Game Theory on the Blockchain: A Model for Games with Smart Contracts

Mathias Hall-Andersen and Nikolaj I. Schwartzbach(✉)

Department of Computer Science, Aarhus University, Aarhus, Denmark
{ma,nis}@cs.au.dk

Abstract. We propose a model for games in which the players have shared access to a blockchain that allows them to deploy smart contracts to act on their behalf. This changes fundamental game-theoretic assumptions about rationality since a contract can commit a player to act irrationally in specific subgames, making credible otherwise non-credible threats. This is further complicated by considering the interaction between multiple contracts which can reason about each other. This changes the nature of the game in a nontrivial way as choosing which contract to play can itself be considered a move in the game. Our model generalizes known notions of equilibria, with a single contract being equivalent to a Stackelberg equilibrium, and two contracts being equivalent to a reverse Stackelberg equilibrium. We prove a number of bounds on the complexity of computing SPE in such games with smart contracts. We show that computing an SPE is PSPACE-hard in the general case. Specifically, in games with k contracts, we show that computing an SPE is Σ_k^{P}-hard for games of imperfect information. We show that computing an SPE remains PSPACE-hard in games of perfect information if we allow for an unbounded number of contracts. We give an algorithm for computing an SPE in two-contract games of perfect information that runs in time $O(m\ell)$ where m is the size of the game tree and ℓ is the number of terminal nodes. Finally, we conjecture the problem to be NP-complete for three contracts.

1 Introduction

This paper is motivated by the games that arise on permissionless blockchains such as Ethereum [22] that offer "smart contract" functionality: in these permissionless systems, parties can deploy smart contracts without prior authorization by buying the "tokens" required to execute the contract. By smart contracts, we mean arbitrary pieces of code written in a Turing-complete language[1] capable of maintaining state (including funds) and interact with other smart contracts by invoking methods on them. Essentially, smart contracts are objects in the Java sense. Parties can also invoke methods on the smart contracts manually. Note that the state of all smart contracts is public and can be inspected by any party

[1] However the running time of the contracts is limited by the execution environment.

I. Caragiannis and K. A. Hansen (Eds.): SAGT 2021, LNCS 12885, pp. 156–170, 2021.
https://doi.org/10.1007/978-3-030-85947-3_11

at any time. This changes fundamental game-theoretic assumptions about rationality: in particular, it might be rational for a player to deploy a contract that commits them to act irrationally in certain situations to make credible otherwise non-credible threats. This gives rise to very complex games in which parties can commit to strategies, that in turn depend upon other players' committed strategies. Reasoning about such equilibria is important when considering games that are meant to be played on a blockchain, since the players - at least in principle - always have the option of deploying such contracts. In the literature, this is known as a Stackelberg equilibrium where a designated leader commits to a strategy before playing the game. In general, because of first-mover advantage, being able to deploy a contract first is never a disadvantage, since a player can choose to deploy the empty contract that commits them to nothing. It is well-known that it is hard to compute the Stackelberg equilibrium in the general case [12], though much less is known about the complexity when there are several of these contracts in play: when there are two contracts, the first contract can depend on the second contract in what is known as a reverse Stackelberg equilibrium [2,9,21]. This is again strictly advantageous for the leader since they can punish the follower for choosing the wrong strategy. In this paper, we present a model that generalizes (reverse) Stackelberg games, that we believe captures these types of games and which may be of wider interest. In practical terms, we believe that our model is of interest when analyzing distributed systems for "game-theoretic security" in settings where the players naturally have the ability to deploy smart contracts. Potential examples include proof-of-stake blockchains themselves and financial applications that build upon these systems.

Contracts	Players	Information	Strategies	Lower bound	Upper bound
0	2	perfect	pure	P-hard [20]	$O(m)$ [16]
0	2	imperfect	mixed	PPAD-complete [7,6]	
1	2	perfect	pure	P-hard [20]	$O(m\ell)$ [4]
1	2	perfect	mixed	NP-complete [13]	
1	2	imperfect	-	NP-complete [13]	
2	2	perfect	pure	P-hard [20]	$O(m\ell)$ [Theorem 3]
3	3	perfect	pure	Conjectured NP-hard	NP [Theorem 3]
k	$2+k$	imperfect	pure	Σ_k^p-hard [Theorem 2]	?
unbounded	-	perfect	pure	PSPACE-hard [Theorem 4]	?

Fig. 1. An overview of some existing bounds on the complexity of computing an SPE in extensive-form games and where our results fit in. Here, m is the size of the tree, and ℓ is the number of terminal nodes.

Our Results. We propose a game-theoretic model for games in which players have shared access to a blockchain that allows the players to deploy smart contracts to act on their behalf in the games. Allowing a player to deploy a smart contract corresponds to that player making a 'cut' in the tree, inducing a new expanded game of exponential size containing as subgames all possible cuts in the game. We show that many settings from the literature on Stackelberg games can be recovered as special cases of our model, with one contract being equivalent to a Stackelberg equilibrium, and two contracts being equivalent to a reverse Stackelberg equilibrium. We prove bounds on the complexity of computing an SPE in these expanded trees. We prove a lower bound, showing that computing an SPE in games of imperfect information with k contracts is Σ_k^{P}-hard by reduction from the true quantified Boolean formula problem. For $k = 1$, it is easy to see that a contract can be verified in linear time, establishing NP-completeness. In general, we conjecture Σ_k^{P}-completeness for games with k contracts, though this turns out to reduce to whether or not contracts can be described in polynomial space. For games of perfect information with an unbounded number of contracts, we also establish PSPACE-hardness from a generalization of 3-COLORING. We show an upper bound for $k = 2$ and perfect information, namely that computing an SPE in a two-contract game of size m with ℓ terminal nodes (and any number of players) can be computed in time $O(m\ell)$. For $k = 3$, the problem is clearly in NP since we can verify a witness using the algorithm for $k = 2$, and we conjecture the problem to be NP-complete. Finally, we discuss various extensions to the model proposed and leave a number of open questions.

