УДК 004.052.42

И.А. Федотов. аспирант,
ivan.fedotov@phystech.edu
А.С. Хританков, к.ф.-м.н., доцент
anton.khritankov@acm.org
Московский физико-технический институт, Москва.

Систематический обзор исследований в области автоматической верификации кода смарт-контрактов.

Смарт-контракты - это обладающий спецификой программный код, исполняемый в системах распределенного реестра (блокчейн платформах). Смарт-контракты все чаще применяются в финансовой, юридической и других сферах, требующих высокой надежности и безопасности. Автоматическая верификация и анализ кода - важнейшее направление в обеспечении надежности и безопасности программ. В данной работе представлен краткий систематический обзор исследований в области верификации смарт-контрактов в период 2015-2019 годы.

Ключевые слова. Смарт-контракт, блокчейн, верификация, систематический обзор, Ethereum, BitCoin, проверка моделей.

1. Введение

Проблема устранения ошибок при разработке и использовании программ является одной из ключевых в области компьютерных наук. Ошибка может появиться в явном виде при работе программы в режиме промышленной эксплуатации. Впоследствии она может быть использована злоумышленниками. Особенно опасна такая ситуация при разработке и сопровождении финансовых приложений, когда от корректности работы кода зависит трансфер средств.

Технология распределенных реестров блокчейн (blockchain) и децентрализованные приложения (dApps), несмотря на новизну, стремительно набирают популярность при создании финансовых приложений, средств

электронного голосования, энергетике, медицине, интернете вещей [1]. Программный код смарт-контракта, исполняемый на платформе блокчейн, реализует алгоритм, в соответствии с которым финансовые средства или другие важные данные передаются от одного пользователя другому. При этом общепринятых средств для проверки и отладки программного кода пока еще не существует. С другой стороны, растет число создаваемых смарт-контрактов [2], соответственно проблема выявления и устранения ошибок становится все более актуальной.

Существует и другая особенность, которая делает ошибки в смарт-контрактах наиболее критичными. Зачастую смарт-контракты после развертывания на распределенном реестре остаются неизменными. Это обстоятельство не позволяет исправить ошибки во время работы и, следовательно, основное значение имеет выявление ошибок на уровне разработки. Уязвимости в смарт-контрактах уже приводили к атакам, которые подрывают доверие к технологии распределенного реестра. Например, печально известная DAO атака [3] привела к потере почти 50 миллионов долларов США. Некоторые другие атаки привели к существенной потери стоимости виртуальной валюты Ether (эфир), включая ошибку Parity Wallet, результатом которой было то, что эфир на 169 миллионов долларов оказался навсегда заблокирован [4]. Эти и многие другие примеры являются показателем актуальности исследования верификации кода смарт-контрактов.

Данное исследование представляет систематический обзор в области автоматической верификации кода смарт-контрактов. Его целью является исследование текущих тенденций и методов верификации программного кода смарт-контрактов, их классификация и анализ. Обзор определяет также существующие инструментальные средства (tools, программные средства), позволяющие проводить верификацию программного кода смарт-контрактов.

Данный обзор отличен от других [5] тем, что в нем исследуется не только вопрос безопасности системы распределенного реестра, но, в частности,

рассматриваются методы верификации смарт-контрактов и соответствующие им инструментальные (программные) средства.

Обзор структурирован следующим образом. В следующем разделе приведено описание технологии распределенного реестра и принцип работы смарт-контрактов. Далее приведена постановка задачи исследования и определены методы исследования. Затем представлена характеристика современных методов верификации программного обеспечения. В четвертом разделе представлен систематический обзор литературы, существующих на данный момент технических, а также инструментальных средств для верификации смарт-контрактов. Исследование завершается анализом и систематизацией рассмотренных статей и выводами о направлениях дальнейшего исследования.

2. Технология распределенного реестра и смарт-контрактов

Технология распределённого реестра. Блокчейн - это выстроенная цепочка блоков, при этом каждый б лок содержит информацию, в том числе и о предыдущем блоке [6]. Ядром блокчейна является распределённое одноранговое (peer-to-peer) хранилище, доступное только для записи значений, но не для их изменений. Данные в хранилище обновляются только посредством консенсуса участников сети, имеющей выход в сеть интернет, как это представлено на рис. 1.

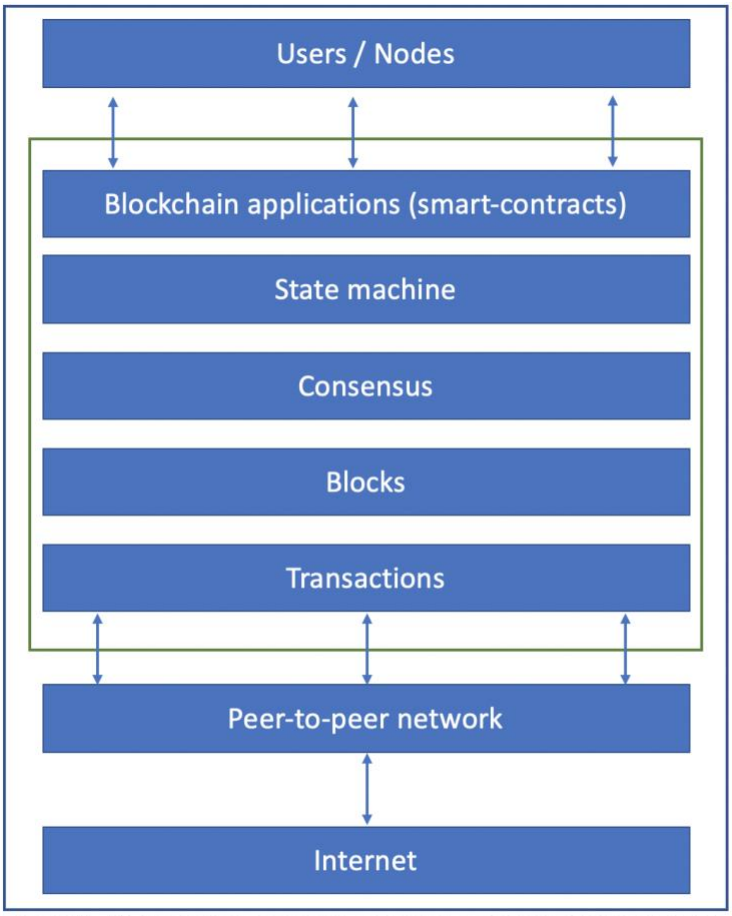


Рисунок 1. Высокоуровневая схема распределенного реестра.

Основой технологии является множество протоколов и криптографических методов, применяемых к сети узлов, которые взаимодействуют с целью обеспечения безопасной записи и хранения данных в распределенном хранилище.

Все изменения в сети, совершаемые посредством одной транзакции, помещаются в соответствующий блок, который можно рассматривать как страницу в памяти. Блок может быть представлен как последовательность транзакций, собранных вместе, в порядке логической организации. Структура блока зависит от архитектуры блокчейна, но в целом заключает в себе набор атрибутов, необходимых для его корректного функционирования, такие как заголовок блока, указатель на предыдущий блок, временные метки, счетчик транзакций и сами транзакции. Каждый блок данных делегируется подтверждающим узлам (майнерам). Подтверждение происходит с помощью технологии, принятой в платформе - это может быть решение криптографической задачи, либо майнеры получают априорное право подтверждать блок по определенным критериям. Такими критериями могут выступать временной штамп, обозначающий, как долго майнер участвует в консенсусе, либо количество виртуальной валюты на аккаунте майнера, либо майнеры могут назначаться выборным путем [7].

Смарт-контракты. Смарт-контракты инкапсулируют программный код, который отображает контрактные соглашения в реальном мире на блокчейн платформы [8]. Ключевой особенностью смарт-контрактов является то, что они автоматизируют выполнение соглашений между двумя и более участниками, без привлечения третьей заверяющей стороны за счет невозможности изменения порядка исполнения контракта одной из сторон после его публикации. Программный код смарт-контрактов размещается на всех подтверждающих узлах распределенного реестра.

Остановимся подробнее на рассмотрении контрактных языков [9]. Принцип их работы проиллюстрирован на рис. 2. Компиляция (А) подразумевает промежуточный уровень, который позволяет выполнить оптимизацию

программы и верификацию. Массовое распространение смарт-контракты получили недавно, и на ранних этапах компиляция происходила без промежуточного представления (B). Язык Bitcoin Script требует написание низкоуровневого кода. Примерами верхнеуровневых языков могут служить Solidity [10] и Vyper [11].

