Примеры применения методов синтеза конечных автоматов для генерации моделей смарт-контрактов

Суворов Д. М., аспирант Университета ИТМО,
dmsuvorov@corp.ifmo.ru

Ульянцев В. И., доцент факультета ИТиП Университета ИТМО,
ulyantsev@corp.ifmo.ru

Аннотация

Современные блокчейн-системы поддерживают создание смарт-контрактов – программ, хранящихся в блокчейне и исполняющихся в узлах распределенной сети. Однако, как показывает практика, разработка корректных смарт-контрактов – сложная задача, и большое количество публичных смарт-контрактов содержат уязвимости. Более того, дизайн блокчейн-систем не позволяет менять код смарт-контрактов после их создания.

Задача автоматического синтеза программ заключается в генерации программного кода на основе некоторой формальной спецификации. В общем случае данная задача неразрешима, однако существуют эффективные методы ее решения для некоторых частных случаев, например, для программ, представленных в виде конечных автоматов. В данной работе мы рассматриваем примеры автоматического синтеза моделей смарт-контрактов в виде конечных автоматов на основе спецификации в линейной темпоральной логике и примеров поведения.

Введение

Современные блокчейн-системы поддерживают создание смарт-контрактов – программ, хранящихся в блокчейне и исполняющихся в узлах распределенной сети. Смарт-контракты удобны тем, что позволяют в виде программного кода выражать произвольные договоренности между участниками сети, причем для выполнения этих договоренностей не требуется наличие третьих доверенных лиц.

Устройство блокчейн-систем не позволяет изменять код смарт-контрактов после их создания, что значительно усложняет задачу про-

ектирования корректных смарт-контрактов. Методы формальной верификации смарт-контрактов активно развиваются, однако задача синтеза смарт-контрактов на основе заданной спецификации не получила достаточного внимания, среди исследований в этом направлении можно выделить работу [3], в которой для специфицирования используется разновидность деонтической логики. В данной работе мы ставим задачу автоматического синтеза автоматных моделей смарт-контрактов, так как логика смарт-контрактов нередко может быть выражена с помощью конечных автоматов [2, 1]. Мы рассматриваем смарт-контракты сети *Ethereum* [5] и им аналогичные – состоящие из набора *методов*, вызов которых может быть осуществлен с помощью отправки транзакции специального вида на адрес смарт-контракта в сети. Такие транзакции могут рассматриваться как события в реактивных системах.

Автоматы и примеры поведения

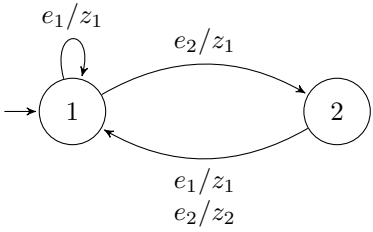

Рис. 1: Пример конечного автомата.

В данной работе *конечным автоматом* называется кортеж $(S, s_{init}, E, Z, \delta, \lambda)$, где S – множество состояний, $s_{init} \in S$ – *начальное состояние*, E – конечное множество входных *событий*, Z – конечное множество выходных *воздействий*, $\delta : S \times E \to S$ – *функция переходов*, и $\lambda : S \times E \to Z^*$, где Z^* – множество строк над Z, – это *функция выходов*. Исполнение автомата – это последовательность циклов: в каждом цикле автомат получает входное событие, генерирует выходные воздействия в соответствии с λ и изменяет свое состояние в соответствии с δ. Пример автомата показан на рисунке 1. Используется следующая нотация для переходов: *входное событие / выходное воздействие*. Если имя выходного воздействия совпадает с именем входного события, то первое может быть опущено.

Примером поведения автомата $(S, s_{init}, E, Z, \delta, \lambda)$ будем называть последовательность пар $(e_1, A_1), \ldots, (e_n, A)_n$, где $e_i \in E$ и $A_i \in Z^*$.

Для решения задачи синтеза конечных автоматов на основе формальной спецификации и примеров поведения был предложен ряд методов, в данной работе мы используем итеративный метод из [4]. Часто для задания спецификации используется темпоральная логика, например *линейная темпоральная логика* (LTL), которая может быть использована для выражения свойств безопасности («событие X не происходит никогда») и живости («событие X произойдет за конечное время») в терминах последовательностей некоторых явлений.

Описание предлагаемого подхода

В данной работе смарт-контракты моделируются с помощью конечных автоматов, при этом входным событиям сопоставлены методы смарт-контрактов, а выходным воздействиям – реализация этих методов в виде программного кода. Таким образом, с помощью инструмента *EFSM-tools* [1] на основе спецификации в темпоральной логике и примеров поведения может быть сгенерирована автоматная модель смарт-контракта, после чего может быть сгенерирован код самого смарт-контракта, если для каждого выходного воздействия был задан соответствующий ему код и определены переменные, описывающие состояние смарт-контракта. Схема данного подхода показана на рисунке 2, для выполнения последнего шага был реализован инструмент *fsmc* [2], генерирующий код на языке *Solidity*.