2 Games with Smart Contracts

In this section, we give our model of games with smart contracts. We mostly assume familiarity with game theory and refer to [16] for more details. For simplicity of exposition, we only consider a somewhat restricted class of games, namely finite games in extensive form, and consider only pure strategies in these games. In addition, we will assume games are in *generic form*, meaning the utilities of all players are unique. This has the effect that the resulting subgame perfect equilibrium is unique. Equivalently, we use a tie breaking algorithm to decide among the different subgame perfect equilibria, and slightly perturb the utilities of the players to match the subgame perfect equilibrium chosen by the tie breaker.

Formally, an *extensive-form game* G is a finite tree T. We denote by $L \subseteq T$ the set of leaves in T, i.e. nodes with no children, and let m denote the number of nodes in T. Each leaf ℓ is labeled by a vector $u(\ell) \in \mathbb{R}^n$ that denotes the utility $u_i(\ell)$ obtained by party P_i when terminating in the leaf ℓ. In addition, the game consists of a finite set of n players. We consider a fixed partition of the non-leaves into n sets, one for each player. The game is played by starting at the root, letting the player who owns that node choose a child to recurse into, this is called a move. We proceed in this fashion until we reach a leaf and distribute its utility vector to the players. When there is perfect information, a player always knows exactly which subgame they are playing, though more generally we may consider a partition of the non-leafs into *information sets*, where each player is only told the information set to which their node belongs.

When all information sets are singletons we say the game has perfect information. The players are assumed to be *rational*, that is they choose moves to maximize their utility: we say a strategy for each player (a strategy profile) constitutes a *(Nash) equilibrium* if no unilateral deviation by any party results in higher utility for that party. Knowing the other players are rational, for games of perfect information, at each branch a player can anticipate their utility from each of its moves by recursively determining the moves of the other parties. This process is called *backward induction*, and the resulting strategy profile is a *subgame perfect equilibrium*. A strategy profile is an SPE if it is an equilibrium for every subgame of the game. For games of perfect information, computing the SPE takes linear time in the size of the tree and can be shown to be P-complete [20]. Later, we will show a lower bound, namely that adding a contract to the tree moves this computation up (at least) a level in the polynomial hierarchy. Specifically, we show that computing the SPE in k-contract games is Σ_k^{P}-hard in the general case with imperfect information.

2.1 Smart Contract Moves

We now give our definition of smart contracts in the context of finite games. We add a new type of node to our model of games, a *smart contract move*. Intuitively, whenever a player has a smart contract move, they can deploy a contract that acts on their behalf for the rest of the game. The set of all such contracts is countably infinite, but fortunately, we can simplify the problem by considering equivalence classes of contracts which "do the same thing". Essentially, the only information relevant to other players is whether or not a given action is still possible to play: it is only if the contract dictates that a certain action cannot be played, that we can assume a rational player will not play it. In particular, any contract which does not restrict the moves of a player is equivalent to the player not having a contract. Such a restriction is called a *cut*. A cut $c^{(i)}$ for player P_i is defined to be a union of subtrees whose roots are children of P_i-nodes, such that: (1) every node in $T \setminus c^{(i)}$ has a path going to a leaf; a cut is not allowed to destroy the game by removing all moves for a player, and (2) $c^{(i)}$ respects information sets, that is it 'cuts the same' from each node in the same information set.

In other words, deploying a smart contract corresponds to choosing a cut in the game tree. This means that a smart contract node for player P_i in a game T is essentially syntactic sugar for the *expanded tree* that results by applying the set of all cuts $c^{(i)}$ to T and connecting the resulting games with a new node belonging to P_i at the top. Computing the corresponding equilibrium with smart contracts then corresponds to the SPE in this expanded tree. Note that this tree is uniquely determined. See Fig. 2 for an example. We use the square symbol in figures to denote smart contract moves. When a game contains multiple smart contract moves, we expand the smart contract nodes recursively in a depth-first manner using the transformation described above.

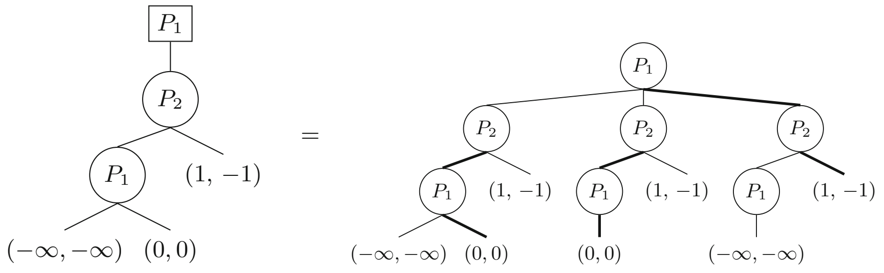

Fig. 2. Expanding a smart contract node for a simple game. The square symbol is a smart contract move for player P_1. We compute all P_1-cuts in the game and connect them with a node belonging to P_1. The first coordinate is the leader payoff, and the second is the follower payoff. The dominating paths are shown in bold. We see that the optimal strategy for P_1 is to commit to choosing $(-\infty, -\infty)$ unless P_2 chooses $(1, -1)$.

