RP_n^i)\rangle$, где $E(RP_j^i)$ — последовательность событий, определяемая точкой ответственности $RP_j^i$. В этом случае естественными «верхними уровнями» рассмотрения трассы является уровень точек ответственности и уровень сегментов, их содержащих.

***Функциональный критерий — иначе критерий функциональных требований***

Определим множество всех исходных требований $REQ$ для данного приложения и с каждым функциональным требованием $REQ_i \in REQ$ свяжем некоторую последовательность наблюдаемых событий (критериальную цепочку) $CC(REQ_i)$, появление которых в наблюдаемом поведении трактуется как выполнение данного требования.

Множество поведенческих трасс $T = \{E_k\}$ удовлетворяет функциональному критерию, если: $\forall i \exists j : (E_j \in T) \wedge (CC(REQ_i) \subset E_j)$, т.е. если для каждого требования $REQ_i \in REQ$ его критериальная цепочка $CC(REQ_i)$ входит в определенном выше смысле хотя бы в одну поведенческую трассу из данного множества $T$.

Если поведенческие трассы интерпретировать как записи поведения тестируемой системы при исполнении тестового набора (причем каждая поведенческая трасса соответствует прогону какого-либо одного теста из этого набора), то функциональный критерий позволяет:

1. Определять степень покрытия функциональных требований данным тестовым набором (в идеале должно быть 100%), а в случае неполного покрытия устанавливать, какие именно требования данным набором не покрываются;

2. Оптимизировать тестовый набор, исключая дублирование покрытия одних и тех же требований разными тестами (и, соответственно, разными поведенческими трассами).

***Структурный критерий — иначе критерий ветвей***

Этот критерий связан с представлением поведенческой модели системы в виде диаграмм UCM и дополняет функциональный критерий. При 100% покрытии требований по функциональному критерию вполне возможно, что некоторые ветви на диаграммах UCM-модели останутся не пройденными ни разу, поскольку в процессе оптимизации тестового набора в него отбираются только пути и принадлежащие к ним ветви, которые необходимы для функционального покрытия. В то же время некоторые из непройденных ветвей могут представлять специальный интерес для пользователя, и он хотел бы, чтобы в тестовый набор обязательно вошел тест, соответствующей поведенческой трассе, которая бы включала все события непройденной ветви. Для этого пользователь отмечает на диаграмме UCM все такие "обязательные к включению" ветви, тогда критерий ветвей состоит в том, чтобы все так отмеченные ветви оказались покрытыми данным тестовым набором. При этом если ветвь входит в состав цикла, то явно задается количество итераций (по умолчанию 2).

Поскольку все точки ответственности имеют уникальные имена, то задание какой-либо точки ответственности в пределах ветви однозначно идентифицирует эту ветвь в диаграмме.

При прогоне какого-либо теста, соответствующего пути в диаграмме UCM, все наблюдаемые события пройденных ветвей (с учетом неопределенности порядка прохождения параллельно исполняемых ветвей) сохраняются в соответствующей поведенческой трассе.

Соответственно этому структурный критерий имеет два уровня рассмотрения:

1. Уровень ветвей — задаются ветви (и, следовательно, точки ответственности), которые должны быть покрыты при прогоне тестов данного тестового набора;

2. Уровень событий — задаются последовательности событий (критериальные цепочки), которые должны появиться в поведенческой трассе при прогоне тестов.

Пусть $RP(S_i) = \langle RP_1^i, RP_2^i, ..., RP_n^i \rangle$ — последовательность точек ответственности ветви $S_i$, а $E(RP_j^i)$ — последовательность событий, определяемая точкой ответственности $RP_j^i$ и критериальная цепочка $E(S_i) = \langle E(RP_1^i), E(RP_2^i), ..., E(RP_n^i) \rangle$. Тогда множество трасс $T = \{E_j\}$ удовлетворяет структурному критерию относительно некоторого подмножества ветвей $S = \{S_i\}$ UCM-диаграммы, если $\forall i \exists j$ : $(E_j \in T) \wedge (E(S_i) \subset E_j)$ и каждая ветвь $S_i$ входит в какую-либо трассу из $T$ на уровне ветвей.

