

AUTOMATED VERIFICATION OF FAIRNESS IN SMART CONTRACTS

TECHNICAL FIELD

5 The present disclosure relates to automated verification of fairness in smart contracts.

BACKGROUND

10 Blockchain technology has been developed rapidly in recent years. The distributed and tamper-resistant nature of blockchain has made it the perfect platform for hosting smart contracts. Smart contracts are computer programs running atop blockchain platforms to manage large sums of money, carry out transactions of assets, and govern the transfer of digital rights between multiple parties. Ethereum and EOS are among the most popular blockchain platforms which support smart contracts.

15 The security of smart contracts has been at the center of attention, ever since their adoption in the management of large monetary transactions. Attacks on Ethereum and EOS have exploited security vulnerabilities neglected during contract development, causing a loss for the contract owners (and possibly other honest participants). These vulnerabilities are programming errors, indicating a mismatch between the contract developers' expectations and how the contract code actually works. They are easy to detect once the vulnerability patterns are recognized. In fact, much research has been dedicated to preventing, discovering, and mitigating such attacks.

25 In contrast, the fairness issues of smart contracts have not yet attracted much attention. A smart contract is unfair to certain participants if there is a mismatch between the participants' expectations and the actual implementation of the game rules. It is possible that a malicious party may gain an advantage over others through the exploitation of contract design flaws, i.e., orthogonal to the security issues.

30 For example, smart contracts may well be advertised as "social games" with a promised 20% return for any investment, but turn out to be Ponzi schemes. In this case, the possibility that the game may eventually slow down and never pay back is intentionally left out. Similarly, many auction distributed applications (DApps) claim to be safe and fair, yet it is still possible for bidders to collude among themselves or with the auctioneer to make a profit at the expenses of the others.

The fairness issues mostly reside in contract logic: some of them are unfair by design, while the rest are careless mistakes. This makes the detection of such issues particularly challenging, because every case can be different and there is no hope in identifying predefined patterns. Since it is often not the contract creators' interest at risk (or even worse when they gain at the expenses of participants), there is little incentive for them to allocate resources in ensuring the fairness of their contracts. On the other hand, it is rather difficult, for inexperienced users, to tell whether a contract works as advertised, even with the source code available.

It would be desirable to address or alleviate the above issues.

SUMMARY

The present disclosure relates to a computer-implemented method of automatically verifying at least one fairness property of a smart contract, the method comprising:

(i) performing symbolic execution on source code of the smart contract to extract a mechanism model,

wherein the mechanism model comprises a representation of processing of inputs from a finite number of users by the smart contract to produce an outcome;

(ii) encoding the mechanism model and the at least one fairness property as a satisfiability modulo theories (SMT) formula; and

(iii) automatically determining, from the SMT formula, whether the at least one fairness property is satisfied for the finite number of users.

In some embodiments, operation (iii) is carried out by an SMT solver.

The at least one fairness property may be selected from one or more of: truthfulness, optimality, efficiency, and collusion-freeness.

In some embodiments, if the at least one fairness property is satisfied for the finite number of users, the method further comprises:

(iv) obtaining, during said determining, one or more invariants of the mechanism model;

(v) instrumenting the source code of the smart contract with the one or more invariants to generate a modified smart contract, said one or more invariants being parametrized by an unlimited number of users; and

(vi) automatically determining, from the modified smart contract, whether the at least one fairness property is satisfied for the unlimited number of users.

In some embodiments, operation (vi) is carried out by a program verification tool.

5 In some embodiments, the one or more invariants are obtained in accordance with one or more invariant templates.

In some embodiments, the method comprises instrumenting the source code with a harness program that orchestrates interactions of the users with a target contract of said smart contract.

10

The smart contract may encode an auction process or a voting process.

The present disclosure also relates to a system for automatically verifying at least one fairness property of a smart contract, the system comprising:

15

a model extraction module that is configured to:

(i) perform symbolic execution on source code of the smart contract to extract a mechanism model,

wherein the mechanism model comprises a representation of processing of inputs from a finite number of users by the smart contract to produce an outcome; and

20

(ii) encode the mechanism model and the at least one fairness property as a satisfiability modulo theories (SMT) formula; and

a property checking module that is configured to:

(iii) automatically determine, from the SMT formula, whether the at least one fairness property is satisfied for the finite number of users.

25

In some embodiments, the property checking module is configured to carry out operation (iii) by an SMT solver.

30 In some embodiments, the at least one fairness property is selected from one or more of: truthfulness, optimality, efficiency, and collusion-freeness.

Some embodiments of the system further comprise a fairness verification module that is configured to execute in response to the at least one fairness property being satisfied for the finite number of users, the fairness verification module further being configured to:

35

(iv) obtain, during operation (iii), one or more invariants of the mechanism model;

(v) instrument the source code of the smart contract with the one or more invariants to generate a modified smart contract, said one or more invariants being parametrized by an unlimited number of users; and

5 (vi) automatically determine, from the modified smart contract, whether the at least one fairness property is satisfied for the unlimited number of users.

In some embodiments, the fairness verification module is configured to carry out operation (vi) by a program verification tool.

10 In some embodiments, the fairness verification module is configured to obtain the one or more invariants in accordance with one or more invariant templates.

In some embodiments of the system, said source code is instrumented with a harness program that orchestrates interactions of the users with a target contract of said smart
15 contract.

In some embodiments of the system, the smart contract encodes an auction process or a voting process.

20 The present disclosure further relates to on-transitory computer-readable storage having stored thereon machine-readable code for causing at least one processor to carry out a method as disclosed herein.

BRIEF DESCRIPTION OF THE DRAWINGS

25 Some embodiments of a system and a method for automated verification of fairness in smart contracts, in accordance with present teachings will now be described, by way of non-limiting example only, with reference to the accompanying drawings in which:

Figure 1 is a schematic overview of an example system for automated verification of
30 fairness in smart contracts;

Figure 2 shows a harness program for mechanism model orchestration in the system of Figure 1;

Figure 3 shows Solidity source code for an example auction contract;

Figure 4 shows a mechanism model extracted from the auction contract of Figure 3;

35 Figure 5 shows a context-free grammar for a mechanism model description language for smart contracts that encode auction or voting processes;

Figure 6 shows semantic rules of the auction/voting model of Figure 5;

Figure 7 shows Solidity source code for an example auction contract (EtherAuction);

Figure 8 shows model extraction time with increasing number of bidders for the auction contract of Figure 3;

Figure 9 shows property checking time for four different fairness properties for different numbers of bidders for the auction contract of Figure 3; and

- 5 Figure 10 is an exemplary architecture of a computer system suitable for implementing embodiments of the presently disclosed method.

