VALIDATION and VERIFICATION of SMART CONTRACTS: A RESEARCH AGENDA

AUTHORS: Daniele Magazzeni, Peter McBurney and William Nash.

ABSTRACT:

In this paper, we explore some of the issues and research challenges involved in the validation and verification of smart contracts, particularly those running over blockchains and distributed ledgers. Smart contracts may encode legal contracts written in natural language, and these in turn may be intended to represent in written form the shared understandings and shared intentions of the contracting parties. Thus, automation of the validation and verification of smart contracts will require formal (machine-readable) representations of the program code of the smart contract and of the written natural language contract. Because distributed ledgers are a form of multi-agent (machine-to-machine) communication, where participating nodes communicate via pre-specified communications protocols, formal representation of the program code is able to draw on research exploring the syntax, semantics and pragmatics of agent communications protocols.

KEYWORDS: Blockchains, Distributed Ledgers, Smart Contracts, Verification.

1. Introduction

The world of commercial computing and information technology has been recently become very excited by a new technology, called variously blockchains, distributed ledgers, and/or shared ledgers. The underlying blockchain technology was invented in 2008 to support the creation and exchange of a cryptocurrency, Bitcoin, without need for a controlling central authority [1]. The Blockchain platform and associated cryptocurrency were implemented and launched in 2009. In the years since, many people have seen potential for applications for this technology, particularly in finance and government. For instance, in the financial sector, more than 70 global financial institutions have partnered in the R3 consortium, which has recently released a conceptual framework, *Corda*, for smart contracts in finance [2]. To any participant old enough, the level of excitement is only matched by that for the emerging World-Wide-Web in the middle 1990s.

A distributed ledger is a shared database, fully replicated at multiple autonomous sites or nodes, and without any node having central or privileged control. Updates to the database require execution of an agreed multi-node decision-protocol; normally, data may only be appended to but not erased from the shared database. A blockchain is a special type of distributed ledger,

where the data is collated into “blocks” before being added to the shared database, and the blocks are linked together to form a single sequential chain. Cryptographic methods may be used to encrypt data, to authorize data for upload, and to chain blocks together. The Bitcoin blockchain was invented for issuing, executing, and recording transactions of a cryptocurrency, Bitcoin. A distributed ledger may also use or support an electronic currency, even if the database is intended for other purposes. Participation in such a network may be open to anybody, as is the Bitcoin blockchain, or a node may require pre-authorization and perhaps identification to join. Distributed ledgers requiring pre-authorization are called “permissioned ledgers” while open networks are called “permission-less”. Permissioned ledgers will typically enable participants to identify one another, unlike the pseudonymity of many permission-less ledgers. Different applications will require different system architectures and designs.

What is the reason for the excitement about this technology? We can answer this by comparing the technology to the Web. The Web made it relatively easy to share information between one person or organization and others, either by placing the information directly onto a web-page, or by enabling access via a webpage to the information (eg, when stored in a database). One way to view a web-page, therefore, is as akin to a blackboard, where one person (controller of the server) has the power to write contents onto the board, and to erase contents from it. If the web-page is public, then anyone else may read the current contents of the board. If the web-page requires access permission (as, for example, subscription newspaper sites do), then the ability to read the contents may be limited to a private group.

Under this metaphor, Distributed Ledger Technologies (DLT) provide the readers of the blackboard with some extra powers, at the possible expense of the powers of a writer. Distributed Ledgers only allow content to be written to the blackboard if the readers first agree, according to some pre-specified voting rules. For example, readers may all have to agree for content to be added, or a majority may have to agree, or a particular nominated reader may have to agree. Contents are added to the board in a manner which links them cryptographically to the past contents of the board, which is the “chain” of blockchain. In addition, the contents of the board are impossible to erase. Indeed, “erasure” may only be possible by starting a new board which is the same as the original blackboard up to the point just prior to that where the contents are to be erased. Because this action creates two parallel chains which are identical to the point where they diverge, the process is called “*forking*” the blockchain.