***Критерий областей действия — иначе критерий значений***

Порождаемые поведенческие трассы могут быть символическими, т.е. содержать символические значения некоторых переменных, для которых вычисляется их область допустимых значений — интервал целых или вещественных чисел или некоторое перечисление скалярных значений. При этом область допустимых значений одной переменной может зависеть от значений других символических переменных.

Для преобразования символической трассы в конкретную, в которой присутствуют только конкретные значения переменных, необходимо последовательно выбирать значения для каждой символической переменной, которая не вычисляется в этой трассе. Поскольку поведение системы одинаково для всех конкретных значений переменной из ее вычисленной области допустимых значений, то для проверки фактического исполнения достаточно взять какое-либо одно из них. Обычно — среднее и два крайних из диапазона допустимых значений и одно вне этого диапазона. После того, как какое-то значение выбрано, пересчитываются зависящие от него диапазоны допустимых значений следующей символической переменной. Процесс продолжается до тех пор, пока все символические переменные символической трассы не получат конкретные значения, которые и будут подставлены вместо них в получаемую таким образом конкретную трассу.

Поскольку перебор всех сочетаний таких выборов приводит к экспоненциальному взрыву вариантов, то пользователь самостоятельно ограничивает этот перебор только интересующими его сочетаниями.

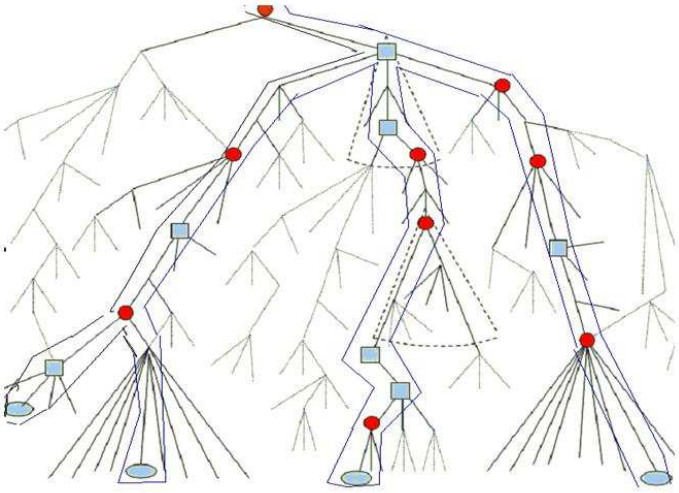

Рис. 2. Модель приложения, ограниченная эвристиками

## 4. Эвристики для генерации трасс и исполнения тестов

Для создания реализации системы ВР и методики, обеспечивающей построение цепочек, используется подход А. Летичевского, описанный в работах [2, 4], при котором система представляется в виде композиции среды и погруженных в нее агентов. Действиями среды являются размеченные действия базовых протоколов, а также внутренние действия **start**, **start** $b$ и **terminate** $b$, где $b$ — имя конкретизированного базового протокола. Агентами среды являются активизированные базовые протоколы и агенты системы.

Переходы из состояний разделяются на следующие классы:

- Изменение состояния переменных, используемых в одном из активизированных базовых протоколов;
- Завершение одного из базовых протоколов;
- Активизация нового базового протокола;
- Завершение работы всей системы.

В работе [3] показано, что такой подход обеспечивает реализацию системы $\mathbf{S}_\alpha$.

Уравнение для полной системы отражает модель, учитывающую все возможные варианты поведения системы (рис. 1). Такое представление избыточно. В условиях реальных проектов нужны модели, отражающие только то подмножество поведений, которое требуется системе и соответствует требованиям на систему. Одним из методов наложения ограничений является введение эвристик [6]. Каждая эвристика представляется упорядоченным множеством $M_i$ базовых протоколов, которые включены в определенной последовательности в сценарий поведения системы (трассу).