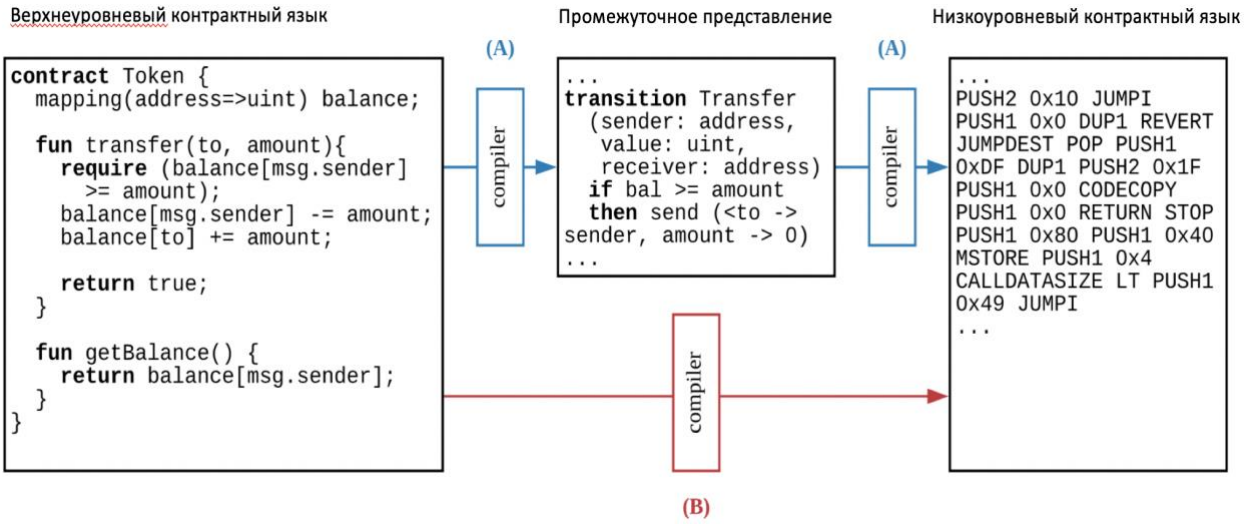


Рис. 2. Различные уровни контрактных языков.

Наибольшее распространение смарт-контракты получили на платформах типа Ethereum [6]. Для их запуска используется виртуальная машина Ethereum (Ethereum Virtual Machine, EVM) [12], что делает возможным написание смарт-контрактов на разных языках, для которых реализован транслятор в исполняемый байт-код для виртуальной машины. Самым распространённым языком с поддержкой трансляции в байт-код EVM является Solidity.

Смарт-контракты на платформе Ethereum должны удовлетворять определенным стандартам. Стандарт ERC20 [13] определяет набор правил для токенов Ethereum, что позволяет формализовать поведение токенов в экосистеме Ethereum. Практически это означает, что смарт-контракт должен реализовывать ряд функций, чтобы удовлетворять стандарту.

3. Методология исследования

Исследование проведено согласно методу систематического обзора [14].

Стоит добавить, что на ошибки исполнения программного кода смарт-контрактов также влияет сторонние факторы, такие как внешние библиотеки,

работа операционной системы, а также виртуальной машины. Предполагается, что все это работает, и ошибки вышеперечисленных факторов не учитываются.

Постановка вопросов исследования. Цель исследования - получить ответы на следующие вопросы:

Q1. Какие методологии верификации кода используются?

Q2. Какие результаты известны по автоматической верификации смарт-контрактов?

Q3. Какие есть ограничения в автоматической верификации программного кода?

Q4. Какие можно выделить будущие направления в автоматической верификации смарт-контрактов?

Протокол поиска. Поиск производился в августе-сентябре 2019 года. Для поиска статей использовались следующие поисковые системы: IEEE Xplore, Scopus, Google Scholar, Research Gate, ScienceDirect. Для проведения поиска ключевые слова были объединены в поисковые строки:

- (blockchain OR ethereum OR bitcoin) AND smart-contract AND (verification OR model-checking OR validation) OR systematic review
- smart contract AND verification tools

Для статей, прошедших поисковый фильтр по вышеперечисленным запросам, а также упомянутых в них исследований был произведён ручной поиск, чтобы выявить статьи, которые косвенно относятся к тематике, но могут иметь существенное влияние на область (неопубликованные статьи, дипломные работы, коммерческие документы, учебные материалы). В исследовании, направленном на построение качественного систематического обзора [14], также подчеркнута важность используемых методик.

После выполнения поисковых запросов был произведён анализ статей по названию, аннотации, введению, заключению, количеству цитирований и году публикаций. Далее приведены критерии включения и критерии исключения, которые были использованы при анализе и выявления наиболее подходящих статей.

Критерии включения.

- Исследование проведено на русском или английском языке.
- Исследование относится к области компьютерных наук.
- Исследование проведено в последние 4 года. Исключением могут быть фундаментальные учебные материалы.
- Исследования, представленные в виду статьи, опубликованы в рецензируемых журналах.

Критерии исключения.

- Основной язык не является русским или английским.
- Исследование нетехнического характера.

В процессе поиска было изучено более 100 статей, коммерческих документов, документации к инструментальным средствам и учебных пособий. В результате в обзор попало 52 исследования.

4. Краткая характеристика современных методов верификации программ

Обратимся сначала к вопросу Q1. В основе **формальной верификаций (formal verification)** лежат формальные методы, используемые для доказательства соответствия модели системы установленным спецификациям требований. Необходимо понимать, что фактически реализованная программа и требования к ней будут отличаться в той степени, в которой отличаются соответствующие им модели. Методы формальной верификации могут различаться как по модели программ (конечные автоматы, сети Петри, размеченные системы переходов), так по и модели требований (программные контракты, продукционные правила, темпоральные утверждения).

Метод проверки эквивалентности предполагает совпадение типа модели программы с типом модели спецификаций. Для сравнения моделей используется отношение эквивалентности, определённое для данного типа модели.

Другой метод формальной верификации называется **проверкой моделей (model checking)** [15]. В данном методе спецификации представляются логическими формулами, а программный код - структурами,

интерпретирующими эти формулы. Под формальным соответствием подразумевается, что структура, представляющая код, является моделью соответствующей формулы. Для представления спецификаций может использоваться логика LTL (Linear-time Temporal Logic - LTL) [16], которая представляет программу как последовательность действий во времени. Такая логика позволяет представлять формулы в контексте будущего развития. Например, условие будет верным, если выполнится ряд предшествующих условий. С помощью такого подхода можно рассматривать действие программы или алгоритма во времени.

В методе **дедуктивного анализа** [17] модель программы - это запись на языке с формализованной семантикой (может быть представлена в виде исходного кода, псевдокода или схемы программы), а модель требований - совокупность программных контрактов для модулей программы. Здесь под программным контрактом понимается совокупность логических формул: пред- и постусловия для функций, процедур, методов. Предусловия обеспечивают выполнение всех условий, при которых программа будет запущена правильно. Постусловия указывают корректный результат выполнения самой подпрограммы, то есть что все выходные переменные и состояние программы после исполнения будут удовлетворять этим условиям. Метод дедуктивного анализа активно применяется для основных языков программирования, таких как C [18], Java [19] или C# [20]. Верификация происходит на уровне исходного кода программы. Инструментальное средство KeY [21] было разработан для дедуктивной верификации объектно-ориентированных языков. В стандартной версии KeY принимает на вход код с расширением .java и спецификации в виде JML (Java Modeling Language).

Гибридная верификация сочетает в себе формальные методы верификации и тестирование. Рассмотрим случай, в котором язык не имеет строгой семантики, и программа на нем не является одновременно своей моделью для верификации. Тогда одна лишь математическая проверка моделей не может полностью обеспечить отсутствие ошибок в программном коде, так как

модели отличаются от реальных программ и требований. С другой стороны, тестирование кода происходит уже после написания программного кода, а значит при появлении ошибок их нужно будет исправлять в уже разработанной программе. Для наиболее эффективного исключения ошибок формальные методы используются вместе с тестированием, и такой подход называется **методом тестирования на основе моделей**.