These additional rights for readers create some interesting consequences, which explain much of the excitement. The first consequence is that anyone writing onto the board knows that the other participants (the readers) have seen the content. They must have seen it to vote on it before it was uploaded to the board. Everyone party to the distributed ledger knows the same content (ie, the participants have “*shared state*” for the variables on the ledger) and everyone

knows that everyone knows this, and so on. In the language of game theory [3], the participants have "*Common Knowledge*". This makes it impossible for a participant to plausibly repudiate the content of a shared ledger at a subsequent time. Second, the chaining of content means that any change to past contents requires changes to all the later contents. Such changes could not happen surreptitiously, and, like all proposed writing to the board, would themselves require prior approval from the participants. Hence, the past records are effectively immutable. Third, the shared ledger is not stored on only one machine, but copies are stored locally, on each of the machines of the participants. Every reader has their copy of the board, and these copies stay in synch with each other. This creates significant challenges for anyone wishing to corrupt or alter the contents, as every participant machine would need to be hacked.

These properties are the origin of the many statements that DLT eliminate the need for trust. Two or more parties engaging in a transaction online need to be sure that commitments will be honored and that individual actions of multi-step transactions will not be reversed. Traditionally parties who did not know one another would use a third party, often a law firm, bank or other regulated entity, whom they each trusted to assist with this, for example, by acting as a witness to promises and by holding funds in escrow until other steps of a transaction were completed. DLT ensures that all promises and actions are witnessed by all the participants in the shared ledger, not only the parties to the transaction, and makes it difficult or impossible for any party subsequently to repudiate, corrupt or reverse the actions undertaken. Of course, the participants may not need a trusted third party to witness their individual transactions, but they will still need to trust the software being used.

2. Smart Contracts

Anything expressible in digital form may be written on an electronic blackboard. In addition to transaction data, a distributed ledger may also hold computer programs. Such programs could use data from blockchain records as inputs and then generate outputs which are in turn written to that same or to another blockchain. If stored on a blockchain and executed, their execution will take place locally at each participant node connected to the blockchain. Because many of the applications envisioned for distributed ledgers involve co-ordination between distinct people or organizations, an obvious application of such computer programs is for the automatic execution of business workflows across multiple organizations. For example, the legal movement of physical goods from one country to another typically requires expert and import permissions from the countries involved. Associated with these permissions may be taxes or duties, evidence of payment of which is required for the relevant permissions to be granted. The vendors and/or buyers of these goods may require bridging finance to cover delays in receipt of payments or goods, and they may purchase insurances of various forms along the

chain. There may be multiple parties – vendors and buyers of the goods; transporters, banks providing trade finance, insurance companies, customs departments of different countries, etc. The workflow could – at least, in theory – be managed by a sequence of computer programs that execute automatically as successive intermediate milestones in the journey of the physical goods are reached.

Such programs are examples of "*smart contracts*", terminology invented by Nick Szabo before the development of the Blockchain. According to Szabo, a smart contract is "*a set of promises, specified in digital form, including protocols within which the parties perform on these promises*" [4]. To the extent that such programs embody agreements between two or more organizations, distributed ledgers are a natural home for these programs. The advantage of putting the records of these events and the controlling programs onto a distributed ledger is that the various parties could all see these programs and monitor the progress of the trade flow as it happens. Moreover, the ledger tracks the chain of custody for the assets involved and records their provenance.

In addition, smart contracts on distributed ledgers of assets often have the capacity to do 'real time settlement' for the assets embedded or represented in the ledger. Where a blockchain is a record of who owns what amounts of a cryptocurrency, for example, smart contracts on that blockchain can effectively settle transfer of that cryptocurrency in its native form. Where assets are dematerialized and then recorded onto a blockchain, smart contracts can perform the same function. These smart contracts have a feature which they would not have had they been recorded on a medium other than the medium in which those assets 'exist'. This feature is a key reason for the disruptive nature of distributed ledger and smart contract technologies in finance.

Because smart contracts embody and execute workflows, they may potentially be used for automation of regulatory workflows, for example: reporting and monitoring of required data; checking of compliance; approval (or not) by a regulator; and even the levying and payment of fines for non-compliance. Philip Treleaven has termed such applications "*RegTech*", and they have significant potential for application of AI to governmental and regulatory activities.

