SoliNomic: A Self-modifying Smart Contract Game Exploring Reflexivity in Law

Joshua Ellul and Gordon J. Pace

Abstract Smart Contracts built on Blockchain systems have brought about the ability to deploy code that is tamperproof, verifiable and guaranteed to do exactly what it is written to do over a network of decentralised systems. This enables for decentralised (and centralised) organisations to define rules for particular operations, or even for the entire organisations' decision-making processes. The rules, decision making processes and ability to alter both the rules and processes are encoded within smart contracts. Nomic was a game of self-modifying rules originally meant to explore self-reference and amendment from a regulatory perspective. In this paper we explore the encoding of such a rule system formally as self-modifying yet otherwise tamper-proof code in smart contracts.

Keywords Blockchain · Smart contracts · Nomic · Normative reasoning · Blockchain game

1 Introduction

The notion of self-reference has long been recognised as a source of paradox, and can be traced back at least to Epimenides' paradox[1] if not even earlier with various religious concepts being rife with self-reference, particularly in creation myths. Philosophers, logicians, and later on, mathematicians have formally explored and studied such self-referential systems recognising that the complexity induced by such a notion is best avoided in a formal system if at all possible. However, many

[1]Epimenides claimed that "All Cretans are liars," despite being himself a Cretan.

J. Ellul (✉) · G. J. Pace
Centre for DLT, University of Malta, Msida, Malta
e-mail: joshua.ellul@um.edu.mt

G. J. Pace
e-mail: gordon.pace@um.edu.mt

Department of Computer Science, University of Malta, Msida, Malta

A. Dingli et al. (eds.), *Disruptive Technologies in Media, Arts and Design*, Lecture Notes in Networks and Systems 382, https://doi.org/10.1007/978-3-030-93780-5_1

domains do not have the luxury of choosing whether or not to allow self-referential statements. It suffices to note that two of the greatest mathematical results of the twentieth century, those of Gödel's incompleteness theorem [3] and Turing's results on computational limitations [12] arise directly due to the inescapability of self-reference in mathematics and computation.

One other domain in which self-reference is unavoidable is that of legislation and regulation. Literature on the philosophy of law and normative logics is rife with instances of paradoxes arising from such self-reference. In particular, regulatory documents which regulate the way they may be modified cannot escape self reference. Not all rule systems require this feature—it suffices to look at the rules of the vast majority of games which regulate how players interact but do not provide means through which the rules can be changed as part of the game itself. A notable exception to this rule is that of Nomic [4], a game developed by Peter Dain Suber in 1982 precisely with the intent of exploring self-amendment, thus embracing rule modification at the very core of the game. One may argue that since then, a number of games have explored player interaction with the rules, from the ubiquitous 'The text on the cards trumps the rules' from many a card game to legacy games which allow players to modify the rules as the game progresses, even if this is largely limited to having first class rules which are immutable, and second class rules with which players may interact. Where rule self-amendment starts and where it stops is nebulous. Consider a re-ruling of snakes and ladders in which (i) players are required to move forward one step during their turn; and (ii) players roll a six-sided die at the beginning of their turn, thereby amending the previous rule to say "players are required to move forward n steps during their turn", where n is the number shown on the die. Clearly, the ruleset is not universally amendable (the second rule cannot be changed), but it clearly has an element of self-amendment (at the ruleset level) albeit lacking player agency.

Perhaps Nomic can be seen as the maximal element in the ordered set of games with self-amending rules, but even this is unclear. Nomic can also be seen to be the all-encompassing game, containing within it the possibility to play any other game—perhaps other games can be seen to be 'bonus games' that are possible within the main game. Nomic allows for rules to be fixed in stone: it suffices that the players agree to enact a zeroth rule saying 'Notwithstanding what any other rule of this game may say, this rule cannot be modified or removed'. Legal discussions aside as to the enforceability of such a rule, perhaps an even freer game is a variant of Nomic in which no rule may be set in stone. Then again, this maxim is itself an immutable (meta?) rule of this variant of Nomic contradicting itself. It is clear that deciding the hierarchy of rule amendability in games is far from easy to define and reason about.

