УДК 519.686.2

# РАЗРАБОТКА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ДЛЯ ПРОВЕРКИ ФОРМАЛЬНЫХ МОДЕЛЕЙ

*А.В. Колчин*

Институт кибернетики им. В.М. Глушкова НАН Украины,
03680, Киев, проспект Академика Глушкова, 40.
E-mail: kolchin_av@yahoo.com

Разработан программный комплекс для автоматической проверки динамических свойств формальных моделей и построения тестовых сценариев. Описаны методы направленного поиска, а так же методы оптимизации обхода пространства поведения.

Software for automated checking of dynamic properties of formal models together with test scenarios generation developed. Guided search and optimizing solutions on behavior space traversal described.

## Введение

Актуальной задачей при создании программных комплексов является проверка правильности их функционирования. С ростом производственных потребностей возрастает сложность поведения программ. При этом требования, предъявляемые к качеству, становятся все более критичными – все чаще автоматизируются системы управления машин, от функционирования которых зависят жизни людей. Тем не менее, с увеличением сложности систем увеличивается вероятность существования не выявленных ошибок. Главным образом, это происходит из-за проблемы отсутствия удовлетворительного для промышленности решения проблемы проверки правильности как программных систем, так и логических схем вычислительных устройств. Методы тестирования не обеспечивают исчерпывающего анализа всех возможных вариантов поведения систем, тем самым, не могут обнаружить все случаи нарушения свойств, которыми должны обладать разрабатываемые системы. Для систем со сложной моделью поведения становится невозможным обходиться без автоматизации проверки правильности.

Техника формальной верификации, получившая название проверки на модели (model checking), является одним из наиболее перспективных и широко используемых подходов к решению проблемы автоматизации проверки правильности программ. Так, в настоящее время существует довольно много систем верификации [1], наиболее популярны такие системы, как, SPIN [2], NuSMV [3], TLC [4, 5, 6], VeriSoft [7, 8].

Описываемые в данной работе средства автоматической верификации позволяют обнаружить такие свойства, как неполнота, противоречивость, недостижимость, использование неинициализированных атрибутов, выход за пределы допустимых значений, а так же нарушение динамических свойств, задаваемых пользователем. Верификация так же подразумевает генерацию контр-примеров, иллюстрирующих ошибки в поведении исходной модели.

К основным проблемам можно отнести как проблему интерфейсов, так и проблему комбинаторного взрыва достижимых состояний верифицируемой модели. Данная работа содержит 4 части. Первая часть посвящена интерфейсной проблеме. В ней описаны методы построения формальных спецификаций, а так же методы трансляции из языков описания поведения систем – MSC [9], SDL [10], UML [11]. Вторая часть посвящена описанию свойств верифицируемой модели, которые могут быть проверены данным программным комплексом. Третья и четвертая части посвящены проблеме комбинаторного взрыва – описаны методы оптимизации обхода поведения верифицируемой модели, а так же способы направления поиска и построение тестовых сценариев.

## Формализация требований и внутренний язык

**Методы построения формальных моделей.** Разные системы верификации используют различные формальные языки описания поведения и динамических свойств моделей. Верификатор CBMC, обеспечивает возможность ограниченной проверки моделей (Bounded Model Checker) для языков ANSI-C и C++. Он позволяет верифицировать выход за границу массива (переполнение буфера), безопасность указателей, исключения и пользовательские утверждения (user-specified assertions). Система SPIN имеет входной язык PROMELA. Система VeriSoft поддерживает языки C, C++, Java. Далее рассмотрены технологии построения формальных моделей с помощью сценарных спецификаций на языках MSC, SDL, UML; спецификаций, заданных в виде таблиц переходов; внедрения частей формального описания в текст исходной документации

посредством специальной разметки; а так же методы автоматической трансляции таких спецификаций в язык базовых протоколов.