DETAILED DESCRIPTION

- 10 The present disclosure generally relates to a framework for verifying fairness properties of smart contracts. Since general fairness is largely a subjective concept determined by personal preferences, there is no universal truth when considering only a single participant. Accordingly, at least some embodiments treat a smart contract as a game or mechanism which accepts inputs from multiple participants, and after a period of time,
- 15 decides the outcome according to some predefined rules. Upon the game ending, each participant receives a certain utility depending on the game outcome. With such a mechanism model, it is possible to verify a wide range of fairness properties, including truthfulness, efficiency, optimality, and collusion-freeness. It is also possible to define customized properties based on specific needs.

20

Figure 1 shows the overall workflow of an example system 100 for verifying fairness properties of smart contracts. The system 100 comprises three modules: a model extraction module 110, a property checking module 120, and a fairness verification module 130.

25

In a first operation 104, source code of a smart contract may be automatically instrumented according to user-provided annotations, based on input 102 of the smart contract and a fairness property p . At this stage, a k -player bounded model is considered. The instrumented contract code contains a harness program 112 which orchestrates the

30 interactions between the players and the target contract.

- Next, model extraction module 110 extracts a mechanism model and encodes it, based on the instrumented contract. The extraction of the mechanism model is powered by symbolic execution of the harness program 112, and an intermediate mechanism model is
- 35 synthesized as a result.

In order to perform property checking, the intermediate mechanism model, along with the desired fairness property p , are encoded as an SMT formula, such that the formula is

unsatisfiable if and only if the property p holds with respect to the model. For example, an SMT solver may be used by property checking module 120 to perform the checking. The SMT solver may declare that the property holds when the number of participants are bounded by k (and generate one or more successful examples 128); otherwise, one or
 5 more counterexamples 126 are generated which disputes the property.

If no counterexample 126 is found in the bounded case (i.e. for a finite number k of users), fairness verification module 130 may be executed to verify one or more fairness properties for an unbounded number of participants. To do that, the harness program 112 is modified
 10 to account for an unlimited number of players, and is instrumented with a program invariant as well as the desired fairness properties as post-conditions. The fairness verification module 130 may invoke one or more program verification tools to discharge the proof obligations. This either provides a result that the property is successfully proved, or that the validity of the property is still unknown, in which case it is only possible to be
 15 confident about fairness for the bounded case.

Further details of operation of the modules 110, 120, 130 will now be provided.

Mechanism Model Extraction 110

20 To extract a mechanism model out of the smart contract source code 104, the contract code 104 is first instrumented with a harness program 112, such as the harness program MechanismHarness shown in Figure 2. The harness program 112 orchestrates the interactions of a finite number k players with the target contract. In the context of an
 25 auction or voting contract, this is achieved by declaring symbolic variables to represent the possible bid and valuation of each player, stored in the arrays "BID" (Line 3 of Figure 2) and "VALUE" (Line 4 of Figure 2), respectively. Then a for-loop (Lines 5-19) is used to simulate the actions performed by the k players. In a smart contract, all players have to move sequentially since parallelization is not allowed. The ordering is not important,
 30 because the players are symmetric.

In some embodiments, the annotations provided by users (see, e.g., Figure 3) are used to construct the loop body, which triggers a move from one particular player. The variables controlling the player's identifier and bid value are assigned the corresponding symbolic
 35 values (Lines 7 and 9 of Figure 2). In the example of Figure 3, these variables are msg.sender and msg.value, respectively. Then the allocation function (e.g., bid() in Figure 3) is inlined, and the resulting variables annotated by @allocate and @price are stored as symbolic expressions (Lines 13 and 15 of Figure 2). There are also two placeholders at

Lines 18 and 21 of Figure 2, for assertions of loop invariant and post conditions, which will be described below.

Symbolic execution is then performed at 114 on the harness program 112 to collect a set of feasible symbolic paths. Each symbolic path is represented in the form of "*Condition* $\wedge$ *Effect*", where "*Condition*" and "*Effect*" are Boolean expressions in terms of the symbolic variables defined earlier (e.g., $BID[i]$ and $VALUE[i]$ in Figure 2). Here, "*Condition*" represents the path condition which enables the execution of a particular program path; "*Effect*" represents the values of the resulting variables (e.g., $ALLOCATE$ and $PRICE$ in Figure 2). Taking all path conditions $Condition_j$, where the effect is successfully computed (i.e., not running into errors or reverts), the disjunction of them is used as the assumption of the model (i.e., "assume $\bigvee_j Condition_j$ "). Similarly, the effects are used as the corresponding allocation and clear price functions. For example, in the mechanism model, "allocate $\bigvee_j (Condition_j \wedge Effect_j[ALLOCATE])$ " and "price $\bigvee_j (Condition_j \wedge Effect_j[PRICE])$ ".

Next, at 116, the results of the symbolic execution are used to synthesize the mechanism model M . For example, this may be automated using a domain-specific (e.g. auction/voting specific) language, an example syntax of which is shown in Figure 5.

Accordingly, at 118, the mechanism model M and fairness property p can be encoded. The goal is to construct a formula ϕ such that ϕ is unsatisfiable if and only if $M \models p$. For example, with the semantic rules defined in Figures 5 and 6, it is straightforward to obtain a formula encoding the allocation and clear price functions, i.e., $\phi_M = d(\theta) \wedge t(\theta)$.