Another element which is largely overlooked in the discussion of Nomic (and other self-amending rulesets) is that such a ruleset resides in an ecosystem of social, economic and technological norms. Players of a game frequently reside to social norms, for instance when they realise that they have misinterpreted a rule and agree to proceed with the game abiding by the misinterpreted rule or when players decide to adopt house-rules which they may deem to be more interesting or fun. Ambiguity in rules typically require players to agree on an interpretation—and yet nothing in the rules of Nomic states how such an agreement must be reached[2] and the social contract between players has to kick in. What stops players from agreeing to ignore a Nomic rule 'for the time being'? Such regulatory frameworks sitting above normative systems similarly exists with legal systems—no revolution ever considered whether the changes it is striving for abide by the current legal system.

In this paper we explore an implementation of Nomic as a set of smart contracts written in Solidity (hence the name). Smart contracts [11, 13], residing on a blockchain [8], or similar distributed ledger technology (DLT) have been hailed as the computational embodiment of legal contracts. The implementation allows for an unambiguous operationalisation of the game rules and the process of self-modification. However, its value lies even more as a vehicle for the exploration of issues and underlying assumptions inherent in self-amending systems.

The paper is organised as follows, in the next section we'll briefly describe what blockchain and smart contracts are followed by a discussion highlighting aspects that can be seen as games. In Sect. 3, we will provide implementation details of SoliNomic, and will follow with a discussion in Sect. 4. Finally, Sect. 5 will provide concluding thoughts.

2 Blockchain, Smart Contracts, Rules, and Games

Since Bitcoin was proposed in 2008 [8], blockchain and related DLTs have been proclaimed as game-changing (no pun intended) technology—allowing for centralised points of trust to be removed. This is achieved by replicating data storage and repeating verification and auditing processes across all nodes in the decentralised network (at least in the traditional platforms). To overcome the double-spend problem (which allows for someone to spend their same last unit of currency at two different points in the network at the same time), proof-of-work was proposed—which provides for a means to: (i) select a node in a random-like manner to be the next node to propose a block of transactions to add to the canonical ledger (so that it would be impossible for hackers to determine which node to attack at any point in time); (ii) allow for the time when blocks are to be added to the ledger to be regulated; (iii) allow for nodes to easily check that the block is valid; and (iv) eventually agree on which blocks to accept as truth in the case of different blocks being added at the same time. This is what a blockchain achieves, a means of creating a network of untrusted computers which can maintain a ledger (or append only database) without a single trusted node—unlike distributed systems that had been proposed prior.

Bitcoin provided for the implementation of a ledger tracking cryptocurrency transactions and ownership, yet Blockchain and DLTs can be used for more than that of cryptocurrencies. Ethereum [13] proposed exactly this—a general purpose blockchain that would allow for different applications, typically written in Solidity,

[2] Not even adopting a rule regulating how to resolve ambiguity solves the problem if an ambiguity is found in this very same rule.

to execute on top of it. These applications that execute on blockchains are often termed smart contracts. Mainly because one of their main features are that the code is transparent and available for anyone to view and no one can manipulate code that has already been deployed—so whatever is written in the code is guaranteed to be executed and therefore can be seen as the execution of promises encoded within. Later in Sect. 3 we'll demonstrate snippets of our implementations in Solidity—ample material can be found online to get started with Solidity and was left due to space restrictions.

At the blockchain level, the different nodes can be seen to be working together to maintain the ledger and ensure that only new valid blocks of transactions can be added, and old valid blocks can neither be removed nor edited—some might say following strict rules. Really though, not all nodes are working together in a benevolent manner and neither are nodes necessarily following strict rules as per the specified blockchain protocol. Actually, a blockchain could be viewed as a game, made up of: (i) benevolent players that are correctly following the protocol; (ii) those that may have bugs in their code which could work against the aim of 'correctly' upkeeping the ledger; and (iii) malicious players aiming to invalidate or alter historical transactions. If either (ii) and/or (iii) acquire a coordinated majority[3] of computational power in the network then it would be possible for an attack to be undertaken in which case the benevolent players would have lost in ensuring the ledger's validity. On smaller sized networks it is more feasible to implement such an attack. On a large network like Bitcoin it becomes infeasible.

Whilst, at the blockchain level, rules to maintain the ledger cannot be enforced, by ensuring enough benevolent players (with enough computational power) are in the game, then the ledger's validity will be sustained. With systems like Ethereum, that allow for different application logic to execute on top of the blockchain (instead of just tracking cryptocurrency transactions), any rules can be encoded. At the abstract level of smart contracts, a valid ledger is assumed (and should be taken care of by the blockchain). As long as the blockchain keeps doing what it should be doing (maintaining the valid ledger), then the smart contracts will continue to execute the code that is encoded inside of them. So, the hardcoded rules encoded within smart contracts cannot change, unless the smart contracts originally factored in rule changes—in which case the original rules would still not have changed, since the original rules include rule changes. Within smart contracts, any game could be encoded (subject to expressibility and/or computationability limitations), allowing for different types of players and rules. The rest of this paper will focus on the implementation of SoliNomic, nomic coded in Solidity allowing for different games to be encoded and evolve within the Smart Contract level.