**Язык базовых протоколов.** В качестве внутреннего языка формального описания моделей рассматривается язык базовых протоколов [12, 13] (БП). БП представляют собой выражения вида троек Хоара: $\alpha \rightarrow \langle u \rangle \beta$, где $\alpha$ – предусловие, $u$ – процесс (действие), $\beta$ – постусловие, и описывают атомарные переходы модели. Преимущество этого формализма в простоте и эффективности использования методов верификации – язык использует базовый набор структур данных и операций над ними, и при этом обладает достаточной выразительной мощностью для описания систем, специфицированных в таких инженерных языках, как MSC [14], SDL, UML [15]. Однако ручное описание спецификаций на языке БП затрудняет процесс построения больших моделей, так как одним БП описывается всего один (возможно, параметризированный) переход системы; возникает необходимость явного введения вспомогательных атрибутов, неявно заданных в исходном описании модели, таких как стек, поток управления, и др. Как следствие, повышается риск внесения ошибок, не присущих исходным спецификациям.

**MSC, SDL, UML.** Графические языки спецификаций MSC, SDL, UML – одни из наиболее известных методов формального описания поведения реактивных и распределенных систем [9–11], в настоящем широко применяются в процессе создания ПО. Трансляция в язык базовых протоколов описана в [14, 15].

**Формализация таблиц переходов.** К табличному методу описания переходов нередко прибегают на практике для описания сложного поведения (например, описание конечных автоматов). К преимуществам табличного описания можно отнести компактность и структурированность, а так же доступность для понимания, однозначную интерпретацию и упрощение инспекций. Действительно, сравнительно небольшой (для понимания, проверки, редактирования человеком) таблицей размера 10x10 задается 100 (как правило, систематизированных) переходов; однако работа над 100 описанными по отдельности переходами уже существенно затруднена и нетривиальна. При определенных условиях допустима автоматизация формализации таких требований. Для данной цели необходимо задать синтаксис описания и семантику (набор правил трансляции) таких переходов. Для обеспечения интеграции с другими формализмами (например, связь с конструкцией reference в MSC-сценарии), таблицу переходов может сопровождать некоторое пред- и постусловие [16].

**Разметка исходного текста спецификаций.** В условиях перманентного обновления и коррекции спецификаций стоит проблема синхронизации документации с ранее построенной формализацией. Элементы формальных нотаций предлагается вписывать в исходный текст спецификаций. Посредством специальной разметки (тегирования) таких элементов и правил трансляции тегов в язык БП, можно добиться автоматической генерации формальной модели непосредственно из документации, избежав при этом явного введения необходимых вспомогательных атрибутов, и таким образом достигнуть большей компактности и прозрачности описания. Помимо автоматизации, к преимуществам подхода можно отнести так же однозначную интерпретацию и трассировку требований, удобства синхронизации, версионного контроля и инспекций. На практике применение описанных методов позволяет уменьшить объемы и повысить качество описания формальных моделей, а так же существенно упростить процесс формализации. Автоматизированная трансляция спецификаций в формальную модель позволяет значительно сократить трудозатраты и сроки выполнения верификации. К недостаткам вышеописанных методов можно отнести невозможность создания универсальных правил трансляции [16].

## Проверяемые свойства формальных моделей

В основном системы верификации осуществляют проверку таких свойств: недетерминизм, выход за пределы допустимых значений, переполнение (over/underflow), обращение к неинициализированному элементу системы, а так же проверку условий безопасности, достижимости и отсутствия тупиков (deadlock, livelock). Всякий раз, когда достигнуто состояние, проверяемое на достижимость, либо нарушающее одно из вышеописанных свойств, будет построена трасса, иллюстрирующая поведение модели из ее начального состояния в текущее. Трасса будет записана на диск, если не исчерпано соответствующее ограничение на число записываемых трасс, определяемое пользователем. Под трассой понимается последовательность имен переходов модели (т.е. имен базовых протоколов с фактическим параметром). Трасса может быть построена из событий (процессов), определенных в базовых протоколах. Такие события описывают наблюдаемое поведение модели, (как правило, записаны на языке MSC) и могут быть использованы в качестве сценарных контр-примеров для тестирования.

