УДК 004.4'242

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ К ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ АВТОМАТНЫХ ПРОГРАММ

А.Ю. Законов, А.А. Шалыто

Описан подход к автоматизации тестирования автоматных программ. Для формализации требований спецификации к модели и объектам управления предлагается использовать контракты. Тест описывается как последовательность переходов в модели. Для автоматизации процесса создания кода теста предложен генетический алгоритм, который позволяет находить значения переменных, удовлетворяющие условиям на переходах.

Ключевые слова: тестирование, автоматное программирование, контракты, генетические алгоритмы.

Введение

Автоматная программа состоит из конечных автоматов и набора объектов управления, с которыми взаимодействуют автоматы [1]. Наиболее распространенным способом проверки автоматных программ является Model Checking [2], так как для этого класса программ уровень автоматизации процесса верификации может быть повышен по сравнению с традиционными программами. Это объясняется тем, что в автоматных программах поведенческая модель задается априори, а не строится по программе, как это делается обычно. Однако проверка моделей позволяет верифицировать только автоматы, но не систему в целом. Поведение объектов управления и их взаимодействие с автоматами не проверяются при указанном подходе. Таким образом, в автоматной программе могут остаться невыявленные ошибки, даже если система автоматов была успешно верифицирована относительно данной спецификации.

В данной работе используется тестирование для проверки автоматных программ в целом. Тестирование является трудоемкой и ресурсоемкой задачей. Около половины всего времени разработки проекта часто тратится на тестирование [3]. В последнее время многие исследования посвящены теме автоматизации процессов создания и запуска тестов [4]. Несмотря на то, что успешное тестирование не гарантирует отсутствие ошибок в программе, большой набор тестов может существенно помочь в обнаружении ошибок в логике работы программы и в ее реализации. Это может обеспечить повышение вероятности того, что программа будет успешно решать поставленные задачи.

В данной работе предлагается подход к тестированию автоматных программ и способ автоматизации процесса создания тестов на основе использования генетических алгоритмов.

В предлагаемом подходе тестирование применяется для проверки соответствия спецификации системы ее реализации. Спецификация, заданная на естественном языке, пригодна только при ручном тестировании. Для автоматизации процесса тестирования спецификация должна быть записана на формальном языке. Поэтому предлагается внести максимально возможное число требований спецификации в описание автомата. Это позволит расширить программу так, что она будет содержать в себе требования для проверки. Для описания моделей автоматных программ естественно использование конечных автоматов. Однако в реактивных системах в конечных автоматах в качестве входных воздействий обычно применяются события. Для моделирования поведения сложных систем, взаимодействующих со средой, в работе предлагается использовать расширенные конечные автоматы, которые позволяют использовать переменные в модели и с их помощью задавать охранные условия на переходах.

Зависимость поведения системы от значений этих переменных позволяет описать сложную логику работы и тем самым внести значительную часть спецификации в модель. В работе [5] предлагается использовать контракты [6] для внесения дополнительных требований спецификации в модель автоматных программ. Наличие исполняемой спецификации внутри автоматной модели позволяет при запуске программы автоматически проверять выполнение указанных условий. Более того, формальное описание условий позволит реализовать автоматическую генерацию тестов и искать значения, которые помогут выявить несоответствия реализации и спецификации.

Создание теста для автоматной программы

Большая часть автоматных программ разрабатывается для взаимодействия с внешней средой: программа получает события и входные переменные, а автомат реагирует на эти воздействия и реализует выходные воздействия. В автоматных системах для реализации выходных воздействий используются объекты управления. Они получают из среды события и входные воздействия, которые используются в автомате в охранных условиях на переходах или передаются как аргументы вызовов методов других объектов управления. В расширенном конечном автомате все эти значения можно представить переменными. Таким образом, переменные, использованные в модели, разделяются на два типа: внутренние, заданные и определенные внутри модели, и внешние, полученные из среды от объектов управления. Во время тестирования значения внешних переменных необходимо генерировать вместе с кодом теста.