Выявление ошибок можно производить и без доступа к исходному коду – с помощью **журнала протоколирования.** Вся информация о работе программы записывается в журнал, в том числе и путь к ошибке с возможной причиной. При аварийном завершении программы разработчик может отследить ошибку в журнале протоколирования и выявить ее причину [22]. Отладка происходит путем повторения исполнения программы в тестовой среде.

5. Применимость методов верификации к смарт-контрактам

Обратимся к вопросу Q2 и укажем, какие из подходов, относящихся к вопросу Q1 применимы в области смарт-контрактов.

Статический анализ кода. Рассмотрим инструментальные средства статического анализа кода смарт-контрактов [23]. Средства SmartCheck [24] выполняет разбор текста исходного кода программы и представляет абстрактное синтаксическое дерево (AST) в виде документа XML с сохранением трассировки по номерам строк с исходным кодом. Строится дерево состояний смарт-контракта и делается разбор схемы запросами XPath. SmartCheck включает набор правил для проверки соответствия полученного дерева рекомендациям и ограничениям языка Solidity. Исследование [25] показало, что SmartCheck среди других инструментальных средств статической верификации является наиболее эффективным по количеству найденных ошибок, но он может также выдавать ложные предупреждения.

Статический анализ кода также интегрирован в некоторые среды разработки, например Intellij Idea Solidity plugin. В приведенных инструментальных средствах учитываются синтаксические ошибки в коде смарт-контрактов.

Формальная верификация. Рассмотрим методы формальной верификации смарт-контрактов. Спецификации могут передаваться на специально созданном для реализации таких методов языке.

Инструментарий VeriSol [26] для платформы Azure Blockchain, комбинирует подходы на основе семантического анализа кода и тестирования: программный код транслируется в формальный язык Boogie [27], происходит семантический анализ кода, который определяет соответствие реализации кода спецификации интерфейса по пред- и пост-условиям. Примером выявляемых ошибок может служить ошибка в реализации интерфейса, при которой снятие средств со счета может произойти несколько раз до обновления баланса. Результатом подобной ошибки была кража 80 миллионов долларов злоумышленником [28]. После анализа в код Solidity внедряются модификаторы с проверками пред- и постусловий функции.

В другой методике [29] выполняется трансляция программного кода смарт-контракта в функциональный язык F*, предназначенный для верификации программ, после чего происходит верификация функциональной корректности спецификаций. В дополнение к этому байт-код декомпилируется и выполняются низкоуровневые проверки, такие как проверка достаточного количества эфира при вызовах.

Инструментальное средство SOLAR [30] проверяет соответствие смарт-контрактов стандарту с помощью символьного исполнения (symbolic execution) и пересчета заданных на python-подобном языке ограничений и инвариантов совместно с исполнением кода смарт-контракта (concolic execution) [31, 32]. Написав один программный контракт (стандарт), можно сделать проверку всех смарт-контрактов, которые должны ему удовлетворять. Для безопасной проверки корректности поведения смарт-контракта средство SOLAR моделирует поведение смарт-контракта и запускает его в тестовой среде с пробными транзакциями. Такой инструментарий примечателен тем, что при поиске логических ошибок он выдает существенно меньше ошибок первого рода (false positive), чем другие аналогичного назначения инструментальные средства.

Упомянем также специальный язык и программный каркас (framework) [33] описания одновременного обращения нескольких участников к смарт-контракту. Работа каркаса основана на теории игр. Каждый участник, который получает доступ к смарт-контракту, рассматривается как игрок, причем количество игроков и вызовов функций в смарт-контракте ограничено. Происходит трансляция контрактов в одновременные игры (simultaneous game), где передвижение средств в контракте сопоставляется с действием в игре. Анализ игры помогает определить возможные состояния для смарт-контракта и выявить логические ошибки, из-за которых средства могут быть перечислены коррумпированному участнику контракта.

Интересно рассмотреть методы верификации, основанные на SMT (Satisfiability Modulo Theories) технике [34]. Решающие устройства SMT используются для доказательства выполнимости формул в различных логиках, которые могут быть использованы для моделирования работы программ. Контракты Solidity могут быть транслированы в SMT формулы [35]. Целью трансляции является верификация свойств программ путем выполнения запросов в решающее устройство SMT.

В исследовании [36] предлагается использование инструментария SPIN [37] для верификации смарт-контрактов. Поведение контракта транслируется в язык Promela (Process Meta Language), после чего происходит верификация кода Promela в соответствии с представленными свойствами ранее разработанными инструментами. Верифицируемые системы представляются на языке Promela, поддерживающим моделирование асинхронных распределенных алгоритмов как недетерминированных автоматов. Получив модель и определение проверяемых свойств, SPIN генерирует программу на языке Си, которая решает конкретную задачу. Свойства выражаются в виде формул линейной временной логики (Linear temporal logic, LTL).

Инструментарий NuSMV [38], использует похожий подход для выражения свойств, которые необходимо проверить на модели: вычислительное дерево логики (Computation tree logic, CTL). В модели CTL логика представлена

древовидной структурой, в которой каждая ветвь может быть будущим развитием событий при выполнении определенных условий. При этом, если выполнены требования, все возможные ветви исполнения программы позволяют избежать нежелательных условий (например, деление на ноль или другие ошибки). Модель написана на специальном языке для NuSMV, а свойства для проверки формализованы в CTL.

Примеры использования логических формул, таких как LTL и CTL, в процессе верификации смарт-контрактов можно найти в соответствующих статьях [37], [38].

Модель рассматривается на трех уровнях: поведение всего реестра на платформе Ethereum, поведение смарт-контракта и его последующая трансляция в язык NuSMV, поведение исполняемого каркаса в соответствии с переданными свойствами. Если условие не выполняется, генерируется контрпример, на основе которого можно проанализировать природу выявленного дефекта. Метод был разработан в ходе исследования применения распределенного реестра на энергетическом рынке Франции.

Дедуктивный анализ. Для смарт-контрактов представлено расширение SolidiKeY [39], в котором инструмент KeY [21] адаптирован для верификации кода смарт-контрактов. Используется метод дедуктивной верификации, который выполняется автоматически с помощью инструментальных средств, реализующих доказательства теорем. Происходит проверка на то, что пост- и предусловия соответствуют поведению смарт-контракта. Примером предусловия для смарт-контракта может быть наличие определенной суммы для транзакции, которую инициирует смарт-контракт. Соответствующем постусловием будет являться начисление суммы на аккаунт пользователя. На выходе инструментарий выдает информацию о том, удовлетворяет ли программный код спецификациями, и если не удовлетворяет, то - по каким пунктам.

В работе [39] предлагается язык спецификаций для Solidity, учитывающий его особенности, а также возможность верифицировать программный код

Solidity в соответствии с переданными спецификациями. Результатом работы SolidiKeY является информация о соответствии программного кода смарт-контракта спецификациям.

Проверка доказательств теорем. В [40], [41] представлен подход верификации с помощью инструментальных средств проверки доказательств Isabelle/HOL. Байт-код представляется в виде линейной последовательности блоков программы. Конструкция байт-кода может содержать скачки, которые ведут поток выполнения к другой части программы, что затрудняет представление в виде последовательного выполнения кода. В связи с этим обстоятельством, декомпиляция происходит с помощью метода контрольных графов (Control Flow Graphs, CFG), который позволяет представить программу в виде базовых блоков и соединить их ребрами, которые соответствуют скачкам. Таким образом, в соответствии с теоремой о структурном программировании [42] программный код будет представлен в виде последовательности подпрограмм. Спецификации передаются в виде псевдокода, после составленная последовательность проверяется в соответствии со спецификациями в инструменте Isabelle/HOL.

В инструментарии FEther [43], использующем автоматическое средство доказательства теорем Coq [44], применяется кэширование уже проверенного кода и, при последующем попадании на него, повторная проверка не требуется.

Каркас VeriSolid [45] не требует кода смарт-контракта. Нужна только его модель, а также список верифицируемых свойств, что позволяет верифицировать поведение контракта на высоком уровне абстракции. Спецификации могут быть записаны на естественном языке, например с помощью шаблонов. Принцип работы такого инструментария следующий: сначала строится поведенческая модель контракта (Behaviour-Interaction-Priority - BIP). В ней рассматриваются только возможные состояния смарт-контракта - это позволяет уменьшить нагрузку при последующей верификации. Из переданных спецификаций строится вычислительное дерево логики CTL. После формальной верификации модели CTL с помощью уже имеющихся инструментальных средств [46], [47]

может быть сгенерирован код смарт-контракта на языке Solidity. Такой процесс соответствует разработке по принципу “корректность в соответствии с дизайном” (correct by design) [48]. При обнаружении ошибок вместо генерации кода пользователю передается вывод спецификации, в котором указаны ошибки проверки.