**Недетерминизм (transition inconsistency)** в модели является не ошибкой ее поведения, а лишь сигналом предупреждения для дальнейшего анализа. Важно отделить естественный недетерминизм от ошибочного. Некоторые системы [4] верификации задают недетерминизм путем введения специальных атрибутов, значения которых выбираются недетерминировано в процессе моделирования.

**Выход за пределы допустимых значений** является распространенной ошибкой, часто являющейся источником нарушения безопасности систем. Проверка обеспечивается путем генерации (неявных) дополнительных проверок индексации в пред- и постусловиях переходов (обращение к элементу массива,

параметризированному атрибуту, либо при обращении к процессу по индексу). При нарушении взводится флаг OUT_OF_BOUNDS, при этом переход считается неосуществимым из данного состояния.

**Переполнение over/underflow** определяется таким образом: арифметические выражения, согласно приоритетам операций, раскладываются на эквивалентную последовательность вычислений значений бинарных операций, вводятся вспомогательные локальные переменные для хранения и дальнейшего использования результата предыдущих вычислений, и перед каждым производится проверка на возможное переполнение. В случае обнаружения взводится влаг OVERFLOW, при этом переход считается неосуществимым из данного состояния.

**Проверка использования неинициализированного элемента** модели обеспечивается посредством (неявного) введения вспомогательных атрибутов, цель которых хранить информацию об инициализации каждого атрибута модели, а так же генерации дополнительных проверок таких атрибутов. Если неинициализированный атрибут участвует в формуле предусловия перехода, либо в качестве индекса, взводится флаг UNINITIALIZED, при этом переход считается неосуществимым из данного состояния. Если производится присваивание, в правой части которого есть неинициализированный атрибут, то переход осуществляется, а атрибут, стоящий в левой части присваивания становится неинициализированным.

**Условия безопасности (safety, liveness)** выражаются LTL формулами. Примером может быть свойство «лифт никогда не едет с открытой дверью», «выделенный ресурс когда-нибудь освободится», и т.д.

**Проверка достижимости** выполняется двумя различными способами. Первый – проверка достижимости состояния (goal state), при этом целевое состояние формулируется пользователем в виде LTL формулы. Второй способ – проверка достижимости переходов, при окончании работы генерируется вердикт, включающий в себя информацию о статистике применения переходов.

**Тупиковой (deadlock)** называется ситуация, из которой нет ни одного допустимого перехода.

**Замкнутой (livelock)** называется ситуация, из которой нет возможности достигнуть хотя бы одно состояние, достижимое из начального (initial) состояния. Другими словами, условие отсутствия livelock можно сформулировать так: каждое достижимое состояние должно быть достижимо из любого изначально достижимого состояния. Проверяется так же, как и «сильная связность» в терминах теории графов.

## Ограничение обхода пространства поведения и оптимизация поиска

Рассмотрены методы оптимизации обхода пространства поведения верифицируемой модели, направления поиска и построения тестовых сценариев. Ядром разрабатываемой системы является классический поиск в глубину, с некоторыми специфическими конфигурациями и оптимизациями.

**Оптимизация 1:** создание хеш-таблиц переходов модели. Наиболее удачной хеш-функцией на практике является разбиение по принципу проверки потока управления процесса. Для каждого процесса такой атрибут должен явно сравниваться в предусловиях всех переходов и только с конкретными значениями; для простоты отделяется от общей формулы предусловия, и называется «ключевым состоянием». Таким образом, список «кандидатов» на проверку допустимости существенно сокращается. На практике трансляции UML стэйт-машин в язык базовых протоколов, описанная оптимизация на каждом состоянии сокращает выбор числа потенциальных переходов для каждого процесса в среднем с нескольких тысяч до одного-трех.

**Оптимизация 2:** элиминация интерливинга. Проектирование и тестирование параллельных систем усложняется тем, что компоненты могут взаимодействовать множеством зачастую непредусмотренных способов. Интерливинг (перестановка выполнения действий параллельно работающих компонент) является одним из источников комбинаторного взрыва вариантов поведения верифицируемой модели. Актуальной задачей является ограничение числа перестановок путем исключения незначимых с точки зрения проверяемых свойств. Параллельные системы состоят из множества асинхронно работающих сообщающихся между собой и внешней средой процессов. Точки сообщения (доступ к общим ресурсам) могут быть определены статически, основываясь на структурном синтаксическом описании верифицируемой модели. Аналогично [7, 8, 17], исследуя переходы, рассматриваются лишь частичные перестановки переходов. Заметим, что такой подход не справедлив для случая символьного моделирования, так как можно задать состояние (начальное, либо построить динамически в процессе обхода), включающее условие зависимости атрибутов разных процессов, при этом оставляя их синтаксически локальными.