При условии наличия множества ограничивающих эвристик $M$ уравнение системы принимает следующий вид:

$$\mathbf{S}_\alpha^M = \sum_{p \in P_M^1(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * \mathbf{S}_{\mathbf{T}(\alpha,p)} + \\ + \sum_{p \in P_M^0(\alpha)} \mathbf{proc}(p) * (\mathbf{T}(\alpha, p) : \Delta) * (\mathbf{S}_{\mathbf{T}(\alpha,p)} + \Delta) \ ,$$

где $P_M(\alpha) = \{p \in P_{inst} \mid p \textit{ удовлетворяет } M \textit{ и } \alpha \rightarrow \mathbf{pre(p)}\}$, $P_M^1 = P_M \setminus P_M^0$, $P_M^0$ — множество заключительных базовых протоколов из $P_M(\alpha)$ . Очевидно, что из $P_M(\alpha) \subset P(\alpha)$, следует $\mathbf{S}_\alpha^M \subset \mathbf{S}_\alpha$ . Рис. 2 иллюстрирует модель $\mathbf{S}_\alpha^M$, выделенную из модели $\mathbf{S}_\alpha$ цепочками ВР (упорядоченными последовательностями ВР) или эвристиками.

Рассмотрим эвристику M, такую, что M $= \{M_1, M_2, ..., M_N\}$, $M_i = \{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$, где $p_{i_1}, p_{i_2}, ..., p_{i_k}$ — базовые протоколы. При этом $M_i$ задает следующее поведение (будем считать, что данное поведение достижимо и после применения протокола $p_{i_k}$ система переходит в состояние $\Delta$):

$$\mathbf{S}_\alpha^{M_i} = (\prod_j \mathbf{proc}(p_j) * (\mathbf{T}(\alpha_j, p_j) : \Delta)) * \Delta \ ,$$

где последовательность $p_{i_1}, p_{i_2}, ..., p_{i_k}$ встречается ровно один раз. Заметим, что в $\mathbf{S}_\alpha^{M_i}$ между протоколами из последовательности $p_{i_1}, p_{i_2}, ..., p_{i_k}$ могут находиться другие протоколы.

Например, на рис. 2, базовые протоколы эвристик (отмечены кружочками) перемежаются с базовыми протоколами, не входящими в эвристики (отмечены квадратиками). Эвристики обеспечивают навигацию в дереве поведения (на рис. 2 две эвристики — окаймлены сплошной линией — направляют поиск трассы от исходной до терминальной вершины). Кроме того, каждый базовый протокол эвристики уменьшает пространство поиска (суженные области поиска на рис. 2 отмечены пунктиром).

Наложим ограничение на $\mathbf{S}_\alpha^{M_i}$ — будем выбирать только одну символьную трассу, содержащую последовательность $M_i = \{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ , и обозначим эту трассу как $T_{M_i}$.

Пусть $T_M = \sum_i T_{M_i}$. Тогда $|T_M| =$ N и из того, что $\mathbf{S}_\alpha^{M_i} \subset \mathbf{S}_\alpha$ следует, что $T_M \subset \mathbf{S}_\alpha$. Тем самым $T_M$ соответствует N трассам, полученным при генерации системой VRS с учетом заданных эвристик, и является их абстрактной моделью.

Запишем трассу $T_{M_i}$ в виде пучка трасс, который соответствует подстановке конкретных значений в символические переменные ВР:

$$\mathrm{C}(T_{M_i}) = \sum_{D_i} (\prod_j \mathbf{proc}(p_j) * (\mathbf{T}(\alpha_j, p_j) : \Delta)) * \Delta \ ,$$

где $D_i$ — максимальное по включению множество конкретных значений, при которых трасса $T_{M_i}$ может быть построена.