Интересным представляется гибридный подход, в котором формальные методы сочетаются с другими методами верификации, такими как тестирование или статический анализ, что обеспечивает большее покрытие кода проверками. Для формальной проверки в инструментальном средстве верификации FSPVM-E [49], в котором реализован гибридный подход к верификации программного кода, используется средство автоматического доказательства теорем Coq и рассмотренный ранее инструментарий FEther.

Динамическая верификация и анализ журналов работы. Методы динамической верификации также применимы к смарт-контрактам. Метод проверки корректности работы смарт-контрактов постфактум, то есть уже после записи в реестр, в одной из работ [50] был обозначен как “query blockchain”. После развертывания контракта в сети, делается запрос на корректность записанных в реестр данных, после чего происходит автоматическая проверка соответствия смарт-контракта и записанных данных.

Другой метод, реализованный в инструментарии ContractFuzzer [51], подразумевает генерацию данных для смарт-контракта и анализ журнала протоколирования в тестовой среде. Таким образом при промышленном использовании смарт-контракта ряд ошибок уже будет учтен.

Наиболее часто встречающимся инструментальным средством динамической верификации смарт-контрактов при составлении обзора был ContractLarva [52]. На вход этому средству передаются спецификации, после чего происходит мониторинг событий смарт-контракта. Код контракта разбирается и в него внедряются проверки на события из спецификации. В базовой версии этого средства предусмотрена возможность взимания со всех участников консенсуса априорной платы, которая будет внесена в случае

нарушения смарт-контракта со стороны определенного пользователя. Другим вариантом является полная остановка смарт-контракта без вычета средств с пользователя. Также имеется ряд наработок по совершенствованию инструментальных механизмов ContractLarva, которые относятся к вопросам устранения последствия нарушений смарт-контракта путем генерирования и развертывания вторичного смарт-контракта [53].

Представляет интерес подход, направленный на автоматическое восстановление спецификаций на основе анализа истории исполнения контракта в распределенном реестре [54]. Спецификации подходят как для самого контракта, так и для межконтрактного взаимодействия. При анализе работы программного кода строится граф зависимостей контрактов и состояний. Полученный результат может быть использован как для выявления ошибок, так и для построения более точной верификации.

Работа [55] посвящена моделированию межконтрактного взаимодействия с помощью анализа траекторий перевода эфира и предсказания связей смарт-контрактов в некотором тензорном пространстве. В этом пространстве одно измерение связано с аккаунтом отправителя, второе с аккаунтом получателя средств, а третье связано со временем. В пересечении в конкретном временном слоте между аккаунтом отправителя и получателя указывается количество переданного эфира. Для полученного тензора выполняется декомпозиция CANDECOMP/PARAFAC (CP) [56]. По полученным данным в приведенном трехмерном пространстве строится картина распределения эфира во времени между участниками, после чего с помощью временных рядов моделируется поведение в будущем. На основе этого анализа можно выявить нарушения в работе системы, а в случае ошибки может быть сгенерирован контрпример.

Аудит и экспертиза. Рассмотрим еще один метод в верификации смарт-контрактов - аудит с применением автоматического анализа кода. Инструмент аудита смарт-контрактов S-gram [57] для платформы Ethereum сочетает методы статического анализа кода и обнаружения аномалий. На этапе обучения S-gram по заданной коллекции (корпусу) смарт-контрактов для кода после разделения

на токены, дополненные информацией от статического анализатора кода, создается статистическая языковая модель (statistical language model) [58]. Такая языковая модель представляет собой конечный автомат с частотными вероятностями переходов от одного токена к следующему. На основе языковой модели строятся взаимосвязи между критическими операциями и условиями их выполнения (например, трансфер средств и проверка адреса отправителя). На этапе анализа с помощью языковой модели рассчитывается, насколько типичен исследуемый смарт-контракт или его методы относительно корпуса, и если нетипичен - передается на дальнейшую проверку. Работа S-gram была проверена на более чем 1500 реальных контрактов, было найдено 95% всех уязвимостей.

Техническая экспертиза может применяться для выявления соответствия смарт-контракта стандартам (например, ERC20 [59]).

Поддержка других языков. Одним из перспективных направлений исследования является верификация смарт-контрактов для менее распространенных языков [9], [60]. Как отмечено выше, Solidity - контрактно-ориентированный язык, в то время как для разработки распределенных приложений созданы процедурные языки (Scilla [61], Bamboo[62]), в которых функцию можно сделать атомарной. Более того, перечисленные языки позволяют разделить контракт на несколько составляющих, по сути независимых контрактов, которые, однако, имеют один и тот же адрес и в реестре являются одним целым. Такой подход позволяет представить один контракт в виде переходов между независимыми состояниями. Исследования в верификации альтернативных языков уже ведутся [62], однако результатов, позволяющих использовать инструменты в промышленной эксплуатации, пока нет.

6. Анализ результатов исследования

В рамках данного раздела анализируются текущие, а также рассмотрены перспективные направления в верификации смарт-контрактов. Таким образом будут даны ответы на исследовательские вопросы Q3, Q4.

Исходя из анализа содержания большого числа статей, можно сделать вывод, что наибольшее распространение получили методы формальной верификации. Это может быть связано с тем обстоятельством, что при верификации смарт-контрактов устранение вероятных ошибок важнее скорости проведения верификации. Основная причина в том, что формальные методы используют строгий математический аппарат, процент выявленных ошибок оказывается выше, нежели при тестировании, при динамическом или статическом анализе. Однако, это не означает, что последними тремя методами можно пренебрегать. Как было отмечено выше, эти методы позволяют выявить ошибки, которые могут быть упущены на этапе применения формальных методов верификации.

Среди методов верификации смарт-контрактов следует выделить статический анализа кода. Различные реализации этого метода включают в себя анализ журнала протоколирования, представление и последующий анализ исходного кода в виде XML, а также в комбинации с другими методами, например, как это реализовано в инструменте VeriSol для платформы Azure Blockchain.

Отдельно стоит выделить виды ошибок смарт-контрактов и соответствующие инструментальные средства, которые помогают их выявить: логические (VeriSol, F*, SOLAR, программный каркас на основе теории игр, решающие устройства SMT, SPIN, NuSMV, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов, тензорный анализ, Isabelle/HOL, FEther, VeriSolid, FSPVM-E, SolidiKeY, S-gram); синтаксические (VeriSol, ContractFuzzer, ContractLarva, SmartCheck, Intellij Idea Solidity plugin, Remix); ошибки компиляции (F*, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов); среды исполнения/runtime (NuSMV, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов). Интересным направлением исследования является выявление ошибок взаимодействия, которые могут возникнуть в связи с отсутствием соответствия

программного обеспечения аппаратному интерфейсу или интерфейсу прикладного программирования.

Относительно методов экспертизы были выявлены инструментальные средства аудита [58]. Исследования по сквозному контролю и инспекции смарт-контрактов найдены не были. Это может быть связано с тем обстоятельством, что для сквозного контроля и инспекции необходима группа разработчиков, а полная автоматизация этих методов противоречила бы их принципам. Ввиду того, что относительно других языков программирования рассматриваемые языки смарт-контрактов появились недавно, сложно найти группу разработчиков, которые были бы компетентны выполнять такого рода экспертизы.

Однако, другие методы верификации имеют свои недостатки. Например, верификация по журналам протоколирования [51] определяет ошибки постфактум, когда они уже произошли и были записаны. Можно совместить данный метод с тестированием: перед развертыванием смарт-контракта в эксплуатационной среде провести несколько итераций запусков на тестовых стендах, проанализировать журнал протоколирования и устранить ошибки. Предложенный процесс также может быть автоматизирован.

При проведении обзора выяснилось, что формальная семантика применяется для языков низкого уровня (байт-код). Высокоуровневые языки программирования смарт-контрактов с полной формальной семантикой находятся в разработке [9]. Формальная семантика для языков всех уровней необходима для разработки верифицированных компиляторов. Верификация формальными методами языков высокого уровня также представляется перспективным направлением исследования.