The encoding of fairness properties may be illustrated using the truthfulness as an example. As defined herein, a model M is truthful if and only if the truth-telling strategy performs no worse than any other strategies. Therefore, the high-level idea is to first encode the truthful and untruthful strategies separately for an arbitrary player, and then assert that the utility of the player is higher when he/she acts untruthfully. The encoding of the truthfulness property p is then as follows:

$$\begin{aligned}
 & \exists i \cdot \forall j \cdot (i \neq j) \Rightarrow \\
 & \quad (\phi_M \wedge (bid_i = val_i) \wedge (bid_j = val_j)) \quad (\text{Truthful}) \\
 & \quad \wedge (\phi_M[bid_i/bid'_i, u_i/u'_i] \wedge (bid'_i \neq val_i)) \quad (\text{Untruthful}) \\
 & \quad \wedge (u_i < u'_i), \quad (\text{Utility})
 \end{aligned}$$

where i is a generic player with utility u_i . The truthful scenario is when all players (including i) bid the same amount as their valuations, i.e., $bid_i = val_i$ and $bid_j = val_j$. The untruthful model is constructed by substituting the bid and utility variables of i with new copies bid'_i and u'_i , and asserting $bid'_i \neq val_i$. Finally, it is asserted that $u_i < u'_i$.

5

The encoding 122 of model M is then tested at 124 to see whether it satisfies p , for example by using an SMT solver. If p is satisfiable, at least one counterexample 126 is found where an untruthful strategy performs better than the truthful strategy. Otherwise, the truthful strategy is a dominant strategy for i . The encodings of other fairness properties are similar.

10

Fairness verification module 130

Consider the harness program in Figure 2. The loop iterates k times to model k players joining in each iteration. Induction can be used to prove that the smart contract satisfies the fairness property p for an arbitrary number of players. Following the standard approach to proving program correctness, an invariant for the for-loop is required, i.e., $\langle \text{invariant} \rangle$ in Figure 2. Typically, the loop invariant has to be derived manually. However, smart contracts are usually written in a more standard way than arbitrary programs, which makes it easier to generalize invariants for the same type of smart contracts, e.g., auctions as considered herein. A set of predefined invariant templates 103 (according to specific types of contracts) may be provided to the fairness verification module 130 as inputs, to perform example-guided invariant generation 132. The following are three common types of invariants required for auctions:

25	ALLOCATE	= $\arg \max(\text{BID})$	(TopBidder)
	PRICE	= $\max(\text{BID})$	(1st-Price)
	PRICE	= $\max(\text{BID} \setminus \{\text{BID}[\arg \max(\text{BID})]\})$	(2nd-Price)

The "TopBidder" invariant requires that the bidder with the highest bid becomes the winner. The "1st-Price" invariant requires that the highest bid is the clear price, while the "2nd-Price" invariant requires that the second highest bid is the clear price. However, the invariant has to satisfy two conditions to conclude that the smart contract satisfies the fairness property. To elaborate on the conditions, the following notations are defined. Let the harness program in Figure 2 be abstracted as:

35

$$\text{for } (\text{Cond}) \{ S; \text{assert } (Q) \} \text{assert}(P)$$

where Cond is the loop condition, Q and P are the $\langle \text{invariant} \rangle$ and $\langle \text{property} \rangle$, respectively, and S represents the statements in the loop body before the assertion of the

invariant. The *strongest postcondition* operator is also required for discussion. The notation $sp(Pre, Stmt)$ represents the strongest postcondition after the program statement $Stmt$ is executed, provided the precondition Pre before the execution. For example, $sp(x = 2, "x := x + 1")$ would be $x = 3$.

5

The validity for invariants is now formally defined. The invariant has to satisfy the following two conditions (denoted as 134 in Figure 1):

(1) the invariant is inductive, i.e., $sp(Cond \wedge Q, S) \Rightarrow Q$. Intuitively, it means that no matter how many iterations the loop performs, the invariant always holds.

10 (2) the invariant is strong enough to guarantee the fairness property, i.e., $Q \Rightarrow P$.

The validity of the conditions 134 is then checked at 136. If the conditions are satisfied, it can be concluded that the smart contract is fair for an arbitrary number of players. The validity of the conditions can be checked by any program verification tool. For example,

15 Dafny (<https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/>) may be used.

Notice that, in the search for a valid invariant, those that violate the two validity conditions when analyzing mechanism models for a bounded number of players are omitted. That is,

20 only those invariants that are valid for the bounded models are considered in proving for an arbitrary number of players. More fairness properties may be proved when customized invariants are provided.

A particular example will now be described with reference to Figures 3 and 4.

25

Figure 3 shows a simplified Ethereum smart contract, named CryptoRomeAuction, written in Solidity, taken from Etherscan (<https://etherscan.io/address/0x760898e1e75dd7752db30bafa92d5f7d9e329a81>). The contract implements a variant of open English auction for a blockchain-based strategy game, where players are allowed to buy virtual lands with cryptocurrencies. The auction is given a predefined life cycle parameterized by start and end times. A participant can place a bid by sending a message to this contract indicating the value of the bid. The address of the participant and the bid amount are stored in variables `msg.sender` and `msg.value`, respectively. The address of the current highest bidder is recorded in

30 `highestBidder` (Line 9), and a mapping `refunds` is used to keep the contributions of each participant (Line 10) for possible refunding later. The `bid()` function (Lines 11-21) is triggered upon receiving the message. The bid is rejected if the bid amount is no more than the sum of the current highest bid and the minimal increment value duration (Lines

35

13-15). Otherwise, the previous highestBidder gets a refund (Lines 16-18), and the highestBidder (Line 19) and highestBid (Line 20) are updated accordingly.

One way that CryptoRomeAuction can become unfair to the participants is through so-called shill bidding. A shill tries to escalate the price without any intention of buying the item. This can be induced by either the auctioneer or adversarial participants, and other bidders may need to pay more as a result. Occasionally, the shill wins the auction if no other higher bid comes before auction ends. The item may then be sold again at a later time.

Apart from shill bidding, there are a number of other well-studied properties from the game theory and mechanism design literature, which can be used to evaluate the fairness of an auction. The example instances shown in Table 1 are used to demonstrate. Suppose there are three bidders, p_1 , p_2 , and p_3 , participating in the auction. Each of them has a valuation of the item, i.e., the item worth three, four, and six units of utility for p_1 , p_2 , and p_3 , respectively. The Columns "Truthful", "Untruthful", and "Collusion" in Table 1 show the three example scenarios, where the players act truthfully, untruthfully, and collude among themselves. The Rows "Bid", "Allocation", "Price", and "Utility" show the bids placed, the final allocation of the item, the clear price, and the utilities obtained by the bidders, respectively.