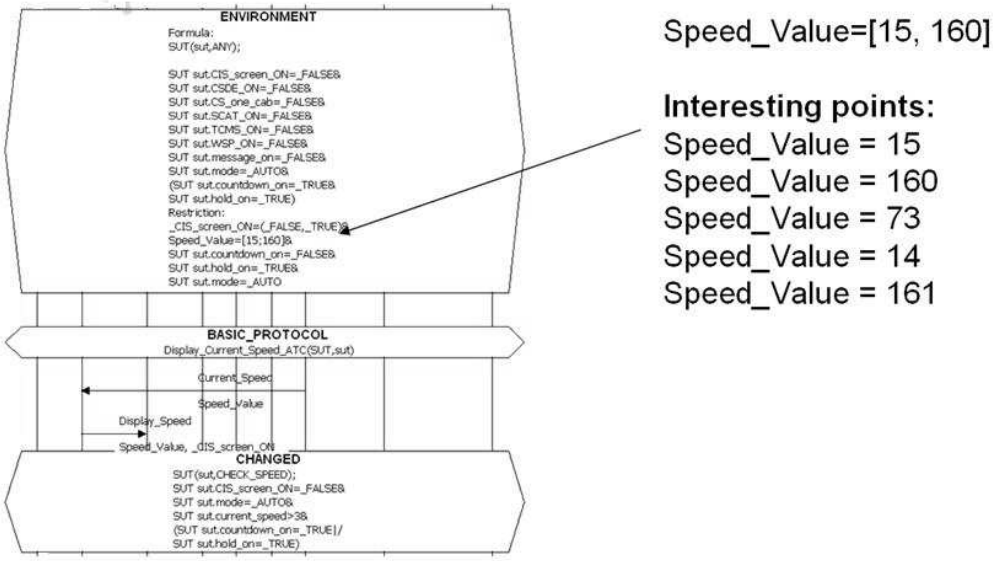


Рис. 3. Фрагмент пучка трасс на основе символьной трассы для переменной типа integer

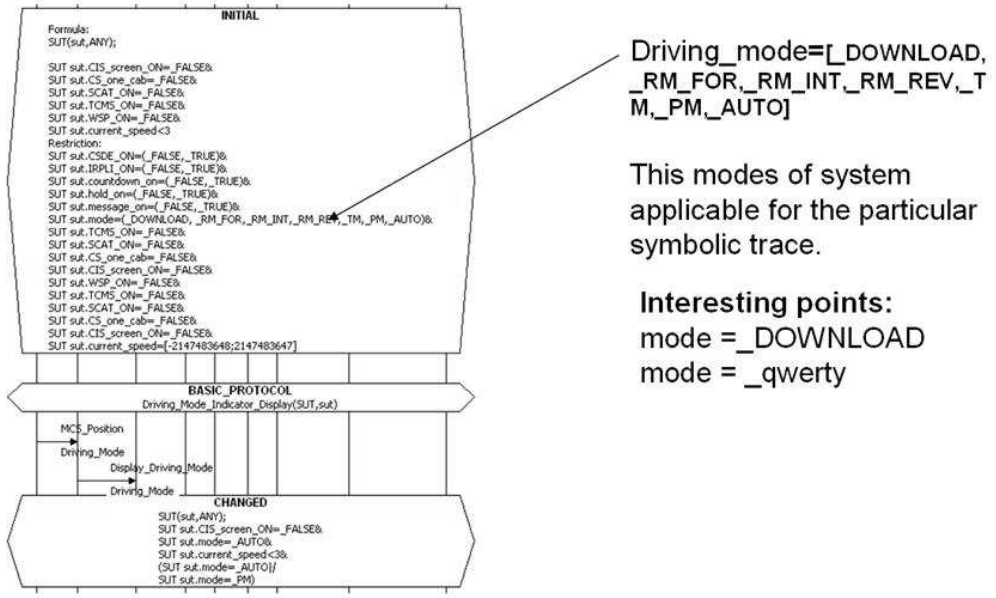


Рис. 4. Фрагмент пучка трасс на основе символьной трассы для переменной типа enumeration

Например, пучок конкретных трасс на рис. 3 определяется диапазоном значений целой переменной Speed_Value = [15, 160], а на рис. 4 определяется диапазоном значений переменной перечислимого типа Driving_mode = [_DOWNLOAD, _RM_FOR, _RM_INT, _RM_REV, _TM, _PM, _AUTO].