Интересным является направление исследования по предотвращению перехода дефекта в ошибку. Имеется в виду интеграция инструментального средства верификации со средством контроля версий, с помощью которого можно было бы предотвратить миграцию ошибок на последующие релизы проекта.

Отмечая инструментальные средства разработки, следует заметить, что разрабатывать смарт-контракты можно не только на специально созданных для этого приложениях, типа Remix [65]. Другие платформы, такие как Intellij Idea [64], Visual Studio, Eclipse, также поддерживают расширения для редактирования исходного кода смарт-контрактов на языке Solidity.

Интересным для рассмотрения и анализа вопросом являются межконтрактные взаимодействия и рекурсивность в работе контрактов. При этом возникает вопрос о том, как очертить область, в которой рассматривать взаимодействие смарт-контрактов. Моделировать взаимодействие всех развернутых смарт-контрактов в распределенном реестре может быть сложно в зависимости от размеров среды моделирования. В данном случае можно будет построить вероятностную картину и в соответствии с ней очертить область смарт-контрактов, между которыми будет рассматриваться взаимодействие. Инструмент CANDECOMP/PARAFAC позволяет выполнить моделирование межконтрактного взаимодействия.

При анализе исследуемых статей было замечено, что верификация исходного кода смарт-контрактов для языков высокоуровневого типа рассматривалась в основном для языка Solidity, либо для низкоуровневого байт-кода EVM. Необходимо провести исследования и для других языков смарт-контрактов, например, IELE, Scilla, Michelson.

Все перечисленные выше методы структурированы в таблице 1.

Метод	Описание	Инструментальные средства
Формальная верификация		
	Используются формальные методы для доказательства соответствия модели системы установленным спецификациям требований.	VeriSol [26], F* [29], SOLAR [30], программный каркас на основе теории игр [33], решающие устройства SMT [34,35], SPIN [36], NuSMV [38]
Динамическая верификация		

Проверка поведения программы в момент ее выполнения. Полезна при отсутствии доступа к коду программы.	ContractFuzzer [51], ContractLarva [53], извлечение спецификаций из журналов [54], тензорный анализ [55]
Статический анализ кода	
В отличии от динамического анализа происходит без запуска программ. Статический анализ обладает высокой степенью надежности и применяется в жизненно-важных областях [63].	SmartCheck [23, 24, 25], Intellij Idea Solidity plugin [64], онлайн среда разработки Remix [65]
Проверка доказательств теорем	
Использование инструментариев доказательств теорем для подтверждения соответствия спецификациям.	Isabelle/HOL [40, 41], FEther [43], VeriSolid [45], FSPVM-E [49]
Дедуктивный анализ	
Сопоставление предусловия с постусловием в контексте работы функций, процедур, методов для определения корректности их исполнения.	SolidiKeY [39]
Аудит и анализ	
Анализ результатов работы смарт-контракта на соответствие спецификациям.	S-gram [57]

Таблица 1. Классификация иснтрументальных средств верификации смарт-контрактов.

7. Заключение

В настоящем обзоре рассмотрены принципы работы технологии распределенных реестров, общие методы верификации программ, и применимость этих методов к верификации смарт-контрактов. Выделены основные исследовательские области, которые включают в себя текущие методы по верификации смарт-контрактов, используемые инструментальные средства, их преимущества и недостатки. Для ответов на поставленные вопросы использовались статьи, опубликованные в аккредитованных журналах за последние 4 года, а также учебные и методические пособия.

Представлены результаты анализа актуальных на момент написания статьи инструментальных средств, выявлены их сильные и слабые стороны, а также возможности усовершенствования некоторых из них. Рассмотрены принципы работы «контрактных» языков и их особенности в контексте верификации смарт-контрактов.

Выявлены перспективные направления разработки новых, а также усовершенствование уже существующих инструментальных средств верификации смарт-контрактов. Полученные результаты можно использовать при выборе направления будущих исследований в относительно новой и востребованной на практике проблемной области.

8. Литература

[1] Srivastava A., Bhattacharya P., Singh A., Mathur A. A Systematic Review on Evolution of Blockchain Generations // ITEE Journal. 2018, Vol. 7, Num. 6. pp. 3-6.

[2] Daniel P., Benjamin L. Smart Contract Vulnerabilities: Does Anyone Care? Preprint. 2019. pp. 1-3

[3] Gelvez, M. Explaining the DAO exploit for beginners in Solidity // https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470. 2016 (accessed 01.09.2019)

[4] Thomson I. Parity: The bug that put $169m of Ethereum on ice? Yeah, it was on the todo list for months // https://www.theregister.co.uk/2017/11/16/parity_flaw_not_fixed. 2017 (accessed 01.09.2019)

[5] Daniel M., Cristian C., Shang G. Smart Contract Applications within Blockchain Technology: A Systematic Mapping Study // Telematics and Informatics. 2018.pp 1-4

[6] Ethereum: A secure decentralised generalised transaction ledger // http://paper.gavwood.com. 2016 (accessed 01.09.2019)

[7] Xinxin F., Qi C. Roll-DPoS: A Randomized Delegated Proof of Stake Scheme for Scalable Blockchain-Based Internet of Things Systems // 15th EAI International Conference. 2018. pp. 482-483

[8] Grishchenko I., Maffei M., Schneidewind C. A Semantic Framework for the Security Analysis of Ethereum Smart Contract // Principles of Security and Trust. 2018. pp 246-248.

[9] Harz D., Knottenbelt W. Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. 2018. pp. 1-6.

[10] Zakrzewski J. Towards Verification of Ethereum Smart Contracts: A Formalization of Core of Solidity // Verified Software. Theories, Tools, and Experiments. 10th International Conference, VSTTE 2018, Oxford, UK. 2018.

[11] Ethereum Foundation. Vyper documentation. 2018. // https://vyper.readthedocs.io/en/v0.1.0-beta.13/ (accessed 08.10.2019)

[12] Buterin, V. A next-generation smart contract and decentralized application platform. white paper, 3, 2014. 37 p. https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf (accessed 01.09.2019)

[13] ERC: Token standard https://github.com/ethereum/eips/issues/20 (accessed 08.10.2019)

[14] Silva R., Neiva F. Systematic Literature Review in Computer Science - A Practical Guide. 2016. pp. 1-5.

[15] Siddiqui J.H., Rauf A., Ghafoor M.A. Advances in Software Model Checking // Advances in Computers, 2017. pp. 60-70.

[16] Sistla A.P., Clarke E.M. The complexity of propositional linear temporal logics // J. of the ACM (JACM) JACM Homepage archive. V. 32, I. 3. 1985. pp. 733-749.

[17] Кулямин, В.В. Методы верификации программного обеспечения // Всероссийский конкурс обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы», 2008. 117 с.

[18] Kosmatov N., Prevosto V., Signoles J. A lesson on proof of programs with Frama-C // Invited tutorial paper. International Conference on Tests and Proofs. Springer, 2013. pp. 1-3.

[19] Ahrendt W., Beckert B. et al. Deductive Software Verification. - The KeY Book, V. 10001 of LNCS. Springer, 2016. pp. 353-357.

[20] Hähnle R., Huisman M. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. Computing and Software Science. State of the Art and Perspectives. Springer, 2019. pp. 359-361.

[21] Ahrendt W., et al.. The KeY platform for verification and analysis of Java programs // In STTE'14, V. 8471 of LNCS. Springer, 2014. pp 55–71.

[22] Reynders F. Modern API Design with ASP.NET Core 2. Apress. 2018. pp. 113-125.

[23] Grishchenko I., Maffei M., Schneidewind C. Foundations and tools for the static analysis of ethereum smart contracts // Computer Aided Verification. 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK. 2018. Proceedings, Part I. pp. 54-57, 65-72,

[24] Tikhomirov S., Voskresenskaya E. et al. SmartCheck: Static Analysis of Ethereum Smart Contracts // 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. 2018. pp. 12-15.

[25] Reza M., Dehghantanha A., Raymond C., Amritraj S. Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains // CASCON'18, 2018. pp. 5-9.

[26] Lahiri K., Chen S., Wang Y., Dillig I. Formal Specification and Verification of SmartContracts for Azure Blockchain. Preprint. 2018. pp. 2-7, 10-13.

[27] Barnet M. et al. Boogie: A modular reusable verifier for object-oriented programs. / In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, 2005, Revised Lectures, V. 4111 of LNCS. Springer, 2006. pp, 364-367, 380-383.