[3] Though Saad et al. [10] cites that less than 51% may be required.

3 SoliNomic

The rules of a game with a static ruleset can be operationalised to provide functionality regulated via smart contracts. The rules would thus be enforced by the logic written on the smart contract, ensuring that players cannot cheat by unilaterally changing the state of the game or playing outside the rules. The immutability of smart contracts ensures that the rules are there to stay, and cannot be tweaked halfway through a game.

3.1 Representing the SoliNomic Ruleset

Representing Nomic in terms of a smart contract thus leaves one in a quandary since at the very heart of Nomic is the axiom that the ruleset will change. One possibility is to represent the ruleset as data (e.g. an expression in a domain specific language rich enough to express any potential behaviour described in the ruleset as it evolves) in which it can (i) be interpreted by the SoliNomic smart contract to execute the logic of the rules, but also (ii) modified by the logic itself as all data stored in a smart contract can. This detaches the isomorphism between the ruleset and the smart contract, since the correspondence would now be between the ruleset and the state of the smart contract, with the smart contract logic acting just as an interpreter of the data. Our intention was to preserve the link between the ruleset and the smart contract, with the latter being a direct operationalised version of the former. Since, however, one cannot update a smart contract's code (bytecode) after it is deployed on the blockchain, we use a standard design pattern for updateable logic—that of allowing the control of a smart contract to migrate to a new one which may have different logic. If such changes are only accepted if invoked by the current ruleset smart contract itself, we are guaranteed that any changes would be according to the current rules of SoliNomic and cannot be changed outside their restrictions.

3.2 The Game Versus the Ruleset

The identity of a non-reflexive game can be associated with its ruleset. The game of chess corresponds to the rules of chess. But this correspondence is lost on reflexive rulesets. If we were to use such a correspondence, a game of Nomic would morph into a new game (since the ruleset has changed) the moment a move is made. But the players would still see themselves playing as the same game instance. We have decided to make this distinction between a game instance and the current ruleset explicit in the implementation of SoliNomic. An overview of the architecture is depicted in Fig. 1. At its heart, a SoliNomic game instance is a smart contract that keeps a reference to the current ruleset, and provides functionality to allow for the ruleset to be updated, as shown below:

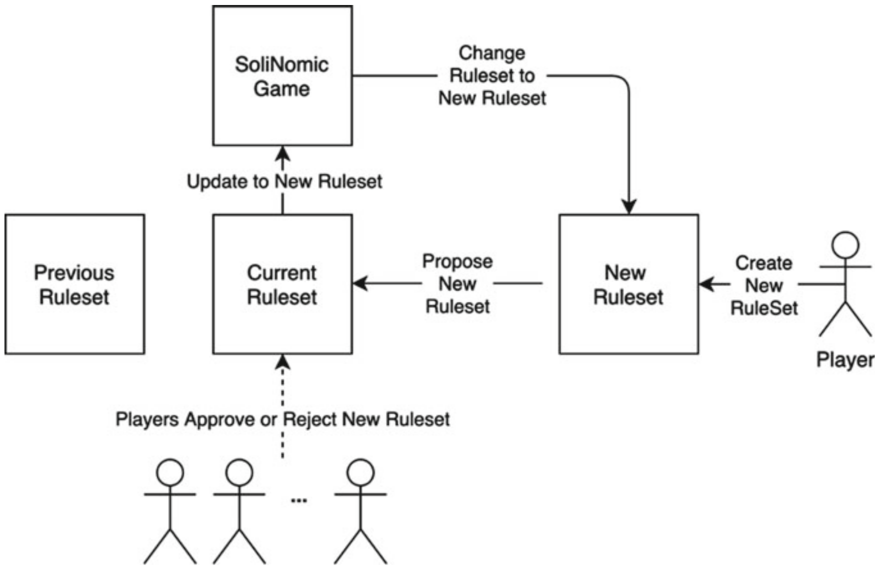

Fig. 1 SoliNomic overview

```
contract SoliNomicGame {
    // Address of the ruleset
  address ruleset public;

  function updateRuleset(address _ruleset) public {
      require (msg.sender == ruleset);
        ruleset = _ruleset;
  }
}
```