As with distributed ledgers themselves, because the technology is still emerging, the nature and form of smart contracts are diverse. For instance, the design of smart contracts may differ significantly according to whether or not the distributed ledger on which they execute is permissioned (since this influences the ability to identify participants), and whether or not the ledger has an internal cryptocurrency. Murphy and Cooper have proposed a spectrum of definitions of smart contracts [5, page 13], according to the extent to which a contract embodies

computer code versus natural language. This is shown in Figure 1 (adapted from page 13 of [5]).

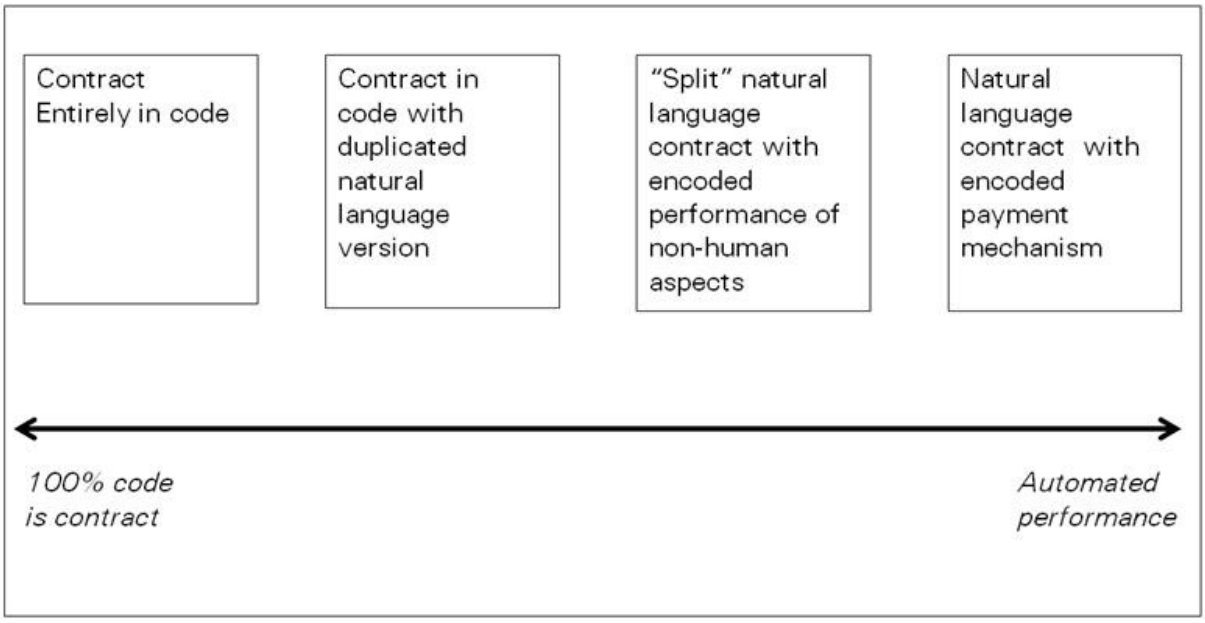

Figure 1: Smart contracts lie on a spectrum (from [5], page 13)

At the left-hand extreme position in Figure 1, any agreement between the parties is only represented in the form of computer code. At each of the other positions, there is a role for a document written in natural language. The document may duplicate the code, or it may complement it. Even at the left-hand extreme position, where there is no natural language document, the code will be intended to execute some intentions of the relevant parties based on some shared understandings; both the intentions and the understandings of the parties may be implicit.

3. Validation and Verification

From the spectrum just presented, we can identify three types of component: electronic program code; a natural language contract; and the shared understandings and intentions of the parties. Any particular smart contract may involve all three components or may not, and the role played by each component may vary, as indicated in Figure 1. For smart contract programs to execute correctly, we could thus identify several classes of questions as follows, corresponding

to the relationships between the different components. These components and the relationships are shown in Figure 2, where the numbers refer to the list of questions here:

1. Does the written natural language contract correctly and fully represent the shared understanding and shared intentions of the parties?
2. Does the computer program correctly encode the written natural language contract?
3. Does the computer program do what it is intended to do?
4. Does the computer program do only what it is intended to do? In other words, does the program not do anything it is not intended to do?
5. If there are multiple computer programs, does the operation of the system as a whole perform without error and only in desirable ways?