Table 1: Example instances of CryptoRomeAuction.

	Truthful			Untruthful			Collusion		
Bidder	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3
Valuation	3	4	6	3	4	6	3	4	6
Bid	3	4	6	3	4	5	3	0	4
Allocation	✗	✗	✓	✗	✗	✓	✗	✗	✓
Price	0	0	6	0	0	5	0	0	4
Utility	0	0	0	0	0	1	0	1	1

In the same way as for other first-price auction schemes, CryptoRomeAuction is not truthful, i.e., bidding truthfully according to one's own valuation of the item is not a dominant strategy. In the ideal truthful scenario, all bidders bid according to their valuations, and p_3 wins the bid with a utility of zero, because the payment equals to his/her valuation of the item. In another scenario, where p_3 bids five (untruthfully), his/her utility would increase by one because of the lower clear price. This is called bid shading, which

only affects the revenue from the auction in this example, but may affect other participants' utilities in some other cases.

In the third scenario, p_2 and p_3 collude in order to gain extra profits. With full knowledge of each other's valuations, p_2 and p_3 may decide to form a cartel and perform bid shading. One possibility is to have p_2 forfeit his/her chance and p_3 bids four, and they divide the profit equally among themselves. Each of them gains one unit of utility as a result.

Given a mechanism model abstracting the auction settings, the set of fairness properties are well-defined and can be formally specified based on the model. An illustration of how to extract the underlying mechanism model from the smart contract source code for CryptoRomeAuction and outline the process of automated property checking as well as verification is now provided.

Albeit variations in implementations, all auction contracts share some common components, such as the bidders' identifiers, their bids, and the allocation as well as clear price rules. Users may provide annotations for these components directly on the source code, as shown in Lines 2-7 in Figure 3. Specifically, the annotations specify the bidders' information as a tuple, "@individual(msg.sender,msg.value)", indicating the variables used to store the identifier and the bid value, respectively. Similarly, "@allocation(highestBidder)" and "@price(highestBid)" indicate that the allocation result and the clear price are stored in highestBidder and highestBid, respectively. Finally, "@outcome" is used to label the function defining the auction allocation logic.

With these labels, symbolic execution (operation 114 in Figure 1) is performed on the bid() function treating the participants' inputs -- msg.value -- as symbolic variables. The result of this would be two symbolic expressions for highestBidder and highestBid, which symbolically represent the allocation and clear price functions, respectively. This information can be used to synthesize an intermediate mechanism model, shown in Figure 4. The model may be specified in a customized language designed for auction and voting contracts, details of which will be described below with reference to Figures 5 and 6. Generally speaking, the model specifies information of the participating individuals and the auction rules: consider a bounded model with only three bidders (i.e., $msgsender_1$, $msgsender_2$, and $msgsender_3$). Their bids have to satisfy the constraint specified in the assume clause, the allocation function is given as " $\arg \max(msgvalue_1, msgvalue_2, msgvalue_3)$ ", and the clear price function is given as " $\max(msgvalue_1, msgvalue_2, msgvalue_3)$ ".

The intermediate mechanism model in Figure 4 has well-defined mathematical semantics, which can be used to check the desired fairness properties. Both the model and the property are encoded with an SMT formula such that a counterexample exists if and only if the formula is satisfiable. More details on the encoding have been provided above.

5

If the formula is unsatisfiable, there is confidence that the property holds for the bounded case with three bidders. A proof of the property can be attempted by instrumenting the contract program with program invariants encoding the allocation and clear price clauses synthesized previously, but parameterized by an unbounded number of bidders. The instrumented program and the property are then passed to a program verification tool, such as Dafny, to perform the automated verification.

10

Analysis framework for smart contract fairness

Some background and definitions on mechanism models and fairness properties, and an abstract syntax and semantics of a mechanism modeling language that supports automated model construction and property checking, will now be described with reference to Figures 5 and 6.

20 *Smart Contracts as Mechanism Models*

Mechanism design is used to design economic mechanisms or incentives to help attain the goals of different stakeholders who participate in the designated activity. The goals are mainly related to the outcome that could be described by participants' payoff and their return in the activity. The logic behind smart contracts is modeled with a mathematical object known as the mechanism.

25

In a mechanism model, there is a finite number of individuals, denoted by $N = \{1, 2, \dots, n\}$. Each individual i holds a piece of private information represented by a type, denoted $\theta_i \in \Theta_i$. Let the types of all individuals be $\theta = (\theta_1, \dots, \theta_n)$, and the space be $\Theta = \times_i \Theta_i$. The individuals report, possibly dishonestly, a type (strategy) profile $\hat{\theta} \in \Theta$. Based on everyone's report, the mechanism model decides an outcome which is specified by an allocation function $d: \Theta \mapsto O$, and a transfer function $t: \Theta \mapsto \mathbb{R}^n$, where $O = \{o_i \in \{0, 1\}^n \mid \sum_i o_i = 1\}$ is the set of possible outcomes.

30

35

The preferences of an individual over the outcomes are represented using a valuation function $v_i: O \times \Theta_i \mapsto \mathbb{R}$. Thus, $v_i(o, \theta_i)$ denotes the benefit that individual i of type θ_i receives from an outcome $o \in O$, and $v_i(o, \theta_i) > v_i(o', \theta_i)$ indicates that individual i prefers o to o' . The

individual i 's utility under strategy profile $\hat{\theta}$ is calculated by subtracting the payment to be made from the valuation of a certain outcome: $u_i(\hat{\theta}) = v_i(\hat{\theta}, \theta_i) - t_i(\hat{\theta})$.

Fairness properties

5 The fairness of smart contracts is usually subject to the understandings and preferences of the participating parties -- a contract fair to someone may be unfair to the others. In particular, fairness can be considered from both the participants' and the contract creators' points of view. To capture such nuances, individual parties may be modeled separately before such subjective fairness properties are specified against the model.

10

Generally speaking, all properties which can be expressed in terms of the mechanism model defined in Figures 5 and 6 (and the relevant accompanying disclosure below) are supported by the present reasoning framework. To keep the presentation simple, in the present disclosure, there is a focus on analyzing a set of generic fairness properties based on the mechanism models. Four types of well-understood properties, namely, truthfulness, 15 optimality, efficiency, and collusion-freeness are discussed in the context of the present disclosure, but it will be understood that other properties may also be used.