**Оптимизация 3:** возврат к точке выбора ветви поведения. Если из некоторого состояния существует допустимый детерминированный переход, не нуждающийся в интерливинге с другими процессами, и таким образом не ветвящий пространство поведения верифицируемой модели, то такое состояние не является точкой выбора. Такое состояние не анализируется на предмет принадлежности множеству пройденных состояний и не сохраняется в нем. Более того, алгоритм обхода модифицируется таким образом, что при одном шаге назад восстанавливается состояние модели, являющееся точкой выбора (“choice point”). На практике верификации UML стэйт машин, такая оптимизация позволяет сократить число сохраненных пройденных состояний в среднем в 30 – 40 раз.

**Оптимизация 4:** выделение памяти исключительно для измененных атрибутов. При обходе сохраняется история изменения каждого элемента (атрибута) модели в отдельности. Каждое состояние текущей трассы хранит указатели только на измененные атрибуты. Таким образом, после осуществления очередного перехода

модели система выделит количество оперативной памяти, пропорциональное числу изменившихся по отношению к предыдущему состоянию атрибутов модели. На практике модель может содержать тысячи атрибутов, в то время как каждый переход в среднем меняет лишь 3 – 5 атрибутов. Заметим, в прочем, что такая оптимизация непригодна для случая символьного моделирования, так как предполагает хранение конкретных значений для каждого атрибута.

**Оптимизация 5:** перестановка экземпляров процессов при проверке эквивалентности состояний. Часто на практике встречаются модели, содержащие множество однотипных процессов, например, телекоммуникационные задачи, в которых участвуют N телефонов. Аналогично TLC [5, 6], функция проверки эквивалентности состояний модифицируется добавлением проверок дополнительных состояний, порожденных путем перестановок атрибутов модели по принципу их принадлежности к процессам, а так же хранению значения идентификаторов процессов, таким образом, ослабляя критерий эквивалентности.

**Оптимизация 6:** трансляция формализации в язык C. Описание среды, пред- и постусловий переходов, а так же функции проверки необходимости интерливинга, функции, порождающей перестановки атрибутов для ослабленной эквивалентности, транслируются в C-функции и компилируются вместе с ядром верификатора в бинарный исполняемый код. На практике верификации UML модели, содержащей ~16.000 переходов и ~10.000 атрибутов, такая трансляция длится ~20 минут (транслятор реализован на языке высокого уровня APLAN с использованием технологии переписывания термов [18]), компиляция gcc длится не более 3х минут. Предварительная трансляция формального описания модели в язык C повышает производительность обхода пространства поведения в десятки тысяч раз по сравнению с методами, основанными на интерпретации.

## Направленный поиск и создание тестовых сценариев

Основная проблема верификации – комбинаторный взрыв достижимых состояний верифицируемой модели. Актуальной так же является задача построения тестовых сценариев, удовлетворяющих некоторому критерию покрытия. Примером является ситуация, в которой, возвращаясь из некоторого состояния при поиске в глубину, стоит вопрос – сохранять текущую трассу на диск или нет? Очевидно, что сохранить на диск все проанализированные трассы невозможно в силу их количества; подбор же эвристик не решит задачу покрытия, так как для разных моделей формулируются различные критерии.

Одним из подходов к решению проблемы является привлечение пользователя (эксперта предметной области) к процессу верификации. Большинство из верификационных систем использует заданные пользователем абстракции [19] и специальные состояния, описанные в виде формул темпоральной логики, в качестве критерия отсечения ветвей поведения («hints», «restricted states») [20–22]. Однако высокий уровень необходимых знаний в области математической логики, предъявляемый пользователю, часто является существенным ограничивающим фактором, усложняющим использование формальных методов в процессе индустриальной разработки программного обеспечения.