These questions are instructive for why we can dismiss smart contracts which lack any form of natural language component: if the only natural language intention is merely '*to execute the code',* then Questions 1-5 become trivial and all possible outcomes are validated. This would leave owners of the smart contract in a situation where they seemingly desired things they had no prior knowledge about. For instance, if there was a bug in the code (which is very plausible) then no developer or participant would know about the bug but the execution of that bug would now become 'intended' by the owners of the smart contract, without their foreknowledge. Since it is not possible to intend something unwittingly, we can see that a smart contract without at least a small natural language underpinning is not sensible.

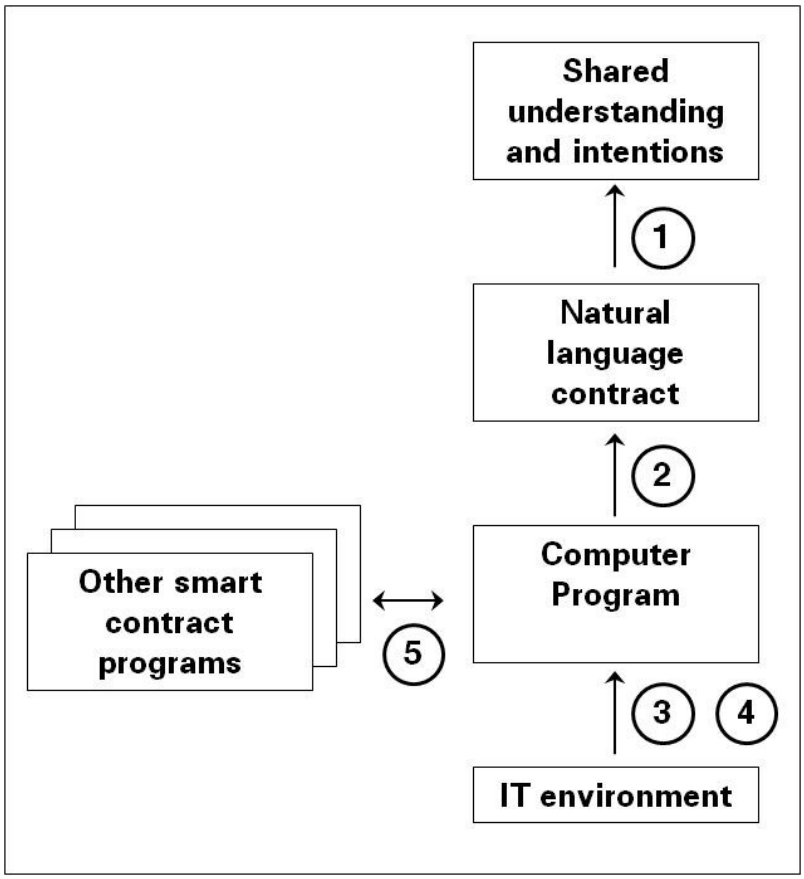

Figure 2: Components of smart contract system

In terminology adopted by the IEEE [6], Questions 1 and 2 are about *validation* – respectively, validation of a natural language contract and validation of an individual computer program. Questions 3 and 4 are about *verification* of the properties of a program. Question 5 concerns *verification* of the properties of a collection or a system of programs.

An example may illustrate these issues. The recent experience of The DAO, a decentralized autonomous organization, is probably well-known to readers [7]. This was an open-source computer program created to run over the Ethereum distributed ledger with the aim of pooling crowd-sourced funds for investment in startups. Over a month in May 2016, some 11,000 investors gave cryptocurrency then valued at more than USD 150 million to it. While the code did what it was intended to do (Question 3), it was also able to be used to do something that was not intended (Question 4). A recursive calling vulnerability was exploited by a malicious person (or team) to transfer many of the invested funds to an account he or she (or they)

controlled. Here the program code ran correctly and without errors, but in a manner unintended by the developers. Because of inbuilt delays in funds transfers, the community of participants were able to reverse this, but only by creating a fork of the Ethereum blockchain, ie, by creating another blackboard forked prior to the point of the hack event.