To formally define the properties, the concept of dominant strategy is first introduced. The notation $\hat{\theta}_{-i}$ is used to denote the strategy profile of the individuals other than i , i.e. $(\hat{\theta}_1, \dots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \dots, \hat{\theta}_n)$. Therefore, $(\hat{\theta}'_i, \hat{\theta}_{-i})$ is used to denote the strategy profile which differs from $\hat{\theta}$ only on $\hat{\theta}_i$. 20

Definition 1 (Dominant Strategy). A strategy $\hat{\theta}_i \in \theta_i$ is a dominant strategy for i , if $\forall \hat{\theta}_{-i} \forall \hat{\theta}'_i \in \theta_i \cdot u_i(\hat{\theta}_i, \hat{\theta}_{-i}) \geq u_i(\hat{\theta}'_i, \hat{\theta}_{-i})$. When equality holds, the strategy is a weak dominant strategy. 25

A mechanism model is truthful if and only if for any individual and strategy profile, reporting one's real type (truthtelling, i.e., $\forall i \in N \cdot \hat{\theta}_i = \theta_i$) is a dominant strategy.

30 Definition 2 (Truthfulness). Formally, a mechanism is truthful if and only if, $\forall \theta_{-i} \forall \hat{\theta}_i \in \theta_i \cdot u_i(\theta_i, \theta_{-i}) \geq u_i(\hat{\theta}_i, \theta_{-i})$.

Given an auction smart contract with many bidders competing for a single indivisible good, the account which creates the contract is the auctioneer and the accounts which join the auction are the bidders. If the auction prevents bidders from benefiting more by bidding less, it is truthful. When bidding untruthfully is not a good strategy, the auction can generally attract more honest bidders and the auctioneer can get higher revenue for the good on sale. 35

Definition 3 (Efficiency). In the present disclosure, a mechanism is efficient if and only if its allocation function achieves maximum total value, i.e., $\forall \hat{\theta} \in \Theta \forall d' \cdot \sum_i v_i(d(\hat{\theta}), \theta_i) \geq \sum_i v_i(d'(\hat{\theta}), \theta_i)$.

5

Suppose no bidder can affect any other bidder's valuation. If the only winner is the bidder who values the good the most, the auction is efficient.

Definition 4 (Optimality). In the present disclosure, a mechanism is optimal if and only if its transfer function achieves maximum total net profit, i.e., $\forall \hat{\theta} \in \Theta \forall t' \cdot \sum_i t_i(\hat{\theta}) \geq \sum_i t'_i(\hat{\theta})$.

10

Similarly, if the winner is the one who bids the highest, the auction is optimal. In this case, the auctioneer receives the highest revenue.

15 The notation $\hat{\theta}_{-ij}$ is used to denote the strategy profile of individuals other than i and j , i.e., $\{\hat{\theta}_1, \dots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \dots, \hat{\theta}_{j-1}, \hat{\theta}_{j+1}, \dots, \hat{\theta}_n\}$.

Definition 5 (2-Collusion Free). In the present disclosure, a mechanism is 2-collusion free if there does not exist a cartel of individuals i and j whose untruthful strategies increase the group utility, formally, $u_i(\hat{\theta}_i, \hat{\theta}_j, \theta_{-ij}) + u_j(\hat{\theta}_i, \hat{\theta}_j, \theta_{-ij}) \geq u_i(\theta_i, \theta_j, \theta_{-ij}) + u_j(\theta_i, \theta_j, \theta_{-ij})$.

20

Collusion is a big concern in auction and other multi-player games. The basic 2-collusion freeness property in an auction means that any two bidders' collusion cannot help them achieve higher gain. This prevents bid price manipulation to a certain extent, which helps guarantee a fair chance for all bidders and maintain good revenue for the auctioneer. A more general version, i.e., n -collusion freeness, can be defined in a similar way.

25

Mechanism Modeling Language

30 With reference to Figures 5 and 6, an example domain-specific language to facilitate the automated translation from smart contracts to mechanism models will now be described.

An abstract syntax of the mechanism modeling language is defined, which is applicable to both auction and voting. Figure 5 shows the context-free grammar of the language. A mechanism model comprises one or more individuals, an assumption, an outcome, and a property to be verified. An individual is defined as a triple containing the identifier "*id*", bid amount "*bid*", and valuation "*val*". An assumption is a Boolean constraint which should be satisfied upon the entry of the contract. The outcome of the contract is specified by the

35

allocation and the clear price functions, which are expressions over id and bid . Voting contract typically does not have a clear price function. Properties are allowed to be specified using a Boolean expression optionally preceded by a “forall” quantifier.

- 5 Language Semantics. The semantic mapping from the modeling language to the underlying mechanism model is summarized in Figure 6. The “Indiv” rule maps the individuals and their reported types as well as valuations. More specifically, the individuals’ bids are mapped to their reported types $\hat{\theta}_i$, and an individual of type θ_i ’s valuation of the item $v_i(o_i, \theta_i)$ is val_i , where o_i denotes the outcome where the item is allocated to i . The “Alloc” rule conjuncts the Boolean expression assumption from the “assume” clause and the symbolic expression allocation in terms of individuals’ strategies from the “allocate” clause, which is evaluated as the allocation function. Similarly, the transfer function is the conjunction of the assumption and the clearprice expressions. There are some differences between auction and voting: clear price is absent from voting, where allocation is done by comparing the number of ballots (bids) by the participating individuals; whereas in auction, the individuals who bid no less than the clear price can be allocated the item.

- This language works for the most commonly seen auction and voting contracts with fairness concerns. For example, Ethereum smart contracts meeting the ERC-1202 (voting) and ERC-1815 (blind auctions, under review) standards all follow the same interface and structure, therefore can be automatically translated into our modeling language. Similar languages can also be designed for other types of contracts (e.g., social games and decentralized finance applications such as token exchanges, insurance, etc.). The proposed modeling language can be modified and extended to establish suitable mappings from new contract types to the classic mechanism model. With the new modeling language, the model extraction and property checking algorithms can be directly reused.