Note that the game smart contract does not embody any game logic, which is reserved to the ruleset smart contract. The only logic in the game is that of allowing updates to the ruleset emanating from the current ruleset smart contract itself. The ruleset smart contract would, in turn, embody the operationalisation of the logic, and route any changes to the ruleset to the game smart contract as follows.

```
contract SoliNomicRuleset0 {
  // Address to the game to which this ruleset belongs
  address game public;
  ...

  // Players can propose a rule change (represented by
      new ruleset smart contract)
  function propose(address _proposedRuleset) public {
    ...
  }

  ...

  // Adopt a proposed change if the majority agree
  function approveAmendment() public {
    // Ensure that the majority have voted in favour
        of the proposed
    require (2*votesInFavour > playerCount);
    // Abdicate in favour of the new ruleset
    game.updateRuleset(proposedRuleset);
    selfdestruct();
  }
}
```

This architecture already raises various implementation options, which run parallel to interpretations in the real world Nomic. Consider the list of players in a game. Given that the ruleset can be changed to enroll new players or drop old ones, should the list of players be stored in the state of the ruleset, or is it an inherent part of the game, and should thus reside in the game smart contract with logic to allow the ruleset to update the list of players? From an implementation perspective, both are possible, with the former resulting in simpler logic (since there is no need of player manipulation to be encoded at the game smart contract level) whilst the latter can be argued to be closer to what the real world game. However, since the game rules proposed for Nomic simply refer to ‘the players’ with no rules to cater for adding or removing players, neither option is the absolutely faithful one. In SoliNomic, one can experiment with either option to investigate the difference in feel of the two options.

3.3 *Rules Versus the Ruleset*

Amendments in Nomic are meant to be a single rule, even if this arises only indirectly from 111, which states that ‘if [a rule-change] arguably consists of two or more rule-changes compounded […] then the other players may suggest amendments or argue against the proposal before the vote’. Although (arguably) still giving leeway to multiple amendments (if the other players choose not to amend or argue against it), the intention is clearly that single rule changes. In contrast, the code shown above allows for ‘global’ amendments—proposed amendments to the rulebook as a whole. We have also explored the option of factoring the ruleset into individual functional elements which can only be amended, added or removed one at a time.

Each rule clause can be encoded as a simple smart contract with a standard means of invoking it:

```
contract Clause17 {
  ...
  function execute(...) public {
    ...
  }
}
```

The individual rules are then collected together in a ruleset contract with a mapping from rule name[4] to the rule contract and allows for invoking of individual clauses, and for amendment of individual rules from the game contract:

```
contract SoliNomicRuleset0 {
  ...
  // The rules making up the  ruleset
    mapping (bytes32 => address) rules public;
  function invoke(bytes32 _rulename) public {
      rules[_rulename].execute(...);
  }

    function amend(bytes32 _rulename, address
        _rulelogic) public {
    require (msg.sender == game);
    rules[_rulename] = _rulelogic;
  }
}
```

The game contract remains almost unchanged from the one we saw earlier, except that amendments are now proposed on a particular named clause:

```
contract SoliNomicGame {
  // Address of the ruleset
  address ruleset public;

  function updateRule(bytes32 _rulename, address
      _rulelogic) public {
    require (msg.sender == ruleset);
    ruleset.amend(_rulename, _rulelogic);
  }
  ...
}
```

In this manner, we can achieve rule-centric amendments being proposed by players. Having shown how such a rule-level amendment process can be adopted and implemented, it is worth highlighting that this clause-level amendment could have easily been implemented as an amendment to the ruleset smart contract using the previous approach, with proposals being made and voted upon at the clause level i.e. leaving the rest of the ruleset unchanged. The flexibility of SoliNomic (and Nomic) ensures that the two approaches can result from one another in the course of a single game.

[4] For efficiency reasons, we use the hash of the name of the clause, but for all intents and purposes this can be seen as a mapping from clause name to clause functionality.

4 Discussion

Regulating interaction in a decentralized manner: Nomic does not adopt a centralised authority regulating the players with respect to the game rules, but instead depends on the players to regulate their own behaviour. SoliNomic goes one step further in that it provides an automated way in which the rules are enforced, thus ensuring that the players do not diverge from them. However, the mechanism inherent in the smart contracts running on a DLT ensure that there is no central authority controlling them. Although one finds other automated Nomic rule enforcement engines in literature e.g. see [1, 9] and Nomyx[5] these were all implemented on a central server, meaning that the administrator of the server can disrupt or modify the rules.