Each of these five validation and verification questions could be answered for any particular smart contract by means of analysis undertaken by humans and discussion between the relevant parties. The human expertise required for Questions 1 and 2 would include legal knowledge and experience, while Questions 2 to 5 would require software development expertise and perhaps other computational knowledge and skills. Using human expertise in this way is not scalable, and probably neither efficient nor very accurate. Our goal is to automate the analysis required to answer each of these questions for any proposed smart contract. Is this possible?

Formal and automated verification of computer programs has been an active area of research in computer science for at least three decades, with several methods now in use. A common method is called *model checking*, in which the task of program verification involves proving properties of a mathematical model of the program. These techniques have been widely applied in industry, for example in the design of computer hardware [8] and autonomous underwater vehicles [9].

Most applications of smart contracts being considered in industry are intended to automate or semi-automate business processes, which are combinations of sequential, parallel or interleaved actions by multiple participants undertaken over time. For this reason, action sequences and message sequence charts (MSCs) are obvious specification and design tools for the interactions and communications between participants in distributed ledgers, and deterministic finite automata (DFA) an obvious computational model of their operation. DFA present an external view of the actions of a participant or a smart contract a ledger, without modeling their internal states.

We can formally model the internal states of entities (participants or smart contracts) and how these change over time as they interact with one another, using modal logics having temporal and epistemic operators. For example, Public Announcement Logics (eg, [10]) aim to represent updates in the knowledge of hearers when speakers make announcements to a group. With such logics, we may reason about the knowledge of the participants over time and to understand, for instance, the extent to which malicious coalitions of nodes may control the knowledge of other participants. A key research challenge here will be formally specifying the properties of interest in the modal logic, so that these properties can be verified.

An important technique for verification of these programs when considered as part of a larger business process will be “trace alignment” [11]. In this approach, we use formal models (DFA and/or modal logics) to create expected execution runs, or traces, and then we use another formal representation, such as Linear Temporal Logic (LTL), to undertake the task of verification. Techniques from AI Planning may then be used to automatically fix any mis-alignments identified.

3.1 Verification of smart contracts: Questions 2—5

To apply the methods of formal verification to the five questions above, we will need to have certain tools and methodologies in place. For questions 2-5, we will need a formal (machine-readable) representation of a smart contract program and its effects on the world in which it exists. If we view a smart contract as just a computer program operating on some real or virtual machine, ignoring (for the moment) the fact that it operates in a distributed fashion across separate nodes, then this is straightforward, at least in principle. The theory of program language semantics distinguishes several different types of formal representations of programs:

The first type of semantics, called an *axiomatic semantics*, defines each program in terms of the pre-conditions which must exist before the program can be executed and the post-conditions which apply following its execution utterance. A second type of semantics, an *operational semantics*, considers the smart contract as computational instructions which operate successively on the states of some abstract or virtual machine. The commands in the program are thus seen as state-transition operators and the operational semantics defines these transitions precisely. Third, in *denotational semantics*, each element of the language syntax is assigned a relationship to an abstract mathematical entity, its denotation. We then seek to reason about the properties of the program by reasoning about the denoted mathematical entities, and conversely.

Each of these types of semantics may be applicable to particular classes of smart contract, when viewed simply as programs (ie, ignoring the blockchain platform). For most smart contracts, creating both an axiomatic and an operational semantics should be straightforward. An operational semantics will be of most value when exploring the properties of collections of smart contracts (ie, in trying to answer Question 5), as this semantics should facilitate the analysis of effects due to concurrency and interaction. It may be that verifying desirable properties of these programs requires an assumption that the machines which run the contracts all have some common implementation environment, possibly a virtual environment. Thus, proving low-level properties – those which depend on the features of particular hardware or software installations – may be challenging. Likewise, assessment of the actual behaviors of a

smart contract will require knowledge of the specific distributed ledger it will run on. This is particularly so for those DLs which use an internal currency, as these may create run-time incentives or disincentives for particular behaviors.