[28] Tom Simonite. $80 million hack shows the dangers of programmable money // MIT Technology review. 2016. https://www.technologyreview.com/s/601724/ 80-million-hack-shows-the-dangers-of-programmable-money/ (accessed 08.10.2019)

[29] Bhargavan K. et al. Short Paper: Formal Verification of Smart Contract. 2016 // ACM Workshop. 2016. pp. 91-95.

[30] Li A., Long F. Detecting Standard Violation Errors in Smart Contracts. ArXiv. 2018. https://arxiv.org/abs/1812.07702

[31] Godefroid P., Klarlund N., Sen K. DART: Directed Automated Random Testing // ACM SIGPLAN Conf. on Programming language design and implementation. 2005. pp. 3-7.

[32] Sen K., Marinov D., Agha G. CUTE: A Concolic Unit Testing Engine for C // 10th European Software Engineering Conf. Lisbon, Portugal. 2005. pp. 263-265.

[33] Chatterjee K., Kafshdaran A., Velner Y. Quantitative Analysis of Smart Contracts // Programming Languages and Systems. 2018. pp. 753-760.

[34] Barrett C., Tinelli C. Satisfiability Modulo Theories. Handbook of Model Checking. 2018. pp. 305-307, 310-315.

[35] Alt L., Reitwiessner C. SMT-Based Verification of Solidity Smart Contract // 8th International Symposium, ISoLA 2018. Limassol, Cyprus. 2018. pp. 380-387.

[36] Bai X., Cheng Z., Duan Z., Hu K. Formal Modeling and Verification of Smart Contracts // 7th Int. Conf. on Software and Computer Applications - ICSCA. 2018. pp. 324-326.

[37] Mikk E., Lakhnech Y., Siegel M., et al. Implementing statecharts in PROMELA/SPIN // 2nd IEEE Workshop on. IEEE. 1998. pp. 90-101.

[38] Zeinab Nehai Z., Piriou P., Daumas F. Model-Checking of Smart Contracts // IEEE International Conference on Blockchain. 2018. pp. 3-7.

[39] Wolfgang J., Gordon J., Schneider G. Smart Contracts: A Killer Application for Deductive Source Code Verification // Principled Software Development. 2018. pp. 10-14.

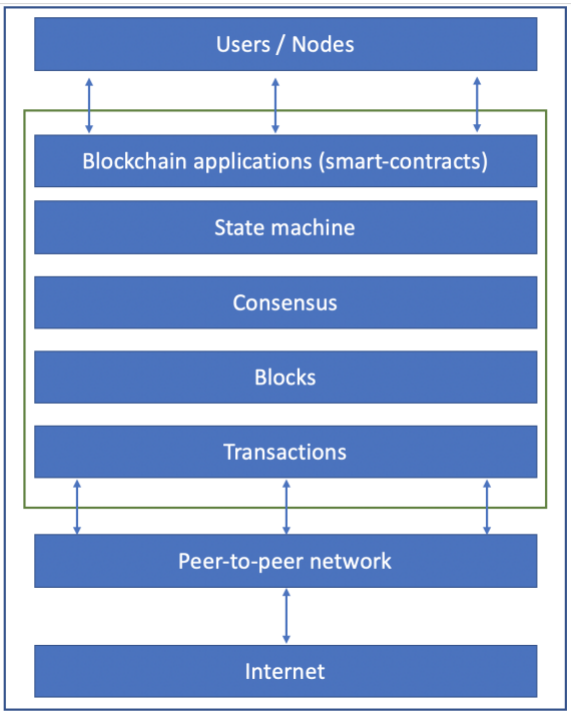
[40] Myriam S., Maksym B., Mark B., Mark S. Towards verifying ethereum smart contract bytecode in Isabelle/HOL // 7th ACM SIGPLAN International Conference. 2018. pp. 67-74.

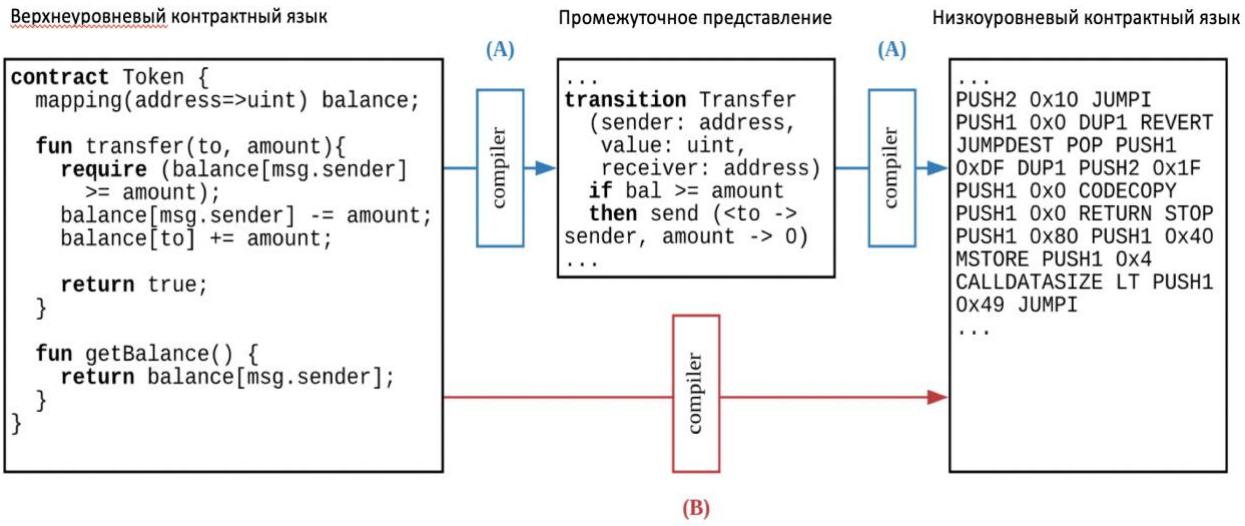
[41] Hirai Y. Defining the ethereum virtual machine for interactive theorem provers. // Int. Conference on Financial Cryptography and Data Security. 2017. pp. 526-531.

[42] Bohm, C., Jacopini G. , Flow diagrams, Turing machines and languages with only two formation rules // Comm. ACM 9. 1966. p. 366.

[43] Yang Z., Lei H. FEther: An Extensible Definitional Interpreter for Smart-contract Verifications in Coq // IEEE Access. 2019. pp. 13-18.
[44] Abhishek Anand A., Boulier S., Cohen C. Towards Certified Meta-Programming with Typed Template-Coq // Interactive Theorem Proving. 2018. pp. 23-29.
[45] Mavridou A. et al. VeriSolid: Correct-by-Design Smart Contracts for Ethereum. // 23rd Int. Conference on Financial Cryptography and Data Security. 2018. pp. 5-16.
[46] Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the bip framework // IEEE Software 28(3). 2011. pp. 41–48.
[47] Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal verification of infinite-state BIP models. In: Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA).Springer, 2015. pp. 326–343.
[48] Zurawski R.Volcano: Enabling Correctness by Design // Networked Embedded Systems. 2017. Chapter 19.
[49] Yang Z., Lei H., Qian W. A Hybrid Formal Verification System in Coq for Ensuring the Reliability and Security of Ethereum-based Service Smart Contracts // Nasa ads 2019. pp. 6-14.
[50] Duchmann F., Koschmider A. Validation of Smart Contracts Using Process Mining // ZEUS. 2019. Workshop on Services and their Composition Proceedings of the 11th Central European Workshop on Services and their Composition, Bayreuth, Germany, February 14–15, 2019. pp. 1-4.
[51] Jiang B., Liu Y., Chan K. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection // 33rd ACM/IEEE Int. Conference. 2018. pp. 260-265.
[52] J Peace G., Ellul J., Azzopardi S. Monitoring Smart Contracts: ContractLarva and Open Challenges Beyond // 18th International Conference on Runtime Verification. Cyprus. 2018. pp. 7-15.
[53] J Peace G., Ellul J., Christian C. Contracts over Smart Contracts: Recovering from Violations Dynamically // Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice. 2018. pp. 300-315.