Учитывая приведенное выше описание модели и ее спецификации, определим понятие теста для автоматной программы. В случае традиционного подхода к разработке программного обеспечения создание тестов осуществляется путем представления требований спецификации в виде программного кода. В предложенном подходе тест определяется не в виде кода, а в терминах автоматной модели и требований спецификации. Тест автоматной программы – последовательность переходов автомата. Тест также содержит требования спецификации к задействованным переходам, состояниям и объектам управления. Такое описание теста значительно понятнее традиционного описания теста, так как не требует изучения исходного кода и позволяет описать важные тестовые сценарии уже на этапе написания спецификации, а не при написании кода программы. Это помогает преодолеть семантический разрыв между реализацией программы и ее спецификацией.

Предложенное определение теста удобно для его описания, но не пригодно для автоматического создания кода этого теста. Чтобы автомат выполнил описанный в тесте сценарий, необходимо определить последовательность событий и набор значений внешних переменных, которые приводят к выполнению автоматом заданной последовательности переходов. Для поиска значений внешних переменных в данной работе предлагается использовать генетические алгоритмы.

Существующие инструменты для тестирования моделей

При тестировании моделей, построенных с использованием расширенных конечных автоматов, выделяется две подзадачи: выбор или генерация выполнимых путей (feasible paths) для тестирования и подбор входных значений для выполнения этих путей.

Выбор выполнимых путей для тестирования изучен во многих работах [7, 8]. Также пути для тестирования можно задавать вручную, выбирая наиболее интересные сценарии для проверки. Для путей, полученных автоматически или вручную, необходимо найти значения переменных для их прохождения с учетом всех охранных условий расширенного конечного автомата. Задача поиска этих значений менее изучена, но не менее актуальна, так как без этих значений невозможно сгенерировать код теста, который проверит заданную последовательность действий.

Существует ряд инструментов для работы с моделями и создания тестов на их основе. В работе [9] рассмотрены инструменты для тестирования на основе моделей программ, написанных на языке C#. При этом рассматривается инструмент NModel, который позволяет создавать модели на текстовом языке, автоматически строить по модели конечные автоматы, описывающие переходы в системе, и генерировать тестовые сценарии для этой системы.

В работе [10] приведен обзор инструментов для тестирования на основе моделей и предложен инструмент ModelJUnit, который позволяет описывать расширенные конечные автоматы как классы на

языке Java и автоматически генерировать для них тесты. Инструмент Tigris MBT позволяет графически описывать расширенные конечные автоматы и, используя случайные значения, пытается найти последовательность событий и набор переменных для полного покрытия переходов или для достижения определенного состояния.

Институт системного программирования РАН предлагает семейство инструментов разработки тестов на основе моделей UniTesK [11], которые предлагают использовать «спецификации ограничений» для построения модели системы, используя ее интерфейсы. Для написания тестов используется SeC – спецификационное расширение языка C.

Перечисленные инструменты не решают проблему тестирования автоматных программ. Во-первых, эти инструменты предполагают ручное создание требований для проверки отдельно от модели, что делает процесс разработки более сложным. Во-вторых, принципиальное отличие автоматных программ от моделей, используемых в существующих инструментах, состоит в том, что автоматная программа включает в себя, кроме автомата, также и объекты управления. Тестирование только автомата не позволяет найти ошибки в программе в целом.

Включение спецификации в модель

Спецификация, записанная на естественном языке, пригодна только для ручного тестирования. Предлагаемый подход развивает идею, изложенную в работе [5], и предлагает использовать контракты для записи требований не только к модели, но и к объектам управления. Наличие исполняемой спецификации внутри автоматной модели позволяет при запуске программы автоматически проверять выполнение описанных условий как моделью, так и объектами управления.

Объекты управления, с которыми взаимодействует автомат, имеют спецификацию на входные воздействия, результаты их работы и особенности взаимодействия с автоматом. Все эти требования спецификации должны быть выполнены во время корректной работы программы. В предложенном подходе спецификация объектов управления включается в описание автоматной модели. Благодаря этому особенности реализации конкретных объектов управления не важны на этапе создания тестов, так как достаточно проверять, что на вход им подаются корректные значения и что автомат корректно обрабатывает любые значения, полученные из объектов управления, которые не противоречат их спецификации.