Smart contracts are intended to read inputs from and write outputs to distributed ledgers, and also perhaps themselves sit upon a ledger. By virtue of the fact that a ledger is shared between multiple machines, then the information recorded there is a form of communication between the nodes of the ledger. Thus, we may also view smart contracts as utterances in a dialog between the participants to the distributed ledger, with the dialog taking place according to some pre-defined protocol which all the participants adopt. Within AI, the last decade has seen considerable research in defining formal protocols for automated communications between autonomous agents (ie, *machine-to-machine* communications). Although not perfect for every application, the most widely-used language is the Agent Communications Language *FIPA ACL* of the standards organization IEEE FIPA [12]. This is a language of 22 atomic utterances, which has been given a formal syntax and semantics. The semantics has been defined, using a modal logic language *SL,* in terms of the beliefs, desires, and intentions of the agents participating in the interaction.

For example, one agent, *A* say, may inform another agent, *B*, of some fact *p* by use of the *inform* locution. The FIPA ACL semantics of *inform* only permits agent A to send this message to B if: (a) agent A believes *p* to be true; (b) agent A intends that agent B believes *p* to be true; and (c) agent A believes that agent B does not already have a belief about *p*. The language ACL has been extended in various ways: for example, the *Fatio Protocol* [13] adds utterances to FIPA ACL to allow agents to argue with one another about some statement, and this has found potential application in identifying and resolving conflicting routing policies in the use of the Border Gateway Protocol at the Network Layer (Layer 3) of the 5-layer Internet Stack [14]. For a review of research on the syntax and semantics of agent communications protocols, see [15].

The blockchain deployed for Bitcoin records exchanges of bitcoin between participants, and *de facto* ownership records. Many of the applications for DL technologies proposed initially aimed to record similar factual information, each of which what philosophers of language would term an *assertion* (a statement purporting to be a fact about the world). Some assertions only become true by virtue of being uttered, as when records validly uploaded as blocks to the Bitcoin Blockchain assert changes of ownership of particular bitcoin. Such assertions are examples of "*speech acts"*, which are utterances that, when executed appropriately, change the state of the world [16]. In exploring potential applications for distributed ledgers, people soon realized that other speech acts besides assertions could also be recorded on blockchains, for example, promises and commands. Indeed, as Szabo's definition of smart contracts quoted earlier makes clear, smart contracts may be viewed as sets of conditional promises: When

some external event occurs or when some statement becomes true, then the program promises to collect some input variables and execute some program language commands, and then export some output variables.

Philosophers of language have long realized that for utterances about actions, such as promises, the pragmatics, or norms of usage, are as important as the semantics, or meaning. Under what conditions, for example, does a promise take effect? In most human cultures, a promise only takes effect when the intended recipient overtly accepts it. Similarly, who has the power to revoke a promise? In most cultures, the maker of a promise that has been accepted cannot unilaterally revoke it as they wish. Only the party accepting the promise may declare it fulfilled or may revoke it. These notions have been explored in the law and more recently in automated communications between machines [17]. They will play an important role in determining that specific program code has desirable properties and not any undesired ones.

3.2 Collections of smart contracts: Question 5

Question 5 concerns verification of system-level properties, where smart contracts may interact with one another, and with other programs. In addition to formal representation of individual contracts and their effects, we will also need representation of the collection as a collective, and its effects. A key learning of the history of information technology systems is that real computer systems operating on the same machine or in the same computer environment may interact with one another in unexpected and unintended ways. In particular, the properties of these complex adaptive systems may not be able to be predicted or inferred from knowing only the properties of the individual components in the system and knowing their rules of interaction. Higher-level properties and phenomena may emerge as more individual programs come into contact with one another.

