



Validation and Verification of Smart Contracts: A Research Agenda

Daniele Magazzeni and Peter McBurney, King's College London

William Nash, Kwôri

Smart contracts might encode legal contracts written in natural language to represent the contracting parties' shared understandings and intentions. The issues and research challenges involved in the validation and verification of smart contracts, particularly those running over blockchains and distributed ledgers, are explored.

The world of commercial computing and IT has been abuzz recently regarding a new technology, called variously blockchains, distributed ledgers, or shared ledgers. The underlying blockchain technology was invented in 2008 to support the creation and exchange of the Bitcoin cryptocurrency without the need for a controlling central authority.¹ The Blockchain platform and associated cryptocurrency were implemented and launched in 2009. In the years since, people have seen many potential applications for such technology, particularly in finance and government. For instance, in the financial sector, more than 80 global financial institutions have partnered in the R3 consortium, which 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 mid-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 multinode decision protocol; normally, data can only be appended to but not erased from the shared database.

A *blockchain* is a special type of distributed ledger in which 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 can be used to encrypt data, authorize data for upload, and chain blocks together. The Bitcoin blockchain was invented for issuing, executing, and recording transactions of a cryptocurrency, Bitcoin. A distributed ledger can also use or support an electronic currency, even if the database is intended for other purposes. Participation in such a network might be open to anybody,

as is the Bitcoin blockchain, or a node might require preauthorization and perhaps identification to join. Distributed ledgers requiring preauthorization are called *permissioned ledgers*, while open networks are called *permissionless*. Permissioned ledgers will typically enable participants to identify one another, unlike the pseudonymity of many permissionless ledgers. Different applications will require different system architectures and designs.

WHY THE EXCITEMENT ABOUT BLOCKCHAIN?

We can answer this question by comparing blockchain 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 webpage, or by enabling access to the information via a webpage (for example, when stored in a database). A webpage, therefore, is akin to a blackboard, where one person—the controller of the server—has the power to write contents onto the board, as well as to erase contents from it. If the webpage is public, then anyone can read the board's current contents. If the webpage requires access permission (as, for example, subscription newspaper sites do), then the ability to read the contents might be limited to a private group.

Under this metaphor, distributed-ledger technologies (DLTs) provide blackboard readers with some extra powers, at the possible expense of the writer's powers. Distributed ledgers only allow content to be written to the blackboard if the readers first agree, according to some prespecified voting rules. For example, for content to be added, all readers, a majority of readers, or a particular nominated reader might have to agree. Board contents

are added in a manner that links them cryptographically to past board contents, forming the “chain” in blockchain. In addition, board contents are impossible to erase. Indeed, “erasure” might only be possible by starting a new board that is the same as the original blackboard up to the point just prior to where the contents are to be erased. Because this action creates two parallel chains that 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 about blockchain technology. The first consequence is that anyone writing onto the board knows that the other participants (the readers) have seen the content: they had to have read it to vote on it before it was uploaded to the board. So everyone party to the distributed ledger knows the same content (that is, 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*, which makes it impossible for a participant to plausibly repudiate a shared ledger's content 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 require prior participant approval. Hence, the past records are effectively immutable.

Third, the shared ledger is not stored on only one machine, but, rather, copies are stored locally on each of the participants' machines. Every reader has a copy of the board, and these copies stay in sync with one another. This creates

significant challenges for anyone wishing to corrupt or alter the contents, because every participant machine would need to be hacked.

These properties are the origin for the statement that DLTs 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 multistep transactions will not be reversed. Traditionally, parties who did not know one another would use a mutually trusted third party, often a law firm, bank, or other regulated entity, to assist with this by, for example, 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 to subsequently repudiate, corrupt, or reverse the actions undertaken. Of course, the participants might not need a trusted third party to witness their individual transactions, but they will still need to trust the software being used.