Experimental evaluation

- 30 An approach consistent with the present disclosure has been implemented in a tool called FairCon, which takes an annotated smart contract source code with the fairness property to be checked as input, extracts its mechanism model for a finite number of players, and then automatically performs symbolic path analysis on the model. If the fairness property is violated in one symbolic path, FairCon generates a counterexample related to that path.
- 35 All the symbolic path analysis is achieved based on the Z3 SMT solver. If no counterexample is founded within a finite number of players, FairCon then tries to prove that the smart contract satisfies the fairness property based on induction with the set of predefined invariants. During the process, Dafny is used to check the two validity

conditions of invariants to establish the fairness proof. The following experiment was conducted to evaluate the accuracy of the fairness properties check performed by FairCon on smart contracts and the efficiency of FairCon as a mechanism model extraction and fairness property checking tool.

5

Experiment Setup

Many Ethereum smart contracts are token-based and derived from standard templates (e.g., there are 259,131 ERC-20 contracts on Etherscan), but such contracts have few fairness concerns. 47,037 verified smart contracts running on Ethereum were collected from the Etherscan website, from among which 17 auction contracts were selected. These selected contracts are actively in use, impacting many real users. For example, CryptoRomeAuction, hotPotatoAuction, and Deed are used to support popular DApps, and Deed has more than 1,469,061 transactions. In fact, the number of DApps on Ethereum is small (2.7K), compared with the total number of contract instances. It includes 12 distinct auction contracts and five voting smart contracts.

To find counterexamples, some configurations on mechanism models to be checked were set. For the auction mechanism model, there are three bidders, and the bid price and the valuation of bidders are arbitrary while allocation will be for one winner only. Similarly, for the voting mechanism model, it was assumed that five voters vote for two proposals as the basic configuration. A voter votes to any of the proposals randomly, and his ballot could be reflected in the bid in the mechanism model. The voter has his own valuation for different proposals, and the winning proposal means the allocation. The actual valuation of a winning proposal or failing proposal is the sum of the voters' valuations of that proposal. On Ethereum, voting contracts are open to users, and it is assumed that a voter cannot get any utility if the proposal supported by the voter is not the winning proposal. The valuation of a proposal by a voter could be measured in two ways. The first is that the valuation is mapped to a real number. For instance, the voter may prefer proposal A much more than any other voters prefer A. That is the situation where voters are heterogeneous. The second setting assigns 0 or 1 to the valuation by the voter of a proposal. For instance, the voter wants proposal A rather than proposal B. This simplified version could be applied to the situation where the voters are homogeneous. Under these settings, FairCon checks the four fairness properties for a given number of participants, aiming to find counterexamples.

With the configurations for mechanism models, 6 human hours were spent to manually annotate mechanism components and instrument the harness in these smart contracts.

Our experiments were conducted on Ubuntu 18.04.3 LTS desktop equipped with Intel Core i7 16-core and 32GB memory.

Table 2: Fairness checking on auction contracts.

Contracts	Properties				Time (seconds)	
	T	C	O	E	t_{model}	t_{check}
Auction1	X	X	X	X	7.96	0.11
Auction2	X	X	X	X	6.04	0.08
Auction3	X	X	X	X	2.34	0.08
AuctionItem	X	X	✓	X	1.29	0.08
AuctionManager	X	X	X	X	1.61	0.10
AuctionMultipleGuaranteed	X	X	X	X	7.48	0.11
AuctionPotato	X	X	X	X	2.45	0.07
BetterAuction	X	X	✓	X	1.58	0.08
CryptoRomeAuction	X	X	X	X	7.94	0.08
Deed	✓	✓	X	✓	14.25	0.07
EtherAuction	✓	✓	X	X	8.43	0.08
hotPotatoAuction	X	X	X	X	5.48	0.09

5

Table 3: Fairness proving on fair auction contracts in Table 2.

Contracts	Allocation Inv.	Price Inv.	Proved Property
AuctionItem	TopBidder	1st-Price	O
Deed	TopBidder	2nd-Price	T, C, E
EtherAuction	N/A	N/A	–
BetterAuction	TopBidder	1st-Price	O

Experiment Results

- 10 FairCon is evaluated by the selected 17 smart contracts with the configurations mentioned above. Tables 2 and 4 show the results for fairness checking on auction and voting contracts, respectively. In Table 2, the first column shows the names of the contracts, and the middle four columns show the result for the four fairness properties: truthfulness (T), collusion-freeness (C), optimality (O), and efficiency (E), respectively. The rightmost
- 15 column indicates the time for mechanism model extraction (t_{model}) and for fairness property checking (t_{check}). Among the selected 12 contracts, four of them are found to be

fair on at least one fairness property, while the remaining eight contracts are not fair for all the four fairness properties (with counterexamples generated). The generated counterexamples were checked and confirmed not to be false positives. Regarding the execution time, in the three-bidders experiments, model extraction time varied from 1.61 to 14.25 seconds because different contracts have different mechanism models to be extracted. Property checking is much faster than model extraction, which took around 0.1 seconds for each contract.

Table 3 shows the result of proving fairness properties for the 4 auction contracts that are fair on at least one fairness property, as shown in Table 2. The first column shows name of contract. The second and third columns show the invariant templates that are valid for proving fairness properties. The last column indicates which fairness property can be proved (T for truthfulness, C for collusion-freeness, O for optimality, and E for efficiency). The AuctionItem and BetterAuction contracts can be proved to satisfy the optimality property using the allocation invariant "TopBidder" together with the price invariant "1st-Price", which also confirms that they are first price auctions. The Deed contract satisfies the "TopBidder" and "2nd-Price" invariants, based on which, FairCon can prove three properties for Deed, namely, truthfulness, collusion-freeness, and efficiency. It also confirms that Deed is a second price auction.

The EtherAuction contract is shown in Figure 7, which is a variant of second price auction for a designated bid price only. Line 13 requires a fixed new higher bid price to update the four variables SecondHighestBid, SecondHighestBiddder, HighestBid, and HighestBiddder in Lines 15-18, respectively. It turned out that none of the predefined invariants are valid to prove any fairness property. This is because EtherAuction adopts the strategy of fixed bid price for each round, which makes it similar to (but actually not) typical second price auctions.