Данная работа предлагает расширение способов управления путем задания регулярных выражений над алфавитом имен параметризированных переходов модели (далее образцов), и стратегий – правил для динамического управления ограничениями [23]. Правила представляют собой условия для выполнения таких инструкций, как: включение/выключение/исключение множества переходов и определение приоритетов процессов и переходов, переключение в интерактивный режим, а так же изменение таких ограничений пространства поведения, как длина трассы и других параметров.

Как правило, события формальной модели ассоциируются с именами ее переходов, поэтому семантически такие регулярные выражения задают «контрольные точки» поведения модели и могут так же определять критерий выбора построенных трасс (вариантов конкретного поведения системы) для последующего их использования в качестве тестовых сценариев. Так, предполагаемые поведения моделируемой системы, описанные пользователем, проверяются на допустимость, одновременно накладывая ограничения на поиск. Для различных видов регулярных выражений существуют алгоритмы различной вычислительной сложности [24, 25]. Обход пространства поведения модифицируется, принимая во внимание приоритетность переходов, путем добавления вызова специальной функции, реализующей динамический множественный поиск образцов в текущей трассе для каждого нового состояния верифицируемой модели. Иными словами, порождается параллельно работающий конечный автомат, хранящий информацию об истории переходов и определяющий критерии для отсечения ветвей поведения модели. Если текущая трасса удовлетворяет образцу, то она будет сохранена на жесткий диск, если не исчерпан соответствующий лимит (пользователь определяет нужное количество трасс по каждому образцу). Если лимит исчерпан, образец исключается из списка актуальных. Ситуация, из которой любое продолжение текущей трассы не приведет обнаружению вхождения очередного символа (т.е. имени перехода) хотя бы одного актуального образца, является дополнительным критерием завершения ее генерации. При этом, если трасса содержит исторически максимальной длины префикс хотя бы одного образца, она будет сохранена на диск с соответствующей пометкой. Таким образом, если полного вхождения какого-либо образца не обнаружено, будет построена трасса, удовлетворяющая максимальному префиксу этого образца. Дополнительным для сохранения трассы является условие вхождения в трассу перехода, не входящего ни в одну из уже сохраненных трасс. Таким образом, будет построен набор трасс, включающий как предполагаемые пользователем поведения модели, так и все достигнутые переходы модели.

Динамическое переопределение приоритетов и исключение множества переходов может быть эффективно использовано для ограничения основного источника комбинаторного взрыва вариантов поведения – недетерминизма и интерливинга. Ужесточение таких ограничителей, как максимально допустимая длина предшествующего образцу префикса трассы и дистанции между контрольными точками, позволяет значительно сократить время верификации и количество трасс для обеспечения необходимого покрытия. На практике использование таких простейших правил, как ассоциация статистики применения перехода с его приоритетом (т.е. переход или процесс считается приоритетнее, если он срабатывал реже всех) позволяет сократить время эксперимента и добиться при этом покрытия ранее недостигнутых переходов.

## Выводы

Описанный программный комплекс осуществляет проверку всех наиболее распространенных свойств моделей, при этом выявленные случаи нарушения сопровождаются контр-примерами. Производительность с учетом оптимизаций 1, 3, 4, 6 возросла в десятки тысяч раз, и составила ~300.000 состояний в сек. на платформе Pentium IV 3.0 GHz с 1GB оперативной памяти. Применение технологии направленного поиска на индустриальном телекоммуникационном проекте, состоящем из ~1000 переходов дало возможность не только выявить ошибки в спецификациях, но и создать набор тестовых сценариев (~100 трасс), обеспечивающих необходимое функциональное покрытие.