2.2 Contracts as Stackelberg Equilibria

As mentioned earlier, the idea to let a party commit to a strategy before playing the game is not a new one: in 1934, von Stackelberg proposed a model for the interaction of two business firms with a designated market leader [18]. The market leader holds a dominant position and is therefore allowed to commit to a strategy first, which is revealed to the follower who subsequently decides a strategy. The resulting equilibrium is called a Stackelberg equilibrium. In this section we show that the Stackelberg equilibrium for a game with leader P_1 and follower P_2 can be recovered as a special case of our model where P_1 has a smart contract. We use the definition of strong Stackelberg equilibria from [5,11]. We note that since the games are assumed to be in generic form, the follower always has a unique response, thus making the requirement that the follower break ties in favor of the leader unnecessary.

Let T be a game tree. A *path* $\mathbf{p} \subseteq T$ is a sequence of nodes such that for each j, $\mathbf{p}_{j+1}$ is a child of $\mathbf{p}_j$. If $\mathbf{p}$ is a path, we denote by $\mathbf{p}^{(i)} \subseteq \mathbf{p}$ the subset of nodes owned by player P_i. Now suppose T has a horizon of h. We let $\mathfrak{p} = (\mathfrak{p}_j)_{j=1}^{h} \subseteq T$ denote the *dominating path* of the game defined as the path going from the root $\mathfrak{p}_1$ to the terminating leaf $\mathfrak{p}_h$ in the SPE of the game.

Definition 1. *Let $i \in [n]$ be the index of a player, and let $f(s_i)$ be the best response to s_i for players other than P_i. We say $(s_i^*, f(s_i^*))$ is a* Stackelberg equilibrium with leader P_i *if the following properties hold true:*

– Leader optimality. *For every leader strategy s_i, $u_i(s_i^*, f(s_i^*)) \geq u_i(s_i, f(s_i))$.*
– Follower best response. *For every $j \neq i$, and every s_{-i}, $u_j(s_i^*, f(s_i^*)) \geq u_j(s_i^*, s_{-i})$.* ◇

Proposition 1. *The Stackelberg equilibrium with leader P_i is equivalent to P_i having a smart contract move.*

Proof. We show each implication separately:

$\Rightarrow$ SPE in the expanded tree T induces a Stackelberg equilibrium in the corresponding Stackelberg game where P_i commits to all moves in $\mathfrak{p}^{(i)}$. It is not hard to see that the follower best response $f(s_i^*)$ is defined by the SPE of the subgame arising after P_i makes the move $\mathfrak{p}_1$ choosing the contract in T.

$\Leftarrow$ A Stackelberg equilibrium induces a SPE in the expanded tree T with the same utility: let $(s_i^*, f(s_i^*))$ be a Stackelberg equilibrium, observe that s_i^* corresponds to a cut $c^{(i)} \subseteq T$ where P_i cuts away all nodes in T not dictated by s_i^*. By letting the first move $\mathfrak{p}_1$ of P_i correspond to $c^{(i)}$, the best follower response $f(s_i^*)$ is the SPE in the resulting subgame, and hence $u(\mathfrak{p}) = u(s_i^*, f(s_i^*))$. $\square$

Multi-leader/multi-follower Contracts. Several variants of the basic Stackelberg game has been considered in the literature with multiple leaders and/or followers [14,17]. We can model this using smart contracts by forcing some of the contracts to independent of each other: formally, we say a contract is *independent* if it makes the same cut in all subgames corresponding to different contracts. It is not hard to see that multiple leaders can be modelled by adding contracts for each leader, where the contracts are forced to be independent. $\diamond$

Reverse Stackelberg Contracts. The reverse Stackelberg equilibrium is an attempt to generalize the regular Stackelberg equilibrium: here, the leader does not commit to a specific strategy *a priori*, rather they provide the follower with a mapping f from follower actions to best response leader actions, see e.g. [1,19] for a definition in the continuous setting. When the follower plays a strategy s_{-i}, the leader plays $f(s_{-i})$. This is strictly advantageous for the leader since as pointed out in [9], they can punish the follower for choosing the wrong strategy.

In the following, if $\mathbf{p}$ is a path of length ℓ, we denote by $G_s(\mathbf{p})$ the subgame whose root is $\mathbf{p}_\ell$.