Table 4 shows the result of property checking for the selected five voting smart contracts, each of which is for five voters and two proposals. The first column shows the names of contracts. The last column, t_{model} , shows the model extraction time (in seconds). The middle two large columns show the average property checking time (in seconds) for the three properties: truthfulness (T), collusion-freeness (C) and efficiency (E). There are two settings for the valuation component in our mechanism model. One is ranging over real numbers $\mathbb{R}$, while the other is ranging over $\{0, 1\}$. The reason for having two settings is that no contract was found fair regarding any property, as shows in the second large column of Table 4, because the diverse $\mathbb{R}$ valuation of proposals brings the incentive for voters to lie and to conspire with others. In the $\{0, 1\}$ valuation setting, as shown in the

third large column of Table 4, all the five contracts are truthful, collusion-free, and efficient. This is because, if a voter lies, he/she gets at most zero worth utility, and thus has no incentive to lie. Based on Table 4, it can be observed that fairness property may depend on the configuration of mechanism models. Different configurations may have different results on fairness property checking.

The checking time for the optimality property is not listed in Table 4 because smart contracts for voting do not have the component of transfer functions in our mechanism model so that optimality cannot be defined. In addition, none of the predefined invariants are valid to prove that the five selected voting contracts are fair.

Since there is a lack of ground truth, the results of FairCon were confirmed by manual inspection.

15 Efficiency

The time costs can be divided into two parts: model extraction and property checking. Overall, the model extraction takes a much longer time and the property checking is efficient (taking less than one second for each case), as shown in Tables 2 and 4.

Table 4: Fairness checking on voting contracts.

Contracts	Valuation: $\mathbb{R}$				Valuation: $\{0, 1\}$				t_{model}
	T	C	E	t_{check}	T	C	E	t_{check}	
Association	X	X	X	0.35	✓	✓	✓	0.37	64.96
Ballot	X	X	X	0.45	✓	✓	✓	0.81	69.73
Ballot-doc	X	X	X	0.48	✓	✓	✓	0.56	126.14
HIDERA	X	X	X	0.12	✓	✓	✓	0.15	52.23
SBIBank	X	X	X	0.27	✓	✓	✓	0.69	56.59

To explore the efficiency of FairCon further, the CryptoRomeAuction contract was selected to make performance experiments on FairCon. Figures 8 and 9 show the execution time for mechanism model extraction and fairness property checking when the number of bidders increases, respectively. In Figure 8, the x-axis shows the number of bidders, while the y-axis shows the mechanism model extraction time in seconds. It can be observed that model extraction time is nearly exponential in the number of bidders involved, which is reasonable because every participant is independent. When the number of bidders is

under six, the model extraction time is less than 10 minutes, which is tolerable. Once the number of bidders goes beyond six, the time increases exponentially.

Figure 9 shows the property checking time, where the x-axis indicates the number of bidders, and the y-axis indicates the property checking time in seconds. The same trend is observed as that in Figure 8, i.e., the execution time is exponential in the number of bidders involved. However, property checking is much faster than model extraction since model extraction requires symbolic execution, which is heavy in computation. Mostly the checking time is less than one second. It is noted that the checking time for truthfulness or collusion-freeness is at least doubled than that for optimality and efficiency. This is because truthfulness and collusion-freeness properties need to consider the strategy as well as the outcome spaces, while optimality and efficiency properties only have to consider the outcome space.

Running FairCon on a large number (i.e., k , which is theoretically unbounded) of players is impractical for symbolic execution. The results indicate that FairCon can find counterexample(s) with small k (e.g., three to five), if the contract is indeed unfair (e.g., $k = 3$ in Table 2 and $k = 5$ in Table 4). Otherwise, as shown in Table 3, FairCon could use invariants observed during the small k runs in an attempt to prove that the fairness property holds for arbitrary values of k .

Although model extraction is the bottleneck of FairCon, it is a one-time task and need not be performed for a large k . FairCon is efficient for fairness property checking.

Through the verification results of FairCon, the following observations were made. Fairness issues in smart contracts are real. For example, EtherAuction (as shown in Figure 7) has been reported as a scam, where bidders compete for one Ether by gradually increasing their bids. The results confirmed its unfairness: although truthfulness and collusion-freeness holds for a bounded k , optimality and efficiency are violated. Based on our review of the subject contracts, some patterns observed are described below.

- (1) Contracts implementing the first price auction and their variants do not satisfy the truthfulness property. For example, the aforementioned BetterAuction is implementing a typical open first price auction, where the top bidder has the incentive to lower his/her bid price but still remain the winner.
- (2) Contracts implementing the first price auction and variants do not prevent against collusion. For example, BetterAuction does not satisfy the collusion-freeness property, since two bidders have the chance to lower the clear price and to be the winner, increasing their group utility.

(3) Contracts implementing the second price auction and their variants do not satisfy the optimality property. For example, Deed is one of the contracts implementing the second price auction. Since the clear price is the second highest bid, the contract may not be optimal with a potential decrease in total revenue.

- 5 (4) Contracts implementing the first price auction and their variants are not efficient. This is because first price auctions are untruthful, and the winner may not be the one who has the highest valuation of the item.

Turning now to Figure 10, an exemplary computer system architecture for implementing
10 embodiments of the present disclosure will be described.

The components of the computer system 1000 can be configured in a variety of ways. The components can be implemented entirely by software to be executed on standard computer server hardware, which may comprise one hardware unit or different computer
15 hardware units distributed over various locations, which may communicate over a network. Some of the components or parts thereof may also be implemented by application specific integrated circuits (ASICs) or field programmable gate arrays.

In the example shown in Figure 10, the computer system 1000 is a commercially available
20 server computer system based on a 32 bit or a 64 bit Intel architecture, and the processes and/or methods executed or performed by the computer system 1000 are implemented in the form of programming instructions of one or more software components or modules stored on non-volatile (e.g., hard disk) computer-readable storage 1024 associated with the computing device 1000. At least parts of the software modules 1022 could
25 alternatively be implemented as one or more dedicated hardware components, such as application-specific integrated circuits (ASICs) and/or field programmable gate arrays (FPGAs).