1. http://en.wikipedia.org/wiki/Model_checking
2. *Ben-Ari. M.* Principles of Spin. // Springer Verlag. – 2008. – P. 216.
3. *Cimatti A., Clarke E. M., Giunchiglia E., and others.* NuSMV 2: An OpenSource Tool for Symbolic Model Checking // In Proceeding of International Conf. on Computer-Aided Verification, Copenhagen, Denmark. – 2002. – P. 359–364.
4. *Yu Y., Manolios P., Lamport L.* Model Checking TLA+ Specifications – 1999. – P. 54–66.
5. *Ip C., Dill D.* Verifying Systems with Replicated Components in Murphi // International Conf. on Computer-Aided Verification. – 1996. – P. 147–158.
6. *Ip C., Dill D.* Better Verification through Symmetry // Formal Methods in System Design. – 1996. – Vol. 9. – P. 41–75.
7. *Godefroid P.* Partial-order methods for the verification of concurrent systems – an approach to the state-explosion problem // Lecture notes in computer science, Springer-Verlag. – 1996. – Vol. 1032. – P. 143.
8. *Godefroid P.* Software model checking: the VeriSoft approach // Formal methods in system design, Springer science, Netherlands. – 2005. – Vol. 26. – P. 77–101.
9. ITU-T Recommendation Z.120 – Message Sequence Chart (MSC), ITU. – 1996.
10. *ITU*-T Recommendation Z.100 – Specification and Description Language (SDL), ITU. – 2002.
11. Unified Modeling Language: Superstructure version 2.0, Object Management Group (OMG). – 2003.
12. *Летичевский А.А., Капитонова Ю.В., Волков В.А., и др.*Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ. – 2005. – № 4. – С. 3–21.
13. *Baranov S.N., Kapitonova J.K., Kolchin A.V., and others.* Tools for Requirements Capturing Based on the Technology of Basic Protocols // Proc. of St.Petersburg IEEE Chapter. – 2005. – P. 92–97.
14. *Потиенко С.В., Колчин А.В.* Трансляция MSC сценариев в язык базовых протоколов // Искусственный интеллект. – 2007. – № 3. – С. 428–435.
15. *Потиенко С.В., Колчин А.В.* Представление SDL-спецификаций в виде базовых протоколов // Искусственный интеллект. – 2006. – № 4. – С. 42–52.
16. *Колчин А.В.* Альтернативные способы формализации требований // Тези доп. міжнар. конф. “Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2006”. – К. НаУКМА, Національний ун-т ім. Т.Г. Шевченка, ін-т програмних систем НАН України. – 2006. – С. 111–115.
17. *Потиенко С.В.* О проблеме интерливинга в верификации формальных моделей // Тези доп. міжнар. конф. "Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2007". – Бердянськ. НаУКМА, національний ун-т ім. Т.Г. Шевченка, ін-т програмних систем НАН України. –2007. – С. 69–72.
18. *Letichevsky A.A., Kapitonova J.V., Konozenko S.V.* Computations in APS // Theoretical Computer Science. – 1993. – Vol. 119. – P. 145–171.
19. *Clarke E.* Model checking and abstraction. // In ACM Transactions on programming languages and systems. –1994. – Vol. 16. – №.5. – P. 1512–1542.
20. *Bloem R., Ravi K., and Somezi F.* Symbolic guided search for CTL model checking // In Design Automation Conference. – 2004. – P. 29–34.
21. *Peranandam P., Weiss R., Ruf J., Kropf T. and Rosenstiel W.* Dynamic guiding of bounded property checking // In IEEE International High Level Design Validation and Test Workshop. – 2004. – P. 15–18.
22. *Barner S., Glazberg Z., and Rabinovitz I.* Wolf – bug hunter for concurrent software using formal methods // In Computer Aided Verification. – 2005. – P. 153–157.
23. *Колчин А.В.* Направленный поиск в верификации формальных моделей // Тези доп. міжнар. конф. “Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2007”. – Бердянськ. НаУКМА, Національний ун-т ім. Т.Г. Шевченка, ін-т програмних систем НАН України. – 2007. – С. 256–258.
24. *Aho A.* Algorithms for finding patterns in strings // Handbook for theoretical computer science, MIT Press. – 1990. – Vol.A. – P. 257–300.
25. *Smyth B.* Computing Patterns in Strings. // ACM Press Books. – 2003. – P. 440.