Definition 2. *Let i be the index of the leader, and $-i$ the index of the follower. We say $(f(s_{-i}^*), s_{-i}^*)$ is a* reverse Stackelberg equilibrium with leader i *if the following holds for every leader strategy s_i and follower strategy s_{-i}, it holds:*

- Leader best response*: $u_i(f(s_{-i}^*), s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$.*
- Follower optimality*: $u_{-i}(f(s_{-i}^*), s_{-i}^*) \geq u_{-i}(f(s_{-i}), s_{-i})$.* $\diamond$

Proposition 2. *The reverse Stackelberg equilibrium for a two-player game with leader P_i is equivalent to adding two smart contract moves to the game, one for P_i, and another for P_{-i} (in that order).*

Proof. We show each implication separately:

$\Rightarrow$ The SPE in the expanded tree induces a reverse Stackelberg equilibrium: for every possible follower strategy s_{-i}, we define $f(s_{-i})$ as the leader strategy in the SPE in the subgame $G_s(\langle \mathfrak{p}_1, s_{-i} \rangle)$ after the two moves, where we slightly abuse notation to let s_{-i} mean that P_{-i} chooses a cut where their SPE is

s_{-i}. Leader best response follows from the observation that $\mathfrak{p}_1$ corresponds to the optimal set of cuts of P_i moves in response to every possible cut of P_{-i} moves.

$\Leftarrow$ A reverse Stackelberg equilibrium induces an SPE in the expanded tree: let $(f(s^*_{-i}), s^*_{-i})$ be a reverse Stackelberg equilibrium and let f be the strategy of P_i in the reverse Stackelberg game, then P_i has a strategy in the two-contract game with the same utility for both players: namely, P_i's first move is choosing the subgame in which for every second move s_{-i} by P_{-i} they make the cut $f(s_{-i})$. $\square$

3 Computational Complexity

Having defined our model of games with smart contracts, in this section we study the computational complexity of computing equilibria in such games. Note that we can always compute the equilibrium by constructing the expanded tree and performing backward induction in linear time. The problem is that the expanded tree is very large: the expanded tree for a game of size m with a single contract has $2^{O(m)}$ nodes since it contains all possible cuts. For every contract we add, the complexity grows exponentially. This establishes the rather crude upper bound of Σ_k^{EXP} for computing SPE in games with perfect information and k contracts. The question we ask if we can do better than traversing the entire expanded tree.

In terms of feasibility, our results are mostly negative: we show a lower bound that computing an SPE, in general, is infeasible for games with smart contracts. We start by considering the case of imperfect information where information sets allow for a rather straightforward reduction from $\mathsf{CircuitSAT}$ to games with one contract, showing NP-completeness for single-contract games of imperfect information. This generalizes naturally to the k true quantified Boolean formula problem (k-TQBF), establishing Σ_k^{P}-hardness for games of imperfect information with k contracts. On the positive side, we consider games of perfect information where we provide an algorithm for games and two contracts that runs in time $O(m\ell)$. However, when we allow for an unbounded number of contracts, we show the problem remains PSPACE-complete by reduction from the generalization of 3-COLORING described in [3]. We conjecture the problem to be NP-complete for three contracts.

3.1 Games with Imperfect Information, NP-Completeness

We start by showing NP-completeness for games of imperfect information by reduction from $\mathsf{CircuitSAT}$. We consider a decision problem version of SPE: namely, whether or not a designated player can obtain a utility greater than the target value.

Reduction. Let C be an instance of $\mathsf{CircuitSAT}$. Note that we can start from any complete basis of Boolean functions, so it suffices to suppose the circuit C consists only of NAND with fanin 2 and fanout 1. We will now construct a game tree for the circuit: we will be using one player to model the assignment of variables, say player 1. The game starts with a contract move for player 1 who can assign values to variables by cutting the bottom of the tree: we construct the game such that player 1 only has moves in the bottom level of the tree. In this way, we ensure that every cut corresponds to assigning truth values to the variables. We adopt the convention that a payoff of 1 for player 1 is *true* ($\top$), while a payoff of 0 for player 1 is *false* ($\bot$). All nodes corresponding to occurrences of the same variable get grouped into the same information set, which enforces the property that all occurrences of the same variable must be assigned the same value (Fig. 3).

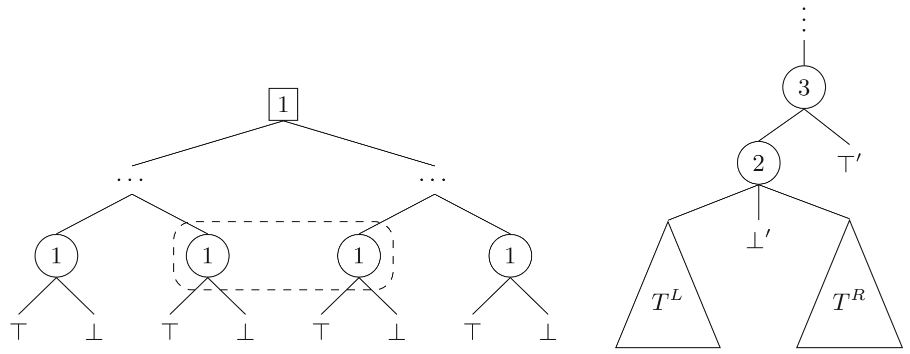

Fig. 3. The basic structure of the reduction. Player 1 has a smart contract that can be used to assign values to the variables. The dashed rectangle denotes an information set and is used when there are multiple occurrences of a variable in the circuit. On the right, we see the NAND-gate gadget connecting the left subgame T^L and the right subgame T^R. We implement the gadget by instantiating the utility vectors such that player 2 chooses $\bot'$ if only if both T^L and T^R propagate a utility vector encoding true.