The computing device 1000 includes at least one or more of the following standard,
30 commercially available, computer components, all interconnected by a bus 1035:

- (a) random access memory (RAM) 1026;
- (b) at least one computer processor 1028, and
- (c) a network interface connector (NIC) 1030 which connects the computer device 1000 to a data communications network and/or to external devices.

35

The computing device 1000 may include a plurality of standard software modules, including an operating system (OS) 1036 (e.g., Linux or Microsoft Windows), and a number of other standard components, such as libraries and the like, used for implementation of

smart contracts (e.g., on Ethereum). The computing device 1000 also includes model extraction module 110, property checking module 120, and fairness verification module 130.

5 The boundaries between the modules and components shown in Figure 10 are exemplary, and alternative embodiments may merge modules or impose an alternative decomposition of functionality of modules. For example, the modules discussed herein may be decomposed into submodules to be executed as multiple computer processes, and, optionally, on multiple computers. Moreover, alternative embodiments may combine
10 multiple instances of a particular module or submodule. Furthermore, the operations may be combined or the functionality of the operations may be distributed in additional operations in accordance with the invention. Alternatively, such actions may be embodied in the structure of circuitry that implements such functionality, such as the micro-code of a complex instruction set computer (CISC), firmware programmed into programmable or
15 erasable/programmable devices, the configuration of a field-programmable gate array (FPGA), the design of a gate array or full-custom application-specific integrated circuit (ASIC), or the like.

Each part of the processes performed by the smart contract verification system 100 may
20 be executed by a module (of software modules 110, 120, 130) or a portion of a module. The processes may be embodied in a non-transient machine-readable and/or computer-readable medium for configuring a computer system to execute the method. The software modules may be stored within and/or transmitted to a computer system memory to configure the computer system to perform the functions of the module.

25 The computing device 1000 normally processes information according to a program (a list of internally stored instructions such as a particular application program and/or an operating system) and produces resultant output information via input/output (I/O) devices 1030. A computer process typically includes an executing (running) program or
30 portion of a program, current program values and state information, and the resources used by the operating system to manage the execution of the process. A parent process may spawn other, child processes to help perform the overall functionality of the parent process. Because the parent process specifically spawns the child processes to perform a portion of the overall functionality of the parent process, the functions performed by child
35 processes (and grandchild processes, etc.) may sometimes be described as being performed by the parent process.

Throughout this specification, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

5

The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

10

CLAIMS

1. A computer-implemented method of automatically verifying at least one fairness property of a smart contract, the method comprising:
 - 5 (i) performing symbolic execution on source code of the smart contract to extract a mechanism model,

wherein the mechanism model comprises a representation of processing of inputs from a finite number of users by the smart contract to produce an outcome;
 - 10 (ii) encoding the mechanism model and the at least one fairness property as a satisfiability modulo theories (SMT) formula; and
 - (iii) automatically determining, from the SMT formula, whether the at least one fairness property is satisfied for the finite number of users.
- 15 2. A computer-implemented method according to claim 1, wherein operation (iii) is carried out by an SMT solver.
3. A computer-implemented method according to claim 1 or claim 2, wherein the at least one fairness property is selected from one or more of: truthfulness, optimality, efficiency, and collusion-freeness.
- 20 4. A computer-implemented method according to any one of the preceding claims, wherein if the at least one fairness property is satisfied for the finite number of users, the method further comprises:
 - 25 (iv) obtaining, during said determining, one or more invariants of the mechanism model;
 - (v) instrumenting the source code of the smart contract with the one or more invariants to generate a modified smart contract, said one or more invariants being parametrized by an unlimited number of users; and
 - 30 (vi) automatically determining, from the modified smart contract, whether the at least one fairness property is satisfied for the unlimited number of users.
5. A computer-implemented method according to claim 4, wherein operation (vi) is carried out by a program verification tool.
- 35 6. A computer-implemented method according to claim 4 or claim 5, wherein the one or more invariants are obtained in accordance with one or more invariant templates.

7. A computer-implemented method according to any one of the preceding claims, comprising instrumenting the source code with a harness program that orchestrates interactions of the users with a target contract of said smart contract.
- 5 8. A computer-implemented method according to any one of the preceding claims, wherein the smart contract encodes an auction process or a voting process.
9. A system for automatically verifying at least one fairness property of a smart contract, the system comprising:
 - 10 a model extraction module that is configured to:
 - (i) perform symbolic execution on source code of the smart contract to extract a mechanism model,
 - wherein the mechanism model comprises a representation of processing of inputs from a finite number of users by the smart contract to produce an outcome; and
 - 15 (ii) encode the mechanism model and the at least one fairness property as a satisfiability modulo theories (SMT) formula; and
 - a property checking module that is configured to:
 - (iii) automatically determine, from the SMT formula, whether the at
 - 20 least one fairness property is satisfied for the finite number of users.
 10. A system according to claim 9, wherein the property checking module is configured to carry out operation (iii) by an SMT solver.
 - 25 11. A system according to claim 9 or claim 10, wherein the at least one fairness property is selected from one or more of: truthfulness, optimality, efficiency, and collusion-freeness.
 12. A system according to any one of claims 9 to 11, further comprising a fairness verification module that is configured to execute in response to the at least one fairness property being satisfied for the finite number of users, the fairness verification module further being configured to:
 - (iv) obtain, during operation (iii), one or more invariants of the mechanism model;
 - 35 (v) instrument the source code of the smart contract with the one or more invariants to generate a modified smart contract, said one or more invariants being parametrized by an unlimited number of users; and
 - (vi) automatically determine, from the modified smart contract, whether the at least one fairness property is satisfied for the unlimited number of users.

13. A system according to claim 12, wherein the fairness verification module is configured to carry out operation (vi) by a program verification tool.
- 5 14. A system according to claim 12 or claim 13, wherein the fairness verification module is configured to obtain the one or more invariants in accordance with one or more invariant templates.
- 10 15. A system according to any one of claims 9 to 14, wherein the source code is instrumented with a harness program that orchestrates interactions of the users with a target contract of said smart contract.
- 15 16. A system according to any one of claims 9 to 15, wherein the smart contract encodes an auction process or a voting process.
17. Non-transitory computer-readable storage having stored thereon machine-readable code for causing at least one processor to carry out a method according to any one of claims 1 to 8.