SMART CONTRACTS

Anything expressible in digital form can be written on an electronic blackboard. In addition to transaction data, a distributed ledger can 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 another blockchain. If stored on a blockchain and executed, their execution will occur locally at each participant node connected to the blockchain. Because many of the applications envisioned for distributed

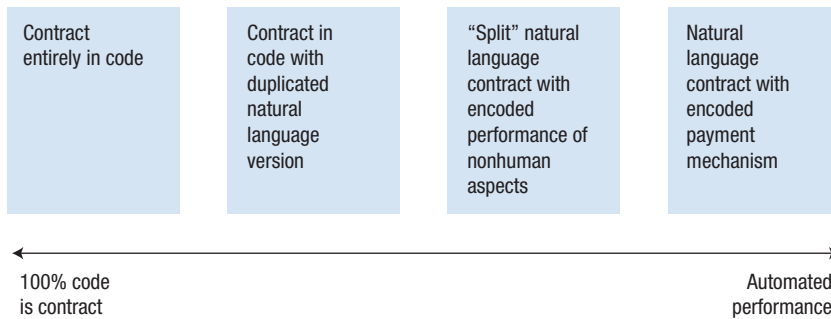


FIGURE 1. Smart contracts lie on a spectrum reflecting the extent to which the contract embodies computer code versus natural language. (Source: Adapted from S. Murphy and C. Cooper, *Can Smart Contracts Be Legally Binding Contracts?*, white paper, R3cev and Norton Rose Fulbright, 2016; www.nortonrosefulbright.com/knowledge/publications/144559/can-smart-contracts-be-legally-binding-contracts.)

ledgers involve coordination between distinct people or organizations, an obvious application of such computer programs is the automatic execution of business workflows across multiple organizations. For example, the legal movement of physical goods from one country to another typically requires export and import permissions from the countries involved. Associated with these permissions might be taxes or duties, evidence of payment of which is required for the relevant permissions to be granted. The vendors and buyers of these goods might require bridging finance to cover delays in receipt of payments or goods, and they might 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, different countries' customs departments, and so on. The workflow could—at least, in theory—be managed by a sequence of computer programs that execute automatically as successive intermediate milestones in the

physical goods' journey are reached.

Such programs are examples of *smart contracts*, a term invented by Nick Szabo before Blockchain's development. 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 can all see these programs and monitor the trade flow progress 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. Whereas a blockchain is a record of who owns what amounts of, for example, a cryptocurrency, smart

contracts on that blockchain can effectively settle transfer of that cryptocurrency in its native form. Whereas assets are dematerialized and then recorded onto a blockchain, smart contracts can perform the same function. These smart contracts have a feature that 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 could 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 levying and payment of fines for noncompliance. Such applications, which Philip Treleven termed *RegTech*, have significant potential for the application of AI to governmental and regulatory activities.

As with distributed ledgers themselves, because the technology is still emerging, smart contracts' nature and form are diverse. For instance, the design of smart contracts might differ significantly according to whether or not the distributed ledger on which they execute is permissioned (because this influences the ability to identify participants), and whether or not the ledger has an internal cryptocurrency. Sean Murphy and Charley Cooper have proposed a spectrum of smart contract definitions according to the extent to which the contract embodies computer code versus natural language (see Figure 1).⁵

The far left-hand of Figure 1 shows agreements between parties that are only represented in computer code form. At each of the other positions,

there is a role for documents written in natural language. The document might duplicate the code, or it might complement it. Even at the far left, where there is no natural language document, the code will execute some intentions of the relevant parties based on some shared understandings, both of which might be implicit.

VALIDATION AND VERIFICATION

From this spectrum, we can identify three components: electronic program code, a natural language contract, and the shared understandings and intentions of the parties. Any particular smart contract might involve any combination of the three components, and each component's role might vary, as indicated in Figure 1. To ensure that smart contract programs execute correctly, we have identified several classes of questions—listed below—corresponding to the relationships between the different components. Figure 2 further illustrates these components and relationships.

1. Does the written natural language contract correctly and fully represent the parties' shared understanding and shared intentions?
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 system operation as a whole perform

without error and only in desirable ways?

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

In terminology adopted by 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 might illustrate these issues. The recent experience of the DAO, a decentralized autonomous organization, is probably well-known to readers.⁷ This open source computer program was created to run over the Ethereum distributed ledger with the aim of pooling crowdsourced funds for investment in start-ups. Throughout May 2016, some 11,000 investors gave cryptocurrency then-valued at more than \$150 million. Although the code did what it was intended to do (question

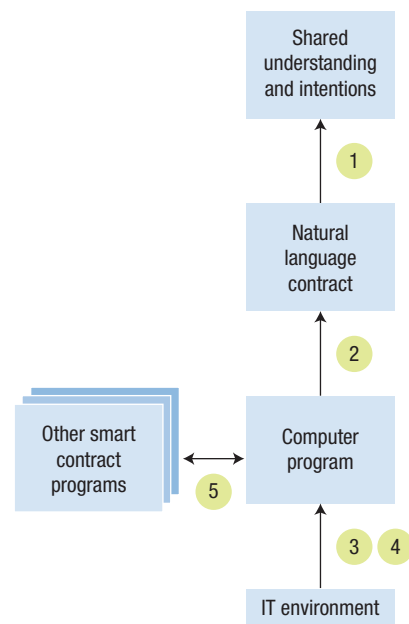


FIGURE 2. Components of a smart contract system. Numbers refer to the validation and verification questions discussed in the main text.