For the NAND-gate, we proceed using induction: let T^L, T^R be the trees obtained by induction, we now wish to construct a game tree gadget with NAND-gate logic. To do this we require two players which we call player 2 and player 3. Essentially, player 2 does the logic, and player 3 converts the signal to the right format. The game tree will contain multiple different utility vectors encoding true and false, which vary their utilities for players 2 and 3. Each NAND-gate has a left tree and a right tree, each with their own utilities for true and false: $\bot^L, \bot^R; \top^L, \top^R$. The gadget starts with a move for player 3 who can choose to continue the game, or end the game with a true value $\top'$. If they continue the game, player 2 has a choice between false $\bot'$ or playing either T^L or T^R. To make the gadget work like a NAND-gate we need to instantiate the utilities to make backward induction simulate its logic. The idea is to make player 2 prefer both $\bot^L$ and $\bot^R$ to $\bot'$, which they, in turn, prefer to $\top^L$ and $\top^R$. As a result, player 2 propagates $\bot'$ only if both T^L, T^R are true, otherwise, it propagates $\bot^L$ or $\bot^R$. Finally, we must have that player 3 prefers $\top'$ to both $\bot^L$ and $\bot^R$, while they prefer $\bot'$ to $\top', \top^L$ and $\top^R$. This gives rise to a series of inequalities:

$$\begin{array}{llll} \bot_2^L > \bot_2' > \top_2^L & \top_3' > \bot_3^L & \bot_3' > \top_3^L & \bot_3' > \top_3' \\ \bot_2^R > \bot_2' > \top_2^R & \top_3' > \bot_3^R & \bot_3' > \top_3^R & \end{array}$$

We can instantiate this by defining $\top, \bot$. For the base case corresponding to a leaf, we let $\bot = (0,1,0), \top = (1,0,0)$. We then define recursively:

$$\top' = \big(1, 0, 1 + \max(\top_3^L, \top_3^R)\big)$$
$$\bot' = \left(0, \frac{\min(\bot_2^L, \bot_2^R) + \max(\top_2^L, \top_2^R)}{2}, 2 + \max\left(\top_3^L, \top_3^R\right)\right)$$

It is not hard to verify that these definitions make the above inequalities hold true. As a result, the gadget will propagate a utility vector corresponding to true if and only if not both subtrees propagate true.

Theorem 1. *Computing an SPE in three-player single-contract games of imperfect information is* NP*-complete.*

Proof. We consider the decision problem of determining whether or not in the SPE, player 1 has a utility of 1. By construction of the information sets, any strategy is a consistent assignment of the variables. It now follows that player 1 can get a payoff > 0 if and only if there is an assignment of the variables such that the output of the circuit is true. This shows NP-hardness. Now, it is easily seen that this problem is in NP, since a witness is simply a cut that can be verified in linear time in the size of the tree. Completeness now follows using our reduction from $\mathsf{CircuitSAT}$. $\square$

Remark 1. Our reduction also applies to the two-player non-contract case by a reduction from circuit value problem. This can be done in logspace since all the gadgets are local replacements. In doing so, we reestablish the result of [20], showing that computing an SPE on two-player games is P-complete. $\diamond$

3.2 Games with Imperfect Information, PSPACE-Hardness

In this section, we show that computing the SPE in a game with k contract moves is Σ_k^{P}-complete, in the general case with imperfect information. Generalizing the previous result of NP-hardness to k contracts is fairly straightforward. Our claim is that the resulting decision problem is Σ_k^{P}-hard so we obtain a series of hardness results for the polynomial hierarchy. This is similar to the results obtained in [10] where the value problem for a competitive analysis with $k+1$ players is shown to be hard for Σ_k^{P}.

Formally, we consider the following decision problem with target value V for a game tree T with k contract players: let T' be the expanded tree with contracts for players $P_1, P_2, \dots P_k$ in ascending order. Can player P_1 make a cut in T' such that their payoff is $\geq V$?

To show our claim, we proceed using reduction from the canonical Σ_k^{P}-complete problem k-TQBF, see e.g. [8] for a formal definition.

Theorem 2. *Computing an SPE in* $2+k$ *player games of imperfect information is* Σ_k^{P}*-hard.*

Proof (sketch). We extend our reduction from Theorem 1 naturally to the quantified satisfiability problem. In our previous reduction, the contract player wanted to satisfy the circuit by cutting as to assign values to the variables in the formula. Now, for each quantifier in ψ, we add a new player with a contract, whose moves range over exactly the variables quantified over. The players have contracts in the same order specified by their quantifiers. The idea is that players corresponding to $\forall$ try to sabotage the satisfiability of the circuit, while those corresponding to $\exists$ try to ensure satisfiability. We encode this in the utility vectors by giving $\exists$-players a utility of 1 in $\top$ and 0 utility in $\bot$, while for the $\forall$-players, it is the opposite. It is not hard to see that ψ is true, only if P_1 can make a cut, such that for every cut P_2 makes, there exists a cut for P_3 such that, ..., the utility of P_1 is 1. This establishes our reduction. $\square$

Remark 2. We remark that it is not obvious whether or not the corresponding decision problem is contained within Σ_k^{P}. It is not hard to see we can write a Boolean formula equivalent to the smart contract game in a similar manner as with a single contract. The problem is that it is unclear if the innermost predicate ϕ can be computed in polynomial-time. It is not hard to see that some smart contracts do not have a polynomial description, i.e. we can encode a string $x \in \{0,1\}^*$ of exponential length in the contract. However, there might be an equivalent contract that *does* have a polynomial-time description. By equivalent, we mean one that has the same dominating path. This means that whether or not Σ_k^{P} is also an upper bound essentially boils down to whether or not every contract has an equivalent contract with a polynomial description. $\diamond$

3.3 Games with Perfect Information, Two Contracts, Upper Bound

In this section, we consider two-player games of perfect information and provide a polynomial-time algorithm for computing an SPE in these games. Specifically, for a game tree of size m with ℓ terminal nodes with two contract players (and an arbitrary number of non-contract players), we can compute the equilibrium in time $O(m\ell)$. Our approach is similar to that of [15], in that we compute the inducible region for the first player, defined as the set of leaves they are able to 'induce' by making cuts in the game tree.