"Interesting points" на рис. 3 и рис. 4 обозначают конкретные значения, которые подставляются в трассу и тем самым фиксируют середину и края диапазона, а также выход за диапазон. Кроме того, разрешен пользовательский выбор значений.

Обозначим $(T_M) = \{\mathrm{C}(T_{M_i})\}$ и $\mathrm{D} = \{D_i\}$ . Тогда если D не пусто, то D является областью определения модели $\mathrm{C}(T_M)$.

Таким образом, эвристика $T_M$ совместно с множеством D дает фильтр, который, присоединяясь к модели $\mathbf{S}_\alpha$, вырезает из множества только те трассы с конкретными значениями, которые принадлежат $\mathrm{C}(T_M)$. Полученная модель служит исходными данными для генерации кода приложения на языке платформы и, объединяясь с фильтром, обеспечивает надежность кода с точностью до контекста модели $\mathrm{C}(T_M)$.

Сгенерированные по $\mathrm{C}(T_M)$ трассам наборы тестов, варьирующие значения переменных по разрешенным диапазонам (например так: левая или правая граница диапазона, середина диапазона, выход за диапазон) дают при тестировании экспериментальное подтверждение результатов верификации.

Результаты применения разработанной модели образуют методику, следующие свойства которой обеспечивают успешное применение в промышленных проектах:

1. Характерной особенностью модели $\mathbf{S}_\alpha^{M_i}$ (для всех $i$) является удовлетворение исходным требованиям при условии, что все требования покрываются множеством эвристик M.

2. Линейные символические трассы, сгенерированные верификатором, создают поведенческую модель поведения с зафиксированным потоком управления. Они представляют собой классы эквивалентности по управлению, инвариантные относительно допустимых значений переменных — параметров состояния, использованных в конкретных трассах.

3. Для каждого шага конкретной трассы (т.е. для каждого использованного ВР) верификатор вычисляет диапазоны допустимых значений параметров состояния. Пробегая по всем допустимым значениям параметров, мы получаем из одной символической трассы множество конкретных трасс одного класса эквивалентности.

4. Для каждой трассы верификатор вычисляет формулу ее области определения.

5. Объединение формул конкретных трасс дает формулу области определения всей модели.

6. Результатом представления области определения модели в виде формулы и эвристики в виде текста на языке программирования является фильтр, который при присоединении к модели вырезает из множества возможных трасс (поведений) только те, которые принадлежат модели $\mathbf{S}_\alpha^{M_i}$ (для всех $i$).

7. Полученная модель служит исходными данными для генерации кода приложения на языке целевой платформы. Код приложения, объединяясь с кодом фильтра, обеспечивает свойство надежности для кода приложения с точностью до контекста модели (множества базовых протоколов проекта, из которых построена модель).

8. Подмножество символических трасс — как минимум по одной из каждого класса эквивалентности — обеспечивает покрытие множества поведений в соответствии с функциональным и структурным критериями.

9. Подмножество конкретных трасс, получаемых из символических — дает исходные данные для генерации тестового набора, обеспечивающего исчерпывающую проверку функциональности приложения в соответствии с исходными требованиями.

10. Тестирование позволяет экспериментально подтвердить соответствие модели требованиям.

## 5. Заключение

Настоящая работа была сфокусирована на разработку частного случая формальной модели Летичевского, направленного на ограничение поведенческих характеристик модели приложения таким способом, чтобы, сохраняя согласование с исходными требованиями на проектируемое приложение, обеспечить для упрощенной модели свойства корректности и полноты по отношению к исходным требованиям. Детальность ограниченной модели должна быть достаточной, чтобы обеспечить автоматическую генерацию исполнимого кода модели приложения и кода тестов.