В предложенном подходе для интеграции требований спецификации в описание автоматной модели используются контракты, записанные на языке JML [6]. В случае автоматной модели предлагается задавать переходам пред- и постусловия, а для состояний записывать инварианты:

- предусловия перехода определяют ожидания системы относительно значений переменных при вызове методов объектов управления, которые используются на переходе;
- постусловия перехода задают требования к значениям переменных модели при завершении вызова методов объектов управления;
- инварианты, заданные для состояния, содержат требования, которые должны выполняться на протяжении всего времени пребывания системы в указанном состоянии.

Создание сценариев для тестирования и поиск значений входных переменных

Проанализировав спецификацию, заданную на естественном языке, необходимо выбрать сценарии работы, интересные для тестирования. Выбранные сценарии далее записываются в автоматных терминах как последовательность переходов автомата. Такое представление теста должно быть интуитивно понятно как автору спецификации, работающему с требованиями, записанными на естественном языке, так и разработчику системы, работающему с автоматной моделью и кодом объектов управления.

Существует ряд исследований [7], в которых изучается задача генерации пути в расширенном конечном автомате, который обеспечивал бы максимальное покрытие переходов и состояний. Описанные технологии автоматического создания сценариев можно успешно сочетать с подходом, представленным в настоящей работе, для создания эффективных наборов тестов.

Автомат реагирует на события и выполняет переходы в зависимости от значений переменных автоматной модели, которые используются в охранных условиях на переходах. Таким образом, возникает задача поиска последовательности событий и набора значений внешних переменных, соответствующих заданной последовательности переходов в модели. В данной работе предложен алгоритм для автоматизации решения этой задачи.

Последовательность событий однозначно определяется по последовательности переходов. Сложнее подобрать значения переменных – они должны удовлетворять ряду требований. Во-первых, охранные условия на всех переходах в описанном пути должны быть выполнены. Во-вторых, все требования спецификации объектов управления должны выполняться, так как при реальном использовании значения этих переменных будут приходить из объектов управления с данными спецификациями.

Применение генетического алгоритма для поиска значений переменных

Генетические алгоритмы считаются успешными для решения оптимизационных задач, в том числе и задач генерации тестов [4, 7, 12]. Задачу поиска набора значений, при котором будет выполнен заданный путь в расширенном конечном автомате, можно свести к задаче оптимизации.

Набор внешних переменных, задействованных на выбранной последовательности переходов, можно рассматривать как вектор значений $<x_1, x_2, \ldots, x_n>$, где x_i – значение внешней переменной, а n – число внешних переменных для этого пути. Для генетического алгоритма набор внешних переменных является хромосомой. Определим для него функцию приспособленности – функцию, которая оценивает, насколько хромосома решает задачу, и позволяет выбирать лучшие решения из всего поколения.

В рассматриваемой задаче функция приспособленности на вход принимает вектор значений и выдает число, характеризующее приспособленность этого вектора значений для выполнения заданного пути. Чем меньше значение этого числа, тем больше подходит данный вектор значений. Если же функция приспособленности равна нулю, то требуемый путь выполняется. Таким образом, задача поиска подходящего набора значений внешних переменных сводится к задаче оптимизации, где требуется найти вектор, которому соответствует минимальное значение функции приспособленности.

Один ген хромосомы – это значение одной из внешних переменных для заданного пути. В работе используется одноточечное (классическое) скрещивание (one-point crossover), в котором две хромосомы разрезаются один раз в соответствующей точке и производится обмен полученными частями.

Оператор мутации (mutation operator) необходим для защиты генетического алгоритма от преждевременной сходимости. В данной работе мутация производит замену произвольного гена на случайно выбранное число из области допустимых значений.

Функция приспособленности

Функция приспособленности должна удовлетворять следующему условию: чем лучше особь подходит для поставленной задачи, тем меньше значение функции. Таким образом, проблема сводится к задаче оптимизации – найти решение с нулевым значением функции приспособленности.