3), it was also 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. So, the program code ran correctly but in a manner unintended by the developers. Because of inbuilt delays in funds transfers, the participant community was able to reverse this, but only by creating a fork of the Ethereum blockchain, that is, 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 human analysis and discussion between the relevant parties. The human expertise required for

questions 1 and 2 would include legal knowledge and experience, while questions 2–5 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 these questions for any proposed smart contract. Is this possible?

Formal and automated verification of computer programs has been an active area of computer science research for at least three decades, with several methods now in use. A common method is *model checking*, in which the program-verification task involves proving properties of a mathematical model of the program. Industry has widely applied these techniques, for example, in the design of computer hardware⁸ and autonomous underwater vehicles.⁹

Most smart contract applications being considered in industry are intended to automate or semiautomate business processes, which are combinations of sequential, parallel, or interleaved actions undertaken over time by multiple participants. For this reason, action sequences and message sequence charts are obvious specification and design tools for the interactions and communications between participants in distributed ledgers, and deterministic finite automata (DFA) are an obvious computational model of their operation. DFA present an external view of the actions of a participant or a smart contract across 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 with temporal and epistemic operators. For example, public announcement logics (such as that described by Jan Plaza¹⁰) represent updates in the knowledge of hearers when speakers make announcements to a group. With such logics, we can reason about the participants' knowledge over time and understand, for instance, the extent to which malicious coalitions of nodes control other participants' knowledge. A key research challenge will be formally specifying the properties of interest in the modal logic so that these properties can be verified.

Trace alignment will be an important technique for verification of these programs when considered as part of a larger business process.¹¹ In this approach, we use formal models (DFA and/or modal logics) to create expected execution runs, or traces, and then another formal representation, such as linear temporal logic, to undertake the verification. Techniques from AI planning might then be used to automatically fix any misalignments identified.

Verification of smart contracts (questions 2–5)

To apply formal verification methods to the five questions, we need to have certain tools and methodologies in place. For questions 2–5, we 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) that it operates in a distributed fashion across separate nodes, then this is straightforward, at least in principle. Program language

semantics theory distinguishes several different types of formal representations of programs.

The first type, *axiomatic semantics*, defines each program in terms of the preconditions that must exist before the program can be executed, and the postconditions that apply following its execution utterance.

A second type of semantics, an *operational semantics*, considers the smart contract as computational instructions that operate successively on some abstract or virtual machine's states. The program commands are thus seen as state-transition operators, and the operational semantics defines these transitions precisely.

Finally, in *denotational semantics*, each language syntax element is assigned a relationship to an abstract mathematical entity, or its denotation. We then seek to reason about the program's properties by reasoning about the denoted mathematical entities, and vice versa.

Each of these types of semantics can apply to particular classes of smart contracts when viewed simply as programs (that is, 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 we are exploring the properties of collections of smart contracts (that is, in trying to answer question 5), because this semantics should facilitate the analysis of effects due to concurrency and interaction. It might be that verifying desirable properties of these programs requires an assumption that the machines running the contracts all have some common implementation environment, possibly a virtual environment. Thus, proving low-level

properties—those that depend on particular hardware or software installation features—might be challenging. Likewise, assessment of a smart contract’s actual behaviors will require knowledge of the specific distributed ledger it will run on. This is particularly true for those distributed ledgers that use an internal currency, because these might create runtime 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 on a ledger. By virtue of the fact that a ledger is shared between multiple machines, the information recorded there is a form of communication between the ledger’s nodes. Thus, we might also view smart contracts as utterances in a dialog between the participants to the distributed ledger, with the dialog occurring according to some predefined protocol that all the participants adopt. Within AI, the last decade has seen considerable research in defining formal protocols for automated communications between autonomous agents (that is, machine-to-machine communications). Although not perfect for every application, the most widely used language is the IEEE Foundation for Intelligent Physical Agents’ (FIPA’s) Agent Communication Language (ACL).¹² This language of 22 atomic utterances 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, agent A might inform 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

- › agent A believes *p* to be true,
- › agent A intends that agent B believes *p* to be true, and
- › agent A believes that agent B does not already have a belief about *p*.