Let A, B be two sets. We then define the set of outcomes from A reachable using a threat against player i from outcomes in B as follows:

$$\text{threaten}_i(A, B) = \{x \in A \mid \exists\, y \in B.\, x_i > y_i\}$$

As mentioned, we will compute the *inducible region* for the player with the first contract, defined as the set of outcomes reachable with a contract. Choosing the optimal contract is then reduced to a supremum over this region.

Definition 3. *Let G be a fixed game. We denote by $\mathscr{R}(P_1)$ (resp. $\mathscr{R}(P_1, P_2)$) the* inducible region *of P_1, defined as the set of outcomes reachable by making a cut in G in all nodes owned by P_1. $\mathscr{R}(P_1)$ is a tuple $(\mathbf{u}, c_1)$ where $\mathbf{u} \in \mathbb{R}^n$ is the utility vector, and c_1 is the contract (a cut) of P_i.* $\diamond$

Algorithm. Let G be the game tree in question and let k be a fixed integer. As mentioned, we assume without loss of generality that G is in *generic form*, meaning all non-leafs in G have out-degree exactly two and that all utilities for a given player are distinct such that the ordering of utilities is unique. We denote by P_1, P_2 the players with contracts and assume that P_i has the i^{th} contract. We will compute the inducible regions in G for P_1 (denoted S for *self*), and for (P_1, P_2) (denoted T for *together*) by a single recursive pass of the tree. In the base case with a single leaf with label $\mathbf{u}$ we have $S = T = \{\mathbf{u}\}$. For a non-leaf, we can recurse into left and right child, and join together the results. The procedure is detailed in Algorithm 1.

Algorithm 1: `InducibleRegion`(G)

```
switch $G$ do
    case Leaf($u$):
        return $(\{u\}, \{u\})$

    case Node($G^L, G^R, i$):
        $(S^L, T^L) \leftarrow$ InducibleRegion($G^L$)
        $(S^R, T^R) \leftarrow$ InducibleRegion($G^R$)
        if $i = 1$ then
            $T \leftarrow T^L \cup T^R$
            $S \leftarrow S^L \cup S^R \cup \text{threaten}_2(T^L \cup T^R, S^L \cup S^R)$
        else if $i = 2$ then
            $T \leftarrow T^L \cup T^R$
            $S \leftarrow \text{threaten}_2(T^L, S^R) \cup \text{threaten}_2(T^R, S^L)$
        else
            $T \leftarrow \text{threaten}_i(T^L, T^R) \cup \text{threaten}_i(T^R, T^L)$
            $S' \leftarrow \text{threaten}_i(S^L, S^R) \cup \text{threaten}_i(S^R, S^L)$
            $S \leftarrow S' \cup \text{threaten}_2(T, S')$
        end
        return $(S, T)$
end
```

Theorem 3. *An SPE in two-contract games of perfect information can be computed in time $O(m\ell)$.*

Proof. First, the runtime is clearly $O(m\ell)$ since the recursion has $O(m)$ steps where we need to maintain two sets of size at most ℓ. For correctness, we show something stronger: let $\mathscr{R}(P_1)$ be the inducible region for P_1 in the expanded tree and $\mathscr{R}(P_1, P_2)$ be the inducible region of (P_1, P_2). Now, let $(S, T) =$ `InducibleRegion`(G). Then we show that $S = \mathscr{R}(P_1)$ and $T = \mathscr{R}(P_1, P_2)$. This implies that $\text{argmax}_{u \in S}\, u_1$ is the SPE. The proof is by induction on the height h of the tree. As mentioned, we assume that games are in *generic form*. This base case is trivial so we consider only the inductive step.

Necessity follows using simple constructive arguments: for S and $i = 1$, then for every $(\mathbf{u}, c) \in S^{\ell}$, we can form contract where P_1 chooses left branch and plays c. And symmetrically for S^R. Similarly, for every $(\mathbf{u}, c_1, c_2) \in T^L$ and $(\mathbf{v}, c') \in S^L$ can form contract where P_1 plays c_1 in all subgames where P_2 plays c_2; and plays c' otherwise. Then $\mathbf{u}$ is dominating if and only if $\mathbf{u}_2 > \mathbf{v}_2$. Similar arguments hold for the remaining cases.