Однозначного способа определения функции приспособленности не существует. В работе [12], описывающей применение генетических алгоритмов для тестирования структурных программ, для определения приспособленности хромосом используется такой критерий, как расстояние до условия (branch distance). Расстояние до условия позволяет оценить, насколько близка была данная хромосома к выполнению конкретного условия, которое на практике не было выполнено. Например, для условия `A=B` расстояние до условия будет вычисляться по формуле |A-B|. Чем меньше значение |A-B|, тем ближе значение A к B и тем ближе хромосома к тому, чтобы это условие было выполнено. Если условие выполнено, то расстояние до условия равно нулю.

Для вычисления приспособленности хромосомы относительно корректного выполнения заданного пути в автоматной модели следует учитывать два типа условий:

1. охранные условия на переходах автомата;
2. требования спецификации методов объектов управления, которые задействованы на переходах.

Все эти условия обязательны для выполнения. Хромосомы, которые нарушают любое из указанных условий, не подходят для создания тестов, так как система в соответствии со своей спецификацией не должна поддерживать работу с этими значениями. В этом случае функция приспособленности должна оценить, насколько близка хромосома к тому, чтобы выполнить нарушенные условия. Это делается при помощи вычисления расстояний до условий.

Для последовательности переходов может быть задано большое число условий, поэтому расстояние до условия всего пути необходимо вычислять по отдельности, рассматривая условия на каждом переходе этого пути. Каждый переход описывается набором параметров:

- событие, по которому этот переход может произойти;
- охранное условие, которое должно быть выполнено для совершения перехода;
- действия на переходе: вызов методов объектов управления, получение значений внешних переменных из среды или изменение значений переменных модели;
- предусловия перехода;
- постусловия перехода.

Следовательно, даже в рамках одного перехода может быть задействовано большое число условий. Для более точного вычисления расстояния до условия перехода в работе каждый переход разбивается на несколько меньших шагов:

1. получение события, поиск возможных переходов и проверка их охранных условий;
2. проверка предусловий перехода, выполнение перехода и заданных действий на переходе;
3. проверка постусловий перехода.

При выполнении каждого из этих шагов могут быть обнаружены ошибки. Переход считается выполненным успешно, если все три шага выполнены успешно. Таким образом, сценарий теста разбивается на переходы, а каждый переход разбивается на три шага. После этого исходная последовательность переходов рассматривается как последовательность шагов. Оценка приспособленности всей последовательности шагов вычисляется как сумма оценок для каждого шага по отдельности. Каждый шаг оценивается по формуле вычисления расстояния для условия.

Следует заметить, что шаги выполняются последовательно и что выполнение шагов в начале пути важнее, чем в конце пути. Например, если выполнены условия всех шагов, кроме первого, то сумма расстояний до условия будет маленькой, так как для всех шагов, кроме первого, это значение будет равно нулю. На практике эта хромосома не позволяет пройти ни одного шага, так как для того, чтобы успешно пройти второй шаг, необходимо выполнить все условия на первом шаге. В предложенном подходе расстояния до условия шагов суммируются с учетом местоположения этих шагов в пути – используется взвешенная сумма. Если для одного шага задано несколько условий, то расстояние до условия этого шага вычисляется как сумма расстояний до условия каждого из этих условий.

Разработанные и используемые инструментальные средства

Для создания автоматной модели предлагается использовать инструмент yEd [13], который позволяет удобно графически описывать расширенные конечные автоматы, содержащие переменные и охранные условия. Возможность задания дополнительных меток к переходам и состояниям можно использовать для записи JML-контрактов.

Для поиска значений внешних переменных для заданного пути разработан инструмент GenValueFinder, который принимает на вход последовательность переходов автоматной модели и, используя описанный генетический алгоритм, осуществляет поиск значений внешних переменных для прохождения этого пути. Генерация кода теста, пригодного для запуска, происходит при помощи разработанного инструмента TestGenerator. На вход подаются файлы, полученные в результате работы программы GenValueFinder.

Проверка выполнения контрактов выполняется при помощи существующего инструмента JML Runtime Assertion Checker [14], который входит в стандартный набор утилит при установке JML. Инструмент в режиме реального времени проверяет, что все контракты, добавленные в Java-код в виде аннотаций, выполнены.

Пример

Проиллюстрируем предложенный подход на примере разработки банкомата, соответствующего приведенной спецификации на естественном языке.