As enterprises deploy multiple blockchains, some public and some private (ie, permissioned), each running various smart contracts reading inputs from and writing outputs to these different ledgers, predicting and preventing such interaction effects will become increasingly important. However, the science of complex systems applied to collections of computer programs and ecosystems of IT systems is itself still fairly immature, and computer scientists have only just recently begun looking, for instance, at verification of the properties of swarms of autonomous entities [18].

3.3 Natural Language Contracts: Question 2

For automation of the validation task of Question 2, we require both formal representation of the smart contract program and of the natural language contract. One way that this may happen is via creation and application of a descriptive mark-up language for legal contracts that would tag the terms of contracts with formal (machine-readable) semantics. A proposal for a mark-up language of this kind is the Legal Knowledge Interchange Format (LKIF), developed as part of the EC-funded research project *Estrella* [19]. Clack *et al.* [20, 21] have recently proposed a standardised format for storage and transmission of marked-up legal agreements, as part of a framework and meta-language that would incorporate both natural language documents and program code. Much work remains to be done in this area before we would have automation of the validation task of Question 2.

3.4 Shared Understandings and Intentions: Question 1

The tasks involved in the validation task of Question 1 are probably the most challenging of these tasks to automate. The human expertise involved is legal expertise, and for contracts in particular domains, such as commodity futures or trade finance, it may need to be very specialist. However, against that, many of the proposed applications of smart contracts are for repeated and essentially routine applications, where only some parameters vary from one instantiation to the next. The spectrum proposed in [5], for instance, includes cases where the encoded component is merely payment actions or is similarly limited. One may thus imagine using human expertise to, say, create and validate a collection of template natural language contracts along with parameters representing the understandings (beliefs) and intentions of participants. An example could be the choice of legal jurisdiction under which disagreements about the contract would be adjudicated. Such parameters would be chosen by the participants – either individually or jointly – and instantiated into the written contract. A theory of document modeling in legal domains has been proposed by [22], and this could encompass such activities.

We could view these efforts as seeking to automate the creation of, and the provision of legal and computational advice on, smart contracts. These goals would be best understood within the context of a larger and long-running research agenda within AI to formalize legal reasoning, both to understand it and as support for its automation. Limitations of space and scope prevent us pursuing these ideas here.

4. Conclusion

In this paper we have discussed some of the issues involved in the validation and verification of smart contracts running over distributed ledgers. While formal verification of computer programs and systems has a long history in computer science, with many successful commercial applications, there are features of this application domain which create major research and implementation challenges for automated verification and validation of smart contracts. While none of these challenges is insurmountable, they will take time and effort to be resolved.

Acknowledgments
We are grateful for conversations on these topics with Sean Murphy, Michael Sinclair and Victoria Birch of Norton Rose Fulbright, Benjamin Herd of Bosch, Philip Treleaven of University College London, and the anonymous reviewers.

5. References

1. S. Nakamoto [2008]: Bitcoin: A peer-to-peer electronic cash system. Available from: https://bitcoin.org/bitcoin.pdf

2. R. G. Brown, J. Carlyle, I. Grigg and M. Hearn [2016]: Corda: An Introduction. White Paper, R3cev. Available from: https://www.r3cev.com/s/**corda**-introductory-whitepaper-final.pdf

3. M. S.-Y. Chwe [2001]: *Rational Ritual: Culture, Coordination, and Common Knowledge.* Princeton, NJ: Princeton University Press.

4. N. Szabo [1996]: *Smart Contracts: Building Blocks for Digital Markets*.

5. S. Murphy and C. Cooper [2016]: *Can smart contracts be legally binding contracts?* White Paper. London, UK: R3cev and Norton Rose Fulbright.

6. IEEE [2011]: IEEE Draft Guide: Adoption of the Project Management Institute (PMI) Standard: *A Guide to the Project Management Body of Knowledge (PMBOK Guide)-2008* (4th edition). P1490/D1, May 2011. Available from: http://ieeexplore.ieee.org/servlet/opac?punumber=5937009

7. N. Popper [2016]: *"*Hacker may have taken $50 million from cybercurrency project*." New York Times*, 17 June 2016.

8. L. Fix [2008]: Fifteen years of formal property verification in Intel. pp.139-144, in: O. Grumberg and H. Veith (Editors): *25 Years of Model Checking: History, Achievements, Perspectives*. Berlin, Germany: Springer.