ACL has been extended in various ways; for example, the Fatio Protocol adds utterances to FIPA ACL to allow agents to argue with one another about some statement,¹³ 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 five-layer Internet stack.¹⁴ In their chapter in *Argumentation in Artificial Intelligence*, Peter McBurney and Simon Parsons review the syntax and semantics of agent communications protocols.¹⁵

The blockchain deployed for Bitcoin records exchanges of bitcoin between participants, and de facto ownership records. Many of the initially proposed applications for DLT aimed to record similar factual information—what philosophers of language would term *assertions* (statements purporting to be facts 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*, 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 be recorded on blockchains, for example, promises and commands. Indeed, as Szabo’s definition makes clear, smart contracts might be viewed as sets of conditional promises: when some external

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

Language philosophers 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 he or she wishes. Only the party accepting the promise can declare it fulfilled or 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.

Collections of smart contracts (question 5)

Question 5 concerns verification of system-level properties, where smart contracts can interact with one another and with other programs. In addition to formal representation of individual contracts and their effects, we also need representation of the collection as a collective, and its effects. A key lesson of the history of IT systems is that real computer systems operating on the same machine or in the same computer environment might interact with one another in unexpected and unintended ways. In particular, the properties of these complex adaptive systems might not be predictable or inferable from

the properties of the individual components in the system and their rules of interaction. Higher-level properties and phenomena might emerge as more individual programs come into contact with one another.

As enterprises deploy multiple blockchains, some public and some private (that is, 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

Natural language contracts (question 2)

Automatizing question 2's validation task will require formal representation of both the smart contract program and the natural language contract. One way is via the 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 proposed mark-up language of this kind is the Legal Knowledge Interchange Format (LKIF), developed as part of the European Commission-funded research project Estrella.¹⁹ Christopher Clack and his colleagues recently proposed a standardized 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.^{20,21} Much work remains to be done in this


area before we can achieve automation of question 2's validation task.

Shared understandings and intentions (question 1)

The tasks involved in question 1's validation task are probably the most challenging to automate. Human legal expertise is necessary, and for contracts in particular domains, such as commodity futures or trade finance, such knowledge might need to be very specialized. However, many of the proposed smart contract applications are for repeated and essentially routine applications, in which only some parameters vary from one instantiation to the next. The spectrum proposed by Murphy and Cooper,⁵ for instance, includes cases where the encoded component is merely payment actions or is similarly limited. Thus, one might imagine using human expertise to 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. Marc Lauritsen and Thomas F. Gordon proposed a theory of document modeling in legal domains,²² and this could encompass such activities.

These efforts seek to automate the creation of, and the provision of legal and computational advice on, smart contracts. These goals are best understood within the context of a larger and long-running AI research agenda to formalize legal reasoning—both to understand it and support its

automation. Space and scope limitations prevent us from pursuing these ideas here.

Although formal verification of computer programs and systems has a long history in computer science, with many successful commercial applications, features of this application domain create major research and implementation challenges for automated verification and validation of smart contracts. None of these challenges is insurmountable, but 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 Treleven of University College London; and the anonymous reviewers.

REFERENCES

1. S. Nakamoto, *Bitcoin: A Peer-To-Peer Electronic Cash System*, report, Bitcoin.org, 2008; bitcoin.org/bitcoin.pdf.
2. R.G. Brown et al., *Corda: An Introduction*, white paper, R3cev, 2016; www.r3cev.com/s/corda-introductory-whitepaper-final.pdf.
3. M.S.-Y. Chwe, *Rational Ritual: Culture, Coordination, and Common Knowledge*, Princeton Univ. Press, 2001.
4. N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," *Extropy*, 1996; www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.
5. S. Murphy and C. Cooper, *Can Smart Contracts Be Legally Binding Contracts?*, white paper, R3cev and Norton Rose

ABOUT THE AUTHORS

DANIELE MAGAZZENI is head of the Planning Research Group of the Department of Informatics at King's College London. His research interests include hybrid systems, robotics, smart cities, and intelligent traffic control. Magazzeni received a PhD in computer science and applications from the University of L'Aquila. He is an elected member of the International Conference on Automated Planning and Scheduling (ICAPS) executive council. Contact him at daniele.magazzeni@kcl.ac.uk.