[54] Guth F., Wüstholz V., Christakis M., Müller P. Specification Mining for Smart Contracts with Automatic Abstraction Tuning // CoRR abs. 2018. pp. 5-10.
[55] Charlier J., State R., Hilger J. Modeling Smart Contracts Activities: A Tensor based Approach // European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD) - Workshop on MIning DAta for financial application. 2019. pp. 2-7.
[56] Bader W. et al. Tensor Decompositions and Applications // SIAM Review. 2009 pp. 455-500.
[57] Liu H. et al. S-gram: Towards Semantic-Aware Security Auditing for Ethereum Smart Contracts // 33rd ACM/IEEE International Conference. 2018. pp. 815-818.
[58] Martin, J. H., & Jurafsky, D. (2009). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. 2nd edition. Upper Saddle River: Pearson/Prentice Hall. 2008. pp. 15-16.
[59] Somin S., Gordon G., Altshuler Y. Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain // Unifying Themes in Complex Systems IX. Proceedings of the Ninth International Conference on Complex Systems. 2018. pp. 440-447.
[60] Parizi M., Singh A., Dehghantanha A. Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security // International Conference on Blockchain. 2018. pp. 4-13.
[61] Hobor A. et al. Scilla: a Smart Contract Intermediate-Level Language. ArXiv. 2018.
[62] Hirai Y.. Bamboo. https://github.com/pirapira/bamboo (accessed 08.10.2019)
[63] Augusto Muñoz C., Schnur C., Siminiceanu R. Static Verification of Spacecraft Procedures // AIAA Infotech@Aerospace Conference. 2009. pp. 2-5.
[64] Intellij Solidity overview. https://plugins.jetbrains.com/plugin/9475-intellij-solidity (accessed 08.10.2019)
[65] Remix documentation. https://remix.readthedocs.io/en/stable/ (accessed 08.10.2019)


Users / Nodes
Blockchain applications (smart-contracts)
State machine
Consensus
Blocks
Transactions
Peer-to-peer network
Internet


Верхнеуровневый контрактный язык
Промежуточное представление
Низкоуровневый контрактный язык
(A)
(A)
compiler
compiler
compiler
(B)
contract Token {
mapping(address=>uint) balance;
fun transfer(to, amount){
require (balance[msg.sender]
>= amount);
balance[msg.sender] -= amount;
balance[to] += amount;
return true;
}
fun getBalance() {
return balance[msg.sender];
}
}
...
transition Transfer
(sender: address,
value: uint,
receiver: address)
if bal >= amount
then send (<to ->
sender, amount -> 0)
...
...
PUSH2 0x10 JUMPI
PUSH1 0x0 DUP1 REVERT
JUMPDEST POP PUSH1
0xDF DUP1 PUSH2 0x1F
PUSH1 0x0 CODECOPY
PUSH1 0x0 RETURN STOP
PUSH1 0x80 PUSH1 0x40
MSTORE PUSH1 0x4
CALLDATASIZE LT PUSH1
0x49 JUMPI
...

Метод	Описание	Инструментальные средства
Формальная верификация		
Используются формальные методы для доказательства соответствия модели системы установленным спецификациям требований.		VeriSol [26], F* [29], SOLAR [30], программный каркас на основе теории игр [33], решающие устройства SMT [34,35], SPIN [36], NuSMV [38]
Динамическая верификация		
Проверка поведения программы в момент ее выполнения. Полезна при отсутствии доступа к коду программы.		ContractFuzzer [51], ContractLarva [53], извлечение спецификаций из журналов [54], тензорный анализ [55]
Статический анализ кода		
В отличии от динамического анализа происходит без запуска программ. Статический анализ обладает высокой степенью надежности и применяется в жизненно-важных областях [63].		SmartCheck [23, 24, 25], Intellij Idea Solidity plugin [64], онлайн среда разработки Remix [65]
Проверка доказательств теорем		
Использование инструментариев доказательств теорем для подтверждения соответствия спецификациям.		Isabelle/HOL [40, 41], FEther [43], VeriSolid [45], FSPVM-E [49]
Дедуктивный анализ		
Сопоставление предусловия с постусловием в контексте работы функций, процедур, методов для определения корректности их исполнения.		SolidiKeY [39]
Аудит и анализ		
Анализ результатов работы смарт-контракта на соответствие спецификациям.		S-gram [57]

Systematic review of automatic verification of smart-contracts.

I.A. Fedotov, PhD student, MIPT, ivan.fedotov@phystech.edu
A.S. Khritankov, Assoc. Prof., MIPT, anton.khritankov@acm.org

Corresponding author:
I.A.Fedotov, PhD student, MIPT, Institutskiy dr., 9, Dolgoprudny, Moscow Region, Russia, 141701.
e-mail: ivan.fedotov@phystech.edu

Abstract.

A smart contract is a special kind of software code that runs on a blockchain platform. As other software smart contracts can contain errors and vulnerabilities that could lead to substantial adverse results and financial losses. These risks are compounded by the growing popularity of blockchain-based systems, devices, and infrastructure in finance, legal, energy and other domains. Early detection of software errors can reduce losses and automatic verification tools are a method of choice for situations where reliability and security is important.

This systematic review analyzes the state of the art in smart contracts verification in 2015-2019. The review gives a brief introduction to blockchain, smart contracts and decentralized applications (dApps), examines verification methods for smart contracts and related tools. Different verification methods are taken into account including formal verification, dynamic and static code analysis, deductive analysis, theorem provers tools, code and execution trace audit.

The following research questions have been considered:

- What code verification methodologies are commonly applied to software verification?
- Which of them are applicable to smart contracts and what tools are available?
- What are current limitations in smart contract verification?

- What are possible future directions in smart contract verification?

During the research process more than 100 academic papers and industrial reports, software tool documentation and guides were studied. As a result, 52 studies, basically in English, were included in the systematic review.

Keywords: Smart-contract, blockchain, verification, systematic review, Ethereum, BitCoin, model checking.

References

[1] Srivastava A., Bhattacharya P., Singh A., Mathur A. A Systematic Review on Evolution of Blockchain Generations // ITEE Journal. 2018, Vol. 7, Num. 6. pp. 3-6.

[2] Daniel P., Benjamin L. Smart Contract Vulnerabilities: Does Anyone Care? Preprint. 2019. pp. 1-3

[3] Gelvez, M. Explaining the DAO exploit for beginners in Solidity // https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470. 2016 (accessed 01.09.2019)

[4] Thomson I. Parity: The bug that put $169m of Ethereum on ice? Yeah, it was on the todo list for months // https://www.theregister.co.uk/2017/11/ 16/parity_flaw_ not_fixed. 2017 (accessed 01.09.2019)

[5] Daniel M., Cristian C., Shang G. Smart Contract Applications within Blockchain Technology: A Systematic Mapping Study // Telematics and Informatics. 2018.pp 1-4

[6] Ethereum: A secure decentralised generalised transaction ledger // http://paper.gavwood.com. 2016 (accessed 01.09.2019)

[7] Xinxin F., Qi C. Roll-DPoS: A Randomized Delegated Proof of Stake Scheme for Scalable Blockchain-Based Internet of Things Systems // 15th EAI International Conference. 2018. pp. 482-483

[8] Grishchenko I., Maffei M., Schneidewind C. A Semantic Framework for the Security Analysis of Ethereum Smart Contract // Principles of Security and Trust. 2018. pp 246-248.

[9] Harz D., Knottenbelt W. Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. 2018. pp. 1-6.

[10] Zakrzewski J. Towards Verification of Ethereum Smart Contracts: A Formalization of Core of Solidity // Verified Software. Theories, Tools, and Experiments. 10th International Conference, VSTTE 2018, Oxford, UK. 2018.

[11] Ethereum Foundation. Vyper documentation. 2018. // https://vyper.readthedocs.io/en/v0.1.0-beta.13/ (accessed 08.10.2019)

[12] Buterin, V. A next-generation smart contract and decentralized application platform. Whitepaper, 3, 2014. 37 p. https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf (accessed 01.09.2019)

[13] ERC: Token standard https://github.com/ethereum/eips/issues/20 (accessed 08.10.2019)

[14] Silva R., Neiva F. Systematic Literature Review in Computer Science - A Practical Guide. 2016. pp. 1-5.

[15] Siddiqui J.H., Rauf A., Ghafoor M.A. Advances in Software Model Checking // Advances in Computers, 2017. pp. 60-70.

[16] Sistla A. P., Clarke E. M. The complexity of propositional linear temporal logics // J. of the ACM (JACM) JACM Homepage archive. V. 32, I. 3. 1985. pp. 733-749.

[17] Kulyamin V.V. Program verification methods // All-Russian competition of review and analytical articles in the priority area "Information and telecommunication systems", 2008. 117 p. (in Russian).

[18] Kosmatov N., Prevosto V., Signoles J. A lesson on proof of programs with Frama-C // Invited tutorial paper. International Conference on Tests and Proofs. Springer, 2013. pp. 1-3.

[19] Ahrendt W., Beckert B. et al. Deductive Software Verification. - The KeY Book, V. 10001 of LNCS. Springer, 2016. pp. 353-357.

[20] Hähnle R., Huisman M. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. Computing and Software Science. State of the Art and Perspectives. Springer, 2019. pp. 359-361.

[21] Ahrendt W., et al.. The KeY platform for verification and analysis of Java programs // In STTE'14, V. 8471 of LNCS. Springer, 2014. pp 55–71.

[22] Reynders F. Modern API Design with ASP.NET Core 2. Apress. 2018. pp. 113-125.

[23] Grishchenko I., Maffei M., Schneidewind C. Foundations and tools for the static analysis of Ethereum smart contracts // Computer Aided Verification. 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK. 2018. Proceedings, Part I. pp. 54-57, 65-72,

[24] Tikhomirov S., Voskresenskaya E. et al. SmartCheck: Static Analysis of Ethereum Smart Contracts // 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. 2018. pp. 12-15.

[25] Reza M., Dehghantanha A., Raymond C., Amritraj S. Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains // CASCON'18, 2018. pp. 5-9.

[26] Lahiri K., Chen S., Wang Y., Dillig I. Formal Specification and Verification of SmartContracts for Azure Blockchain. Preprint. 2018. pp. 2-7, 10-13.

[27] Barnet M. et al. Boogie: A modular reusable verifier for object-oriented programs. / In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, 2005, Revised Lectures, V. 4111 of LNCS. Springer, 2006. pp, 364-367, 380-383.

[28] Tom Simonite. $80 million hack shows the dangers of programmable money // MIT Technology review. 2016. https://www.technologyreview.com/s/601724/ 80-million-hack-shows-the-dangers-of-programmable-money/ (accessed 08.10.2019)

[29] Bhargavan K. et al. Short Paper: Formal Verification of Smart Contract. 2016 // ACM Workshop. 2016. pp. 91-95.

[30] Li A., Long F. Detecting Standard Violation Errors in Smart Contracts. ArXiv. 2018. https://arxiv.org/abs/1812.07702

[31] Godefroid P., Klarlund N., Sen K. DART: Directed Automated Random Testing // ACM SIGPLAN Conf. on Programming language design and implementation. 2005. pp. 3-7.

[32] Sen K., Marinov D., Agha G. CUTE: A Concolic Unit Testing Engine for C // 10th European Software Engineering Conf. Lisbon, Portugal. 2005. pp. 263-265.

[33] Chatterjee K., Kafshdaran A., Velner Y. Quantitative Analysis of Smart Contracts // Programming Languages and Systems. 2018. pp. 753-760.

[34] Barrett C., Tinelli C. Satisfiability Modulo Theories. Handbook of Model Checking. 2018. pp. 305-307, 310-315.

[35] Alt L., Reitwiessner C. SMT-Based Verification of Solidity Smart Contract // 8th International Symposium, ISoLA 2018. Limassol, Cyprus. 2018. pp. 380-387.
[36] Bai X., Cheng Z., Duan Z., Hu K. Formal Modeling and Verification of Smart Contracts // 7th International Conference on Software and Computer Applications - ICSCA. 2018. pp. 324-326.
[37] Mikk E., Lakhnech Y., Siegel M., et al. Implementing statecharts in PROMELA/SPIN // 2nd IEEE Workshop on. IEEE. 1998. pp. 90-101.
[38] Zeinab Nehai Z., Piriou P., Daumas F. Model-Checking of Smart Contracts // IEEE International Conference on Blockchain. 2018. pp. 3-7.
[39] Wolfgang J., Gordon J., Schneider G. Smart Contracts: A Killer Application for Deductive Source Code Verification // Principled Software Development. 2018. pp. 10-14.
[40] Myriam S., Maksym B., Mark B., Mark S. Towards verifying ethereum smart contract bytecode in Isabelle/HOL // 7th ACM SIGPLAN International Conference. 2018. pp. 67-74.
[41] Hirai Y. Defining the ethereum virtual machine for interactive theorem provers. // Int. Conference on Financial Cryptography and Data Security. 2017. pp. 526-531.
[42] Bohm, C., Jacopini G. , Flow diagrams, Turing machines and languages with only two formation rules // Comm. ACM 9. 1966. p. 366.
[43] Yang Z., Lei H. FEther: An Extensible Definitional Interpreter for Smart-contract Verifications in Coq // IEEE Access. 2019. pp. 13-18.
[44] Abhishek Anand A., Boulier S., Cohen C. Towards Certified Meta-Programming with Typed Template-Coq // Interactive Theorem Proving. 2018. pp. 23-29.
[45] Mavridou A. et al. VeriSolid: Correct-by-Design Smart Contracts for Ethereum. // 23rd Int. Conference on Financial Cryptography and Data Security. 2018. pp. 5-16.
[46] Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the bip framework // IEEE Software 28(3). 2011. pp. 41–48.
[47] Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal verification of infinite-state BIP models. In: Proceedings of the 13th

International Symposium on Automated Technology for Verification and Analysis (ATVA).Springer, 2015. pp. 326–343.
[48] Zurawski R. Volcano: Enabling Correctness by Design // Networked Embedded Systems. 2017. Chapter 19.
[49] Yang Z., Lei H., Qian W. A Hybrid Formal Verification System in Coq for Ensuring the Reliability and Security of Ethereum-based Service Smart Contracts // Nasa ads 2019. pp. 6-14.
[50] Duchmann F., Koschmider A. Validation of Smart Contracts Using Process Mining // ZEUS. 2019. Workshop on Services and their Composition Proceedings of the 11th Central European Workshop on Services and their Composition, Bayreuth, Germany, February 14–15, 2019. pp. 1-4.
[51] Jiang B., Liu Y., Chan K. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection // 33rd ACM/IEEE Int. Conf. 2018. pp. 260-265.
[52] J Peace G., Ellul J., Azzopardi S. Monitoring Smart Contracts: ContractLarva and Open Challenges Beyond // 18th International Conference on Runtime Verification. Cyprus. 2018. pp. 7-15.
[53] J Peace G., Ellul J., Christian C. Contracts over Smart Contracts: Recovering from Violations Dynamically // Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice. 2018. pp. 300-315.
[54] Guth F., Wüstholz V., Christakis M., Müller P. Specification Mining for Smart Contracts with Automatic Abstraction Tuning // CoRR abs. 2018. pp. 5-10.
[55] Charlier J., State R., Hilger J. Modeling Smart Contracts Activities: A Tensor based Approach // European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD) - Workshop on MIning DAta for financial application. 2019. pp. 2-7.
[56] Bader W. et al. Tensor Decompositions and Applications // SIAM Review. 2009 pp. 455-500.
[57] Liu H. et al. S-gram: Towards Semantic-Aware Security Auditing for Ethereum Smart Contracts // 33rd ACM/IEEE International Conference. 2018. pp. 815-818.

[58] Martin, J. H., & Jurafsky, D. (2009). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. 2nd edition. Upper Saddle River: Pearson/Prentice Hall. 2008. pp. 15-16.
[59] Somin S., Gordon G., Altshuler Y. Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain // Unifying Themes in Complex Systems IX. Proceedings of the Ninth International Conference on Complex Systems. 2018. pp. 440-447.
[60] Parizi M., Singh A., Dehghantanha A. Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security // International Conference on Blockchain. 2018. pp. 4-13.
[61] Hobor A. et al. Scilla: a Smart Contract Intermediate-Level Language. ArXiv. 2018.
[62] Hirai Y.. Bamboo. https://github.com/pirapira/ bamboo (accessed 08.10.2019)
[63] Augusto Muñoz C., Schnur C., Siminiceanu R. Static Verification of Spacecraft Procedures // AIAA Infotech@Aerospace Conference. 2009. pp. 2-5.
[64] Intellij Solidity overview. https://plugins.jetbrains.com/plugin/9475-intellij-solidity (accessed 08.10.2019)
[65] Remix documentation. https://remix.readthedocs.io/en/stable/ (accessed 08.10.2019)