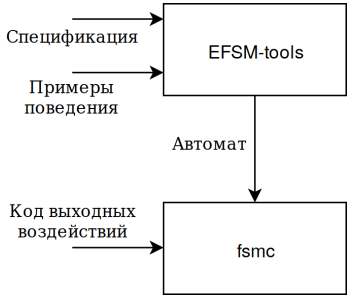

Рис. 2: Схематичное описание предлагаемого подхода.

[1] https://github.com/ulyantsev/EFSM-tools/
[2] https://github.com/d-suvorov/fsmc

Пример: скрытый аукцион

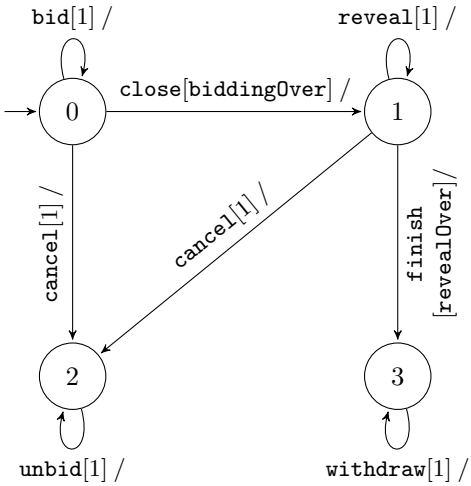

Рис. 3: Скрытый аукцион. Автомат, сгенерированный для числа состояний `size = 4`.

Рассмотрим пример скрытого аукциона из работы [2]. Пользователи могут делать скрытые ставки (событие `bid`) в течение некоторого периода, по окончании которого аукцион может быть закрыт (событие `close`). Для моделирования этого свойства мы вводим предикат `biddingOver`, принимающий истинное значение, когда данный период заканчивается. Далее, в течение следующего периода, пользователи раскрывают совершенные ставки (событие `reveal`), по окончании этого периода (`revealOver == true`) аукцион может быть завершен (событие `finish`), после чего пользователи могут вернуть внесенные средства (событие `withdraw`). В любой момент до завершения аукцион может быть отменен (событие `cancel`), после чего пользователи могут отменить сделанные ставки и вернуть внесенные средства (событие `unbid`).

Таким образом, формальные требования можно сформулировать следующим образом:

1. `close`, `finish` и `cancel` не могут произойти больше одного раза;
2. `bid` не может произойти, если произошло `close`;
3. `reveal` и `cancel` не могут произойти, если произошло `finish`;

4. `finish`, `close`, `bid` и `reveal` не могут произойти, если произошло `cancel`;

5. `finish` и `reveal` не могут произойти пока не произошло `close`;

6. `unbid` не может произойти пока не произошло `cancel`;

7. `withdraw` не может произойти пока не произошло `finish`;

8. `close` может произойти только если `biddingOver == true`;

9. `finish` может произойти только если `revealOver == true`.

Данная спецификация может быть без труда записана в LTL, сгенерированный автомат показан на рисунке 3. В квадратных скобках на переходах отмечены условия, выполнение которых необходимо для совершения перехода.

Заключение

В работе были рассмотрены некоторые примеры смарт-контрактов, логика которых может быть выражена с помощью конечных автоматов. Для данных примеров была записана формальная спецификация и сценарии поведения, с помощью которых были успешно сгенерированы автоматные модели этих смарт-контрактов. При условии того, что выходным воздействиям сопоставлен программный код, на основе построенного автомата может быть сгенерирован код смарт-контракта. Использование формальной логики позволяет верифицировать некоторые свойства полученных смарт-контрактов.

Недостатком рассмотренного подхода является недостаточная выразительность LTL. Например, невозможно выразить свойство вида «`bid` может выполняться произвольное количество раз, пока не выполнится `close`». Для сравнения, в ветвящейся темпоральной логике можно записать свойство вида $\mathbf{EG}\phi$ – существует путь, на котором всюду верно ϕ. Эту проблему можно решить, задав сценарии, в которых событие `bid` происходит подряд N раз, где N заведомо больше числа переходов в автомате. В дальнейшем планируется исследование применения других формальных систем, более подходящих для специфицирования смарт-контрактов различных типов.

Литература

[1] Solidity documentation: Common patterns. — URL: `https://solidity.readthedocs.io/en/develop/common-patterns.html#state-machine`.

[2] Mavridou A. et al. VeriSolid: Correct-by-design smart contracts for Ethereum //arXiv preprint arXiv:1901.01292. – 2019.

[3] Idelberger F. et al. Evaluation of logic-based smart contracts for blockchain systems //International Symposium on Rules and Rule Markup Languages for the Semantic Web. – Springer, Cham, 2016. – С. 167-183.

[4] Ulyantsev V., Buzhinsky I., Shalyto A. Exact finite-state machine identification from scenarios and temporal properties //International Journal on Software Tools for Technology Transfer. – 2018. – Т. 20. – №. 1. – С. 35-55.

[5] Wood G. et al. Ethereum: A secure decentralised generalised transaction ledger //Ethereum project yellow paper. – 2014. – Т. 151. – С. 1-32.