PETER MCBURNEY is a professor of computer science and a former head of the Department of Informatics at King's College London. His research interests include distributed ledgers, agent (machine-to-machine) communications, and agent-based modeling. McBurney received a PhD in computer science from the University of Liverpool. Contact him at peter.mcburney@kcl.ac.uk.

WILLIAM NASH is founder and CEO of Kwōri Ltd., a London-based software development and IT strategy firm specializing in distributed ledger technologies. He received a BSc Honours degree in philosophy and physics from King's College London. Nash previously worked with the Information Security Forum, a British corporate partnership undertaking research on business security. Contact him at william.nash@kwori.co.uk.

- Fulbright, 2016; www.nortonrosefulbright.com/knowledge/publications/144559/can-smart-contracts-be-legally-binding-contracts.
6. IEEE Draft Guide: Adoption of the Project Management Institute (PMI) Standard: A Guide to the Project Management Body of Knowledge (PMBOK Guide) 2008, 4th ed., P1490/D1, IEEE, May 2011; ieeexplore.ieee.org/servlet/opac?punumber=5937009.
 7. N. Popper, "A Hacking of More Than \$50 Million Dashes Hopes in the World of Virtual Currency," *New York Times*, 17 June 2016; www.nytimes.com/2016/06/18/business/dealbook/hacker-may-have-removed-more-than-50-million-from-experimental-cybercurrency-project.html.
 8. L. Fix, "Fifteen Years of Formal Property Verification in Intel," *25 Years of Model Checking: History, Achievements, Perspectives*, O. Grumberg and H. Veith, eds., Springer, 2008, pp. 139–144.
 9. J. Ezekiel et al., "Verifying Fault Tolerance and Self-Diagnosability of an Autonomous Underwater Vehicle," *Proc. 22nd Int'l Joint Conf. Artificial Intelligence (IJCAI 11)*, 2011, pp. 1659–1664.
 10. J. Plaza, "Logics of Public Communications" *Synthese*, vol. 158, no. 2, 2007, pp. 165–179.
 11. R.P. Jagadeesh, C. Bose, and W.M.P. van der Aalst, "Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges," *Information Systems*, vol. 37, no. 2, 2012, pp. 117–141.
 12. *Standard SC00037J, Communicative Act Library Specification*, IEEE Foundation for Intelligent Physical Agents, 3 Dec. 2002.
 13. P. McBurney and S. Parsons, "Locutions for Argumentation in Agent Interaction Protocols, Agent Communication," *Rev. Proc. Int'l Workshop on Agent Communication (AC 04)*, LNAI 3396, R.M. van Eijk, M-P. Huget, and F. Dignum, eds., Springer, 2004, pp. 209–225.
 14. P.A. Kodeswaran et al., "A Declarative Approach for Secure and Robust Routing," *Proc. 3rd ACM Workshop Assurable and Usable Security Configuration (SafeConfig 10)*, 2010, pp. 45–52.
 15. P. McBurney and S. Parsons, "Dialogue Games for Agent Argumentation," *Argumentation in Artificial Intelligence*, I. Rahwan and G. Simari, eds., Springer, 2009, pp. 261–280.
 16. J.L. Austin, *How to Do Things With Words*, Harvard Univ. Press, 1962.
 17. P. McBurney and S. Parsons, "Talking about Doing," *From Knowledge Representation to Argumentation in AI, Law and Policy Making*, K. Atkinson, H. Prakken, and A. Wyner, eds., College Publications, 2013, pp. 151–166.
 18. C. Dixon et al., "Towards Temporal Verification of Swarm Robotic Systems," *Robotics and Autonomous Systems*, vol. 60, no. 11, 2012, pp. 1429–1441.
 19. T. F. Gordon, "Constructing Legal Arguments with Rules in the Legal Knowledge Interchange Format (LKIF)," *Computable Models of the Law*, LNCS 4884, P. Casanovas et al., eds., Springer, 2008, pp. 162–184.
 20. C.D. Clack, V.A. Bakshi, and L. Braine, "Smart Contract Templates: Foundations, Design Landscape and Research Directions," *ArXiv*, 2016; arxiv.org/pdf/1608.00771v2.
 21. C.D. Clack, V.A. Bakshi, and L. Braine, "Smart Contract Templates: Essential Requirements and Design Options," *ArXiv*, 2016; arxiv.org/pdf/1612.04496v2.
 22. M. Lauritsen and T.F. Gordon, "Toward a General Theory of Document Modeling," *Proc. 12th Int'l Conf. Artificial Intelligence and Law (ICAIL 09)*, 2009, pp. 202–211.

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>