For sufficiency, we only show the case of $i = 1$ as the other cases are similar. Assume (for contradiction) that there exists $(\mathbf{u}, c_1) \in \mathscr{R}(P_1) \setminus S$, i.e. there is a P_1-cut c_1 such that $\mathbf{u}$ is dominating. Then,

$$\begin{aligned}(\mathbf{u}, c_1) &\in (T^L \cup T^R) \setminus (S^L \cup S^R \cup \text{threaten}_2(T^L \cup T^R, S^L \cup S^R))\\ &= \{\mathbf{v} \in (T^L \cup T^R) \setminus (S^L \cup S^R) \mid \forall \mathbf{v}' \in S^L \cup S^R .\, \mathbf{v}_2 < \mathbf{v}'_2\}\end{aligned}$$

That is, $\mathbf{u}$ must be a utility vector that P_1 and P_2 can only reach in cooperation in a one of the two sub-games, say by P_2 playing c_2. However, for every cut that P_1 makes, the dominating path has utility for P_2 that is $> \mathbf{u}_2$, meaning P_2 strictly benefits by not playing c_2. But this is a contradiction since we assumed $\mathbf{u}$ was dominating. □

3.4 Games with Perfect Information, Unbounded Contracts, PSPACE-Hardness

We now show that computing an SPE remains PSPACE-complete when considering games with an arbitrary number of contract players. We start by showing NP-hardness and generalize to PSPACE-hardness in a similar manner as we did for Theorem 2. The reduction is from 3-COLORING: let (V, E) be an instance of 3-COLORING and assume the colors are $\{R, G, B\}$. The intuition behind the NP-reduction is to designate a coloring player P_{color}, who picks colors for each vertex $u \in V$ by restricting his decision space in a corresponding move using a contract. They are the first player with a contract. This is constructed using a small stump for every edge $e \in E$ with three leaves R_u, G_u, B_u. We also have another player P_{check} whose purpose is to ensure no two adjacent nodes are colored the same. We attach all stumps to a node owned by P_{check} such that P_{check} can choose among the colors chosen by P_{color}. If P_{color} are able to assign colors such that no adjacent nodes share a color, then P_{color} maximizes their utility, however, if no such coloring exists then P_{check} can force a bad outcome for P_{color}. It follows that P_{color} can obtain good utility if and only there is a valid coloring (Fig. 4).

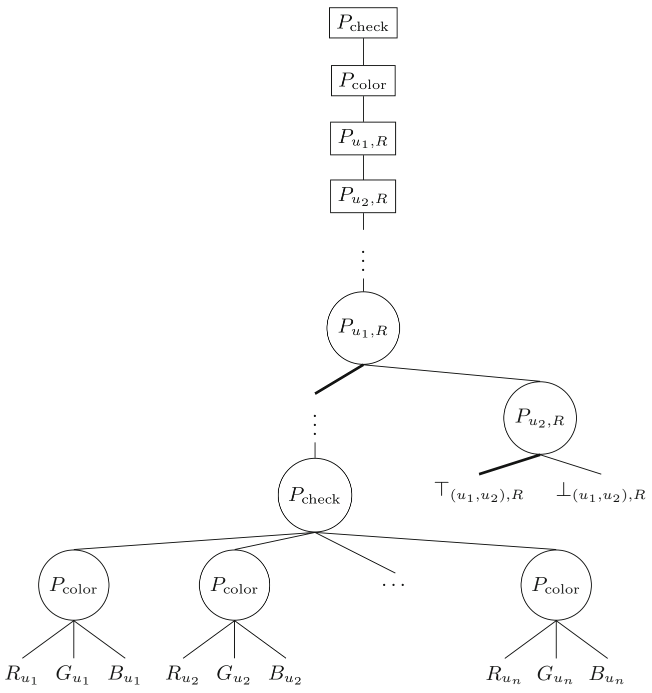

Fig. 4. The structure of the reduction. First, P_{color} is allowed to assign a coloring of all vertices. If there is no 3-COLORING of the graph, there must be some vertex (u_1, u_2) where both vertices are colored the same color c. In this case, P_{check} can force both c_{u_1}, c_{u_2}, which are undesirable to $P_{u_1,c}$, resp. $P_{u_2,c}$: then in every $P_{u_1,c}$-contract where they do not commit to choosing $P_{u_2,c}$, P_{check} cuts as to ensure c_{u_1} and analogously for P_2. It follows that P_{check} can get $\bot$ if and only if the graph is not 3-colored. Then P_{color} can get a different outcome from $\bot$ if and only if they can 3-color the graph.

Reduction. We add six contract players for every edge in the graph. Specifically, for every edge $(u, v) \in E$ and every color $c \in \{R, G, B\}$, we introduce two new contract players $P_{u,c}$ and $P_{v,c}$ who prefer any outcome except c_u (resp. c_v) being colored c. That is, if $c = R$, then the leaf R_u has a poor utility for $P_{u,R}$. We add moves for $P_{u,c}$ and $P_{v,c}$ at the top of the tree, such that if they cooperate, they can get a special utility vector $\bot_{u,v}$ which has a poor utility for P_{color} and great utility for P_{check}, though they themselves prefer any outcome in the tree (except c_u, resp. c_v) to $\bot_{u,v}$. We ensure that P_{check} has a contract directly below P_{color} in the tree. If no coloring exists, then P_{check} can force a bad outcome for both $P_{u,c}, P_{v,c}$ in all contracts where they do not commit to choosing $\bot_{u,v}$. Specifically, P_{check} first threatens $P_{u,c}$ with the outcome c_u, and subsequently threatens $P_{v,c}$ with c_v. Though they prefer any other node in the tree to $\bot_{u,v}$, they still prefer $\bot_{u,v}$ to c_u, c_v, meaning they will comply with the threat. This means P_{color} will receive a poor outcome if the coloring is inconsistent. It follows that P_{color} will only receive a good payoff if they are able to 3-color the graph, see e.g. Sect. 3.4 for an illustration.