Games as decentralised (autonomous) organisations: The issue of decentralisation highlighted an aspect of many games which provide regulated interaction between players without giving the power of enforcement to a subset of the participants. The rules are adhered to in a decentralized manner, using social norms to discourage players from cheating, and decentralised enforcement with players observing each other to ensure that everyone is playing by the rules. By adding a decentralised engine to enforce conformance with the rules, as we have done in SoliNomic, we have essentially created a decentralised autonomous organisation (DAO) [2], in which the behaviour is encoded by unambiguous executable rules, yet providing a degree of control by the organization members without being influenced by a central authority.

Tyranny of the majority: Some may argue that the automated enforcement provided by SoliNomic ensures that the game cannot diverge from the rules themselves, along the lines of Lessig's code is law maxim [6]. For instance, one cannot revert back time and undo changes in an arbitrary manner. However, nothing stops the majority of the players from deciding to stop using the original SoliNomic game smart contract and create a new one redistributing points or taking off from a point in the past. This is analogous to how the Ethereum community (controversially) chose to undo a successful hack of the original DAO fund raising smart contract by collectively choosing to resume the Ethereum blockchain from a few blocks back [7]. The existence of a social layer above the automated system leaves space for such divergences, and it can only be the social contract between the members of the organisation that discourages them taking such action.

Pushing the boundary of what is mutable and what is immutable: As we have seen in the previous section, the implementation of SoliNomic brought out more clearly underlying assumptions about mutability and immutability of rules. The choice of having any logic starting in the game smart contract which is immutable ensures that the rules embodied in that logic cannot be modified. However, this can be pushed further by adopting a hierarchy of smart contracts, allowing more mutability the lower down one moves in the hierarchy. We have seen this with named clauses, which allow controlled mutability of the ruleset, and this notion is already present

[5] https://github.com/nomyx/Nomyx.

in the original rules of the game of Nomic which consist of mutable and immutable rules (Nomic, not blockchain terms) and a process of transmutation to allow for rules to migrate from one form to another.

Mutability of state versus mutability of rules: One interesting feature of games which is highlighted by the implementation of SoliNomic is that games offer a degree of mutability of game state (e.g. the position of the pieces in chess), which is distinct from mutability of rules which only a few games permit. However, the former can be modelled as a limited form of the latter, e.g. adapting the chess rules to include information as to where the pieces lie and whose turn it is, and then interpreting the moving of a piece as an amendment to the rules. The dividing wall between rules and game state has been explored in a number of game genres, but the degree to which it is explicit in SoliNomic enables exploration of this space more clearly.

Bugs and features: SoliNomic takes the approach of code is law—what the code does is what the rules say. Note that this is distinct from saying that what the code is intended to do is what the rules say. The notion of bugs in the code does not exist, since the code is by this definition canonical. Features of the game arising from bugs typically cut the game short, allowing the perceptive player to use the behaviour to their advantage. However, occasionally it may lead to interesting unplanned scenarios which diverge from the players' original intent.

Dealing with ambiguity: The fact that Nomic rules are written in a natural language means that this may result in ambiguity which the players would have to resolve, so much so that the original rules explicitly refer to such a possibility. In contrast, an automated implementation means that such ambiguities would have to be resolved when implementing as an operationalised ruleset. We have already discussed player management. The rules simply refer to the players without providing any mechanism to manage who is playing. An inherent assumption may be that the players remain the same throughout the game, but some players may allow for players to join or leave the game as it evolves. Others may choose a midway interpretation in that the players are fixed but one may adopt new rules to allow for onboarding and jettisoning of players. Another source of ambiguity in the English ruleset lies in the proposal and voting mechanism. While the rules are specific in saying that 'Turns may not be skipped or passed', it is unclear how a group of players are to proceed if a player refuses to make a proposal when it is their turn. Similarly, it is underspecified whether all players have to vote, whether the vote is secret or not, whether players vote simultaneously or in sequential order, etc. Operationalising the rules means that one has to make an explicit decision for each of these choices. If different options are to be kept, then there still has to be the explicit choice handled by the logic (e.g. the current player takes the decision of which interpretation to adopt).