9. J. Ezekiel, A. Lomuscio, L. Molnar, M. Pebody and S. Veres [2011]: Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI11)*. Barcelona, Spain. pp. 1659-1664. AAAI Press.

10. J. Plaza [2007]: Logics of public communications. *Synthese*, 158 (2): 165-179.

11. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst [2012]: Process diagnostics using Trace Alignment: Opportunities, issues, and challenges. *Information Systems*, 37(2): 117-141.

12. FIPA [2002]: *Communicative Act Library Specification.* IEEE Foundation for Intelligent Physical Agents. Standard SC00037J. 3 December 2002.

13. P. McBurney and S. Parsons [2004]: Locutions for argumentation in agent interaction protocols. pp. 209-225, in: R. M. van Eijk, M-P. Huget and F. Dignum (Editors): *Agent Communication. Revised Proceedings of the International Workshop on Agent Communication (AC2004)*, LNAI 3396. Berlin, Germany: Springer.

14. P.A. Kodeswaran, A. Joshi, T. Finin and F. Perich [2010]: A declarative approach for secure and robust routing. In: *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*, October 2010.

15. P. McBurney and S. Parsons [2009]: Dialogue games for agent argumentation. Chapter 13 in: I. Rahwan and G. Simari (Editors): *Argumentation in Artificial Intelligence*. Berlin, Germany: Springer, pp. 261-280.

16. J.L. Austin [1962]: *How to Do Things With Words*. Cambridge, MA: Harvard University Press.

17. P. McBurney and S. Parsons [2013]: Talking about doing. In: K. Atkinson, H. Prakken and A. Wyner (Editors): *From Knowledge Representation to Argumentation in AI, Law and Policy Making*. London, UK: College Publications, pp. 151-166.

18. C. Dixon, M. Fisher, A. F. T. Winfield and C. Zeng [2012]: Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems,* 60 (11): 1429-1441.

19. T. F. Gordon [2008]: Constructing legal arguments with rules in the Legal Knowledge Interchange Format (LKIF). In: P. Casanovas, G. Sartor, N. Casellas, and R. Rubino (Editors): *Computable Models of the Law*. LNCS volume 4884. Berlin, Germany: Springer.

20. C. D. Clack, V. A. Bakshi and L. Braine [2016a]: Smart Contract Templates: foundations, design landscape and research directions. Available from: https://arxiv.org/pdf/1608.00771v2

21. C. D. Clack, V. A. Bakshi and L. Braine [2016b]: Smart Contract Templates: essential requirements and design options. Available from: https://arxiv.org/pdf/1612.04496v2

22. M. Lauritsen and T. F. Gordon [2009]: Toward a general theory of document modeling. In: C. D. Hafner (Editor): *12th International Conference on Artificial Intelligence and Law* (ICAIL 2009). ACM Press.

Author Bios

Daniele Magazzeni is Head of the Planning Research Group of the Department of Informatics at King's College London, where he undertakes research in AI Planning and in Model Checking. His research is particularly focused on hybrid systems, robotics, smart cities, and intelligent traffic control. He is an elected member of the Executive Council of ICAPS, the leading conference on Planning and Scheduling.

Contact him at: daniele.magazzeni@kcl.ac.uk

Peter McBurney is Professor of Computer Science and a former Head of the Department of Informatics at King's College London. His research focus includes distributed ledgers, agent (machine-to-machine) communications, and agent-based modeling. He has a first degree in pure mathematics and statistics from the Australian National University and a PhD in computer science from the University of Liverpool, UK.

Contact him at: peter.mcburney@kcl.ac.uk

William Nash is founder and CEO of Kwôri Ltd, a London-based software development and IT strategy firm specializing in distributed ledger technologies. The company's clients include Tier 1 banks and leading global law firms. He has a degree in Philosophy and Physics from King's College London. He previously worked with the Information Security Forum, a British corporate partnership undertaking research on business security.

Contact him at: william.nash@kwori.co.uk