Разработанная методика пилотировалась в технологии VRS/TAT [7, 8], где обеспечила существенное сокращение трудоемкости проектирования телекоммуникационных приложений.

## Список литературы

1. Letichevsky A.A., Kapitonova J.V., Volkov V.A., Vyshemirskii V.V., Letichevsky Jr. A.A. Insertion Programming // Cybernetics and Systems Analysis. 2003. Volume 39, Issue 1 (January 2003). P. 16–26.

2. Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky O.O., Volkov V.V., Baranov S.N., Weigert T. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Proc of ISSRE04 Workshop on Integrated Reliability Engineering (ISSRE04:WITUL), IRISA, Rennes France, 2004.

3. Летичевский О.А. Верификация и тестирование интерактивных систем, специфицированных базовыми протоколами: дис. ... канд. физ.-мат. наук. Киев, 2005. 138 с.

4. 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, Russia, 2005. P. 81–86.

5. Letichevsky A., Gilbert D. A Model for Interaction of Agents and Environments // D.Bert, C.Choppy, P.Moses (Eds), Resent Trends in Algebraic Development Techniques. 1999. LNCS 1827. P. 311–328.

6. Колчин А.В. Разработка инструментальных средств для проверки формальных моделей асинхронных систем: дис. ... канд. физ.-мат. наук. Киев, 2009. 140 с.

7. Baranov S., Kapitonova J., Letichevsky A., Kolchin A., Letichevsky A. (Jr)., Radchenko V., Potiyenko S., Volkov V. Tools For Requirements Capturing Based on the Technology of Basic Protocols // Proc. of St. Petersburg IEEE Chapter, International Conference, May 18-21. St. Petersburg, Russia, 2005. P. 92–97.

8. Баранов С.Н., Котляров В.П., Летичевский А.А. Индустриальная технология автоматизации тестирования мобильных устройств на основе верифицированных поведенческих моделей проектных спецификаций требований: Международная научная конференция: космос, астрономия и программирование. Лавровские чтения. 20–22 мая, мат.-мех. факультет СПбГУ. СПб, 2008. С.134–145.

9. Recommendation ITU-T Z.151 User requirements notation (URN) — Language Definition, 2008.

## A Formal Requirements Model, Used in the Process of Application Code and Test Code Generation

Baranov S.N., Kotlyarov V.P.

**Keywords:** requirements model, behavior traces, basic protocols, model scope

The considered model is used in manual development of application specifications and is based on the theory of basic protocols and respective symbolic verification tools. Means to limit the behavioral characteristics of the model still matching the source requirements are discussed. If the model is verified successfully, the executable code of the application and the respective test code are generated from the model. The technique of using the developed model is described.

**Сведения об авторах:**

**БАРАНОВ Сергей Николаевич**,

Санкт-Петербургский государственный политехнический университет,
доктор физ.-мат. наук, профессор.

Окончил Ленинградский университет в 1972 г., в разное время работал в СПбГУ, СПИИРАН, ЗАО "Моторола ЗАО"; ведет исследования в СПИИРАН, научный консультант ООО "Моторола Мобилити". Основные научные интересы: технология и языки программирования, методы компиляции, анализ и верификация программных спецификаций, формальные методы, символьные вычисления.

**КОТЛЯРОВ Всеволод Павлович**,

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

Окончил Ленинградский политехнический университет в 1972 г. В течение более 30 лет участвует в промышленных и научно-технических проектах по линии университета, РАН и крупных промышленных компаний Санкт-Петербурга. С 1993 по 2008 г. работал в Санкт-Петербургском центре компании "Моторола", совмещая научную деятельность с преподаванием. Основные научные интересы: программная инженерия, инструментальные средства автоматизации процесса разработки программного продукта, вопросы верификации и тестирования на базе формализованных спецификаций программного проекта.