Declarative versus operational descriptions: Although the vast majority of smart contract languages are operational i.e. they allow one to express how to achieve a goal, one can find a number of languages which attempt to take a more declarative approach i.e. allowing one to express what is to be achieved as opposed to how to achieve it. Legal texts and game rulebooks are typically declarative, and this leads to a gap between the implementation of SoliNomic and the original ruleset. The only implementation of a declarative approach to Nomic we are aware of is Bananomic [1] which, however, limits the rules to talk about a concrete scenario (rules regulating monkeys which are throwing bananas at each other in a tree) in order to ensure that the declarative rules can be interpreted and executed.

Other attempts at automating Nomic: In Nomic, as is the case with most games, rules are expressed in a declarative, not operational manner. The discussion as to the relationship between legal and smart contracts has long been discussed [5]. Normative rules specify the ideal course of affairs (‘One ought to submit their tax return by the end of June of the following year’), but recognise that such ideal behaviour is not guaranteed (‘A fine of €10 per day shall be incurred for late submissions of one’s tax return’). This is distinct from smart contracts which typically operationalise behaviour to ensure compliance with the rules. It is worth noting that in the case of game rules, despite them being expressed as declarative normative rules, only compliant behaviour can typically be observed: if the rules say that ‘A player may play no more than three cards per turn’, players do not have the option to violate the rule (in the same way that a person may choose not to submit their tax return by the due date).

Our approach in SoliNomic follows that of various automated versions of Nomic such as PerlNomic [9] take an operational view, equating proposals with code to be executed. In contrast, BanaNomic [1] took a declarative approach, but suffers from a need to understand various actions (e.g. ‘play card’, ‘propose amendment’) and states (e.g. ‘no player has more than 10 points’, ‘no amendment has yet been accepted’) and their relationship. The operational approach provides a shortcut in that the code refers to and modifies the state of the game directly.

Without the use of smart contracts, previous implementations suffered from the problem of centralised governance of hardware. The owner of the server may choose to change data without going through the rules of the game. Consider PerlNomic, in which the rules took the form of Perl scripts accessible from a server. The scripts provided limited means through which the players were allowed to change the scripts themselves. However, the person owning the server could easily edit these scripts directly without following these rules. Smart contracts ensure that this is not possible.

5 Conclusions

In this paper we presented a decentralised version of nomic, SoliNomic, implemented in Solidity. To the best of our knowledge this is the first decentralised version—which does not require players to regulate or keep in check the actions of others, and neither allows for any centralized system operator or hardware maintainer to manipulate the state and/or game rules. Further work should be undertaken to investigate other design options in regards to: (i) which logic and/or rules should be well-defined in the smart contract code and which should remain as unstructured logic left up to the players to define; and (ii) different design patterns for updating rulesets, rules and players.

References

1. Camilleri, J.J., Pace, G.J., Rosner, M.: Controlled natural language in a game for legal assistance. In: International Workshop on Controlled Natural Language, pp. 137–153. Springer (2010)
2. Chohan, U.W.: The decentralized autonomous organization and governance issues. Available at SSRN 3082055 (2017)
3. Gödel, K.: On Formally Undecidable Propositions of Principia Mathematica and Related Systems. Courier Corporation (1992)
4. Hofstadter, D.R.: Nomic: a self-modifying game based on reflexivity in law. In: Metamagical Themas: Questing for the Essence of Mind and Pattern, pp. 70–86. Bantam, New York (1985)
5. Ladleif, J., Weske, M.: A unifying model of legal smart contracts. In: International Conference on Conceptual Modeling, pp. 323–337. Springer (2019)
6. Lessig, L.: Code: and other laws of cyberspace. ReadHowYouWant.com (2009)
7. Mehar, M.I., Shier, C.L., Giambattista, A., Gong, E., Fletcher, G., Sanayhie, R., Kim, H.M., Laskowski, M.: Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. J. Cases Inf. Technol. (JCIT) **21**(1), 19–32 (2019)
8. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
9. Phair, M.E., Bliss, A.: PerlNomic: rule making and enforcement in digital shared spaces. Online Deliberation (2005)
10. Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., Mohaisen, A.: Exploring the attack surface of blockchain: a systematic overview. arXiv preprint arXiv:1904.03487 (2019)
11. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY J. Transhuman. Thought (16) **18**(2) (1996)
12. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. Lond. Math. Soc. **2**(1), 230–265 (1937)
13. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**(2014), 1–32 (2014)