Theorem 4. *Computing an SPE in smart contract games of perfect information is* PSPACE*-hard when we allow for an unbounded number of contract players.*

Proof. Let (V, E) be an instance of 3-COLORING. Our above reduction works immediately for $k = 1$, showing NP-hardness. To show PSPACE-hardness we reduce from a variant of 3-COLORING as described in [3] where players alternately color an edge and use a similar trick as Theorem 2 by introducing new players between P_{color} and P_{check}. □

It remains unclear where the exact cutoff point is, though we conjecture it to be for three contracts: clearly, the decision problem for three-contract games of perfect information is contained in NP as the witness (a cut for the first contract player) can be verified by Algorithm 1.

Conjecture 1. Computing an SPE for three-contract games is NP-complete. ◇

4 Conclusion

In this paper, we proposed a game-theoretic model for games in which players have shared access to a blockchain that allows them to deploy smart contracts. We showed that our model generalizes known notions of equilibria, with a single contract being equivalent to a Stackelberg equilibrium and two contracts equivalent to a reverse Stackelberg equilibrium. We proved a number of bounds on the complexity of computing an SPE in these games with smart contracts, showing, in general, it is infeasible to compute the optimal contract.

References

1. Averboukh, Y.: Inverse stackelberg solutions for games with many followers. Mathematics **6** (2014). https://doi.org/10.3390/math6090151
2. Basar, T., Selbuz, H.: Closed-loop stackelberg strategies with applications in the optimal control of multilevel systems. IEEE Trans. Autom. Control **AC-24**, 166–179 (1979)
3. Bodlaender, H.L.: On the complexity of some coloring games. In: Möhring, R.H. (ed.) WG 1990. LNCS, vol. 484, pp. 30–40. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53832-1_29
4. Bošanský, B., Brânzei, S., Hansen, K.A., Lund, T.B., Miltersen, P.B.: Computation of stackelberg equilibria of finite sequential games. ACM Trans. Econ. Comput. **5**(4) (2017). https://doi.org/10.1145/3133242
5. Breton, M., Alj, A., Haurie, A.: Sequential stackelberg equilibria in two-person games. J. Optim. Theory Appl. **59**(1), 71–97 (1988). https://doi.org/10.1007/BF00939867
6. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 261–272 (2006). https://doi.org/10.1109/FOCS.2006.69

7. Daskalakis, C., Goldberg, P., Papadimitriou, C.: The complexity of computing a Nash equilibrium. SIAM J. Comput. **39**, 195–259 (2009). https://doi.org/10.1137/070699652
8. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1990)
9. Ho, Y., Olsder, G.: Aspects of the stackelberg problem – incentive, bluff, and hierarchy1. IFAC Proc. Vol. **14**(2), 1359–1363 (1981). 8th IFAC World Congress on Control Science and Technology for the Progress of Society, Kyoto, Japan, 24–28 August 1981
10. Jeroslow, R.G.: The polynomial hierarchy and a simple model for competitive analysis. Math. Program. **32**(2), 146–164 (1985). https://doi.org/10.1007/BF01586088
11. Leitmann, G.: On generalized stackelberg strategies. J. Optim. Theory Appl. **26**(4), 637–643 (1978). https://doi.org/10.1007/BF00933155
12. Letchford, J.: Computational aspects of stackelberg games. Ph.D. thesis, Duke University, Durham, NC, USA (2013)
13. Letchford, J., Conitzer, V.: Computing optimal strategies to commit to in extensive-form games. In: Proceedings of the 11th ACM Conference on Electronic Commerce, EC 2010, pp. 83–92. Association for Computing Machinery, New York (2010). https://doi.org/10.1145/1807342.1807354
14. Liu, B.: Stackelberg-Nash equilibrium for multilevel programming with multiple followers using genetic algorithms. Comput. Math. Appl. **36**(7), 79–89 (1998)
15. Luh, P.B., Chang, S.C., Chang, T.S.: Brief paper: solutions and properties of multi-stage stackelberg games. Automatica 251–256 (1984)
16. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. The MIT Press (1994). Electronic edition
17. Sherali, H.D.: A multiple leader stackelberg model and analysis. Oper. Res. **32**(2), 390–404 (1984)
18. von Stackelberg, H.: Marktform und Gleichgewicht. Verlag von Julius Springer (1934)
19. Stankova, K.: On Stackelberg and Inverse Stackelberg Games & Their Applications in the Optimal Toll Design Problem, the Energy Market Liberalization Problem, and in the Theory of Incentives. Post-Print hal-00391650, HAL, February 2009
20. Szymanik, J.: Backward induction is PTIME-complete. In: Grossi, D., Roy, O., Huang, H. (eds.) LORI 2013. LNCS, vol. 8196, pp. 352–356. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40948-6_32
21. Tolwinski, B.: Closed-loop stackelberg solution to a multistage linear-quadratic game. J. Optim. Theory Appl. **34**, 484–501 (1981)
22. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151**, 1–32 (2014)