1. Банкомат позволяет пользователям снимать деньги с определенного счета.
2. Сначала на счету находится сумма от 0 до 100 000.
3. Пользователь может снимать деньги произвольное число раз, пока на счету неотрицательная сумма.
4. Ввод суммы для снятия происходит с клавиатуры. Пользователь может ввести число от 1000 до 15 000.
5. В день со счета можно снять не более 50 000.

Спецификация в таком виде удобна для разработчика и заказчика, но такую спецификацию можно проверять и тестировать только вручную.

Выделение объектов управления и построение автоматной модели. Для формализации спецификации необходимо выделить спецификацию объектов управления и требования к логике работы системы, которые будут описаны при помощи автоматов. Как видно из текста спецификации, в системе задействованы два объекта управления:

1. *Счет* отвечает за взаимодействие с банковским счетом (вернуть число денег на счету, снять деньги со счета);
2. *Клавитура* отвечает за взаимодействие с пользователем (ввод суммы для снятия на клавиатуре устройства).

Требования, которые описывают логику работы системы и также должны быть учтены при построении автоматной модели, состоят в следующем:

- Счет: вначале на счету находится сумма от 0 до 100 000.
- Клавиатура: пользователь может ввести число от 1000 до 15 000.

В данной модели будут использованы две внешние переменные:

1. `ext_sum` – количество денег на счету.
2. `ext_x` – сумма для снятия во время транзакции, которую вводит пользователь.

Внутренняя переменная `today` используется для суммирования снятых за сеанс работы денег. Добавление этой переменной позволяет записать условие, которое ограничивает число снятых денег за один сеанс.

На рисунке приведена модель, представленная в виде расширенного конечного автомата и требований, записанных при помощи контрактов.

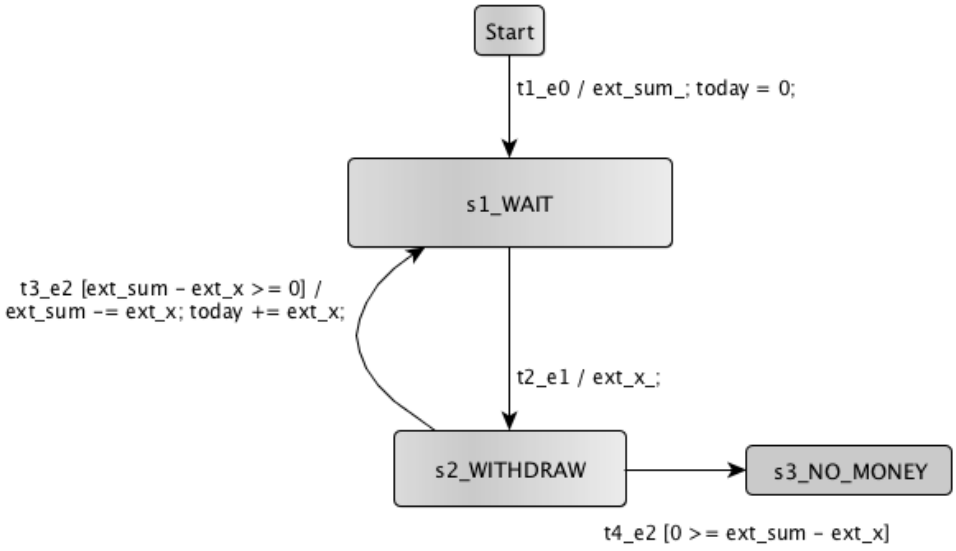

Рисунок. Расширенный конечный автомат, содержащий контракты

В модель включены требования спецификации к объектам управления. Клавиатура: `@ensures ext_x >= 1000 && ext_x <= 15000`. Счет: `@ensures ext_sum >= 0 && ext_sum <= 100000`. Требование спецификации, ограничивающее число снятых денег за сеанс работы, добавлено как контракт к состоянию: `@invariant today <= 50000`.

Пример тестовых сценариев и создания тестов. Проанализировав текстовую спецификацию системы, предложим сценарий использования и создадим тест на его основе. Сначала запишем сценарий на естественном языке: три раза снимаются деньги со счета и на счету заканчиваются средства на четвертой попытке.

Следующий шаг – формальная запись сценария в терминах автоматной модели. Тестовый сценарий, записанный как последовательность переходов, имеет вид

– `path.txt: t1, t2, t3, t2, t3, t2, t3, t2, t4.`

В тестовом сценарии задействовано пять внешних переменных: `ext_sum` – изначальная сумма на счету; `ext_x1` – пользователь ввел первый раз; ... `ext_x4` – пользователь ввел четвертый раз.

Для поиска подходящих значений использован разработанный инструмент `./GenValueFinder model.xml path.txt`. Значения найдены за 10 секунд:

– файл `events.txt: e0 e1 e2 e1 e2 e1 e2 e1 e2;`
– файл `variables.txt: 33177 13115 14485 4382 8513.`

Код теста генерируется запуском инструмента TestGenerator. Далее необходимо скомпилировать тест командой jmlc и запустить при помощи команды jmlrac. Запуск теста выдает следующий результат:

```
Exception in thread "main" JMLInvariantError: by method Generated-
Test.transition5 File "GeneratedTest.java", line 34
```

Данное описание ошибки позволяет понять, что условие `invariant` было нарушено после выполнения пятого перехода. Таким образом, при найденных значениях внешних переменных на тестируемом сценарии обнаружено несоответствие спецификации и реализации.

Заключение

Использование предложенного подхода к тестированию автоматных программ позволит повысить качество разрабатываемых систем и проверять соответствие реализации системы заданной спецификации. В результате работы получены следующие результаты:

– предложен способ проверки автоматных программ в целом;
– разработан метод нахождения входных параметров для выполнения заданного сценария в автоматной модели при помощи использования генетических алгоритмов;
– разработан инструмент для автоматизации предложенного подхода.

Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы» в рамках государственного контракта П2373 от 18 ноября 2009 года.

Литература

1. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2010. – 176 с.
2. Кларк Э.М, Грамберг О., Пелед Д. Верификация моделей программ. Model Checking. – М.: МЦНМО, 2002. – 416 с.
3. Myers G. The Art of Software Testing. – John Wiley & Son. Inc, 2004.
4. McMinn P. Search-based software test data generation: a survey: Research Articles // Software Testing, Verification & Reliability. – 2004. – № 14 (2). – P. 105–156.
5. Степанов О.Г. Методы реализации автоматных объектно-ориентированных программ. Диссертация на соискание ученой степени кандидата технических наук. – СПб: СПбГУ ИТМО, 2009. – Режим доступа: http://is.ifmo.ru/disser/stepanov_disser.pdf, своб.
6. Meyer B. Applying design by contract // Computer. – 1992. – 25(10). – P. 40–51.
7. Kalaji A.S., Hierons R.M. and S. Swift. Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM) // Software Testing, Verification, and Validation (ICST). 2nd International IEEE Conference. – 2009. Denver, Colorado: IEEE.
8. Duale Y., Ümit Uyar M. A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models // IEEE Transactions on Computers. – 2004. – Vol. 53. – № 5. – P.614–627.
9. Jacky J., Veanes M., Campbell C., Schulte W. Model-Based Software Testing and Analysis with C#. – Cambridge University Press, 2008.
10. Mark U., Legeard B. Practical Model-Based Testing: A Tools Approach. – Morgan–Kaufmann, 2007.
11. Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A. UniTesK Test Suite Architecture // Proc. of FME 2002. LNCS 2391. – Springer-Verlag, 2002. – P. 77–88.
12. Wegener J., Buhr K., Pohlheim H. Automatic test data generation for structural testing of embedded software systems by evolutionary testing // Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). – NY. – 2002. – P. 1233–1240.
13. Веб-сайт yEd – Graph Editor [Электронный ресурс]. – Режим доступа: www.yworks.com/en/products_yed_about.html, своб.
14. Cheon Y., Leavens G.T. A Runtime Assertion Checker for the Java Modeling Language (JML) // Proceedings of the SERP '02, Las Vegas, Nevada, USA. – CSREA Press, June 2002. – P. 322–328.

Законов Андрей Юрьевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, andrew.zakonov@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru