Blockchains and the Logic of Accountability

Keynote Address

Maurice Herlihy
Brown University and Oracle Labs
mph@cs.brown.edu

Mark Moir
Oracle Labs
mark.moir@oracle.com

1. Introduction

Bitcoin (21) is in the news, and many of us just can't look away. There are lurid stories of drug trafficking (31), extortion (9), vanishing riches (10), mysterious identities (30), as well as sabotage provoked by disputes over technical standards (32).

Nevertheless, this transient drama should not distract from the possibly transformative effects of the mechanisms underlying Bitcoin and similar systems At the heart of Bitcoin is the *blockchain*, a data structure rendered tamper-proof by cryptographic hashes, and updated by a distributed consensus protocol. Blockchains are likely to transform how the modern world manages financial instruments, property deeds, governance, or indeed any processes whose integrity matters.

This keynote address poses some blockchain-related challenges to the formal methods community. Although blockchains in some form have been around for a long time (20), they have only recently risen to prominence. Blockchain technology has recently been embraced by start-ups (1; 26; 23), financial institutions (5), and technology companies (16). The current state of flux in this area provides the computer science research community an opportunity to influence developments in a way that would be difficult in a more mature area.

Blockchains provide certain kinds of *accountability*: once an agent has performed a particular action, others can be certain that that particular agent indeed performed that action. Accountability can be *proactive*: a Bitcoin balance is transferred from one account only after an agent has established that it is authorized. Accountability also can be *reactive*: a tamper-proof audit trail might allow after-the-fact analysis of agents' actions, determining who did what when, and punishing the guilty. Different kinds of accountability have different consequences: reactive accountability is of little use without some mechanism for penalizing agents for past misdeeds.

There are many ways to reason about what blockchains do. Here, we focus on three kinds of challenges blockchain-based systems are sometimes called on to address.

- *Authorization*: proving that a particular agent is authorized to do something
- *Fairness:* proving that a system is not illegitimately favoring one client over another.
- *Incentives:* proving that a system is correctly creating incentives for clients to behave in a certain way.

The formal methods community is in a unique position to provide a *logic* of accountability to help design, implement, and verify emerging blockchain systems. To be clear: our focus here is not in how to reason about well-understood cryptographic statements such as "agent A is accountable for event x because A signed x with its secret key". Instead, we are interested in reasoning about the *implications* of such statements: "because A has posted bond x, A can now be trusted not to misbehave", or "because A endorsed false statement x, A can no longer be trusted with the nuclear codes". There is a need to *define* useful notions, to provide *rules* for reasoning about such properties, and *tools* for establishing correctness of systems based on such properties.

Rather than propose solutions to these difficult problems, this talk undertakes the much easier task of reviewing a few informally-described (but, we think, realistic) scenarios to test any such logic. Many of these demands are unreasonable, if not provocative, and perhaps impossible, and so we hope you enjoy the discussion.

2. So Many Blockchain Models

There are many, many ways to structure systems that use blockchains. Here, we outline some of the high-level distinctions.

2.1 Permissioned or Permissionless?

Blockchains fall into two broad categories. In *permissionless* blockchains, such as Bitcoin, anyone can participate, and anyone willing to pay the fees can create accounts and propose transactions. A lack of control over who can participate is often an explicit goal of permissionless systems.

By contrast, in *permissioned* applications, participation is controlled by an authority, perhaps one organization, perhaps a consortium. Such control helps comply with "know your customer" regulations, and enables effective governance by providing an orderly procedure for updating the underlying blockchain protocols.

In both kinds of systems, bad behavior can be blocked via proactive accountability mechanisms, say, by ignoring double-spending transactions in Bitcoin. Good behavior can be rewarded, say, by coinbase transactions that pay successful Bitcoin miners. Permissioned systems, however, can also exploit reactive accountability: punishing bad behavior detected after-the-fact, say, by lawsuits, or revoking a rogue agent's right to participate in future transactions. This distinction is not absolute: proof-of-stake (PoS) (3) mechanisms need not be permissioned, but still allow for some degree of reactive accountability. A key advantage of permissioned systems is that that consequences can be based on context (the punishment can fit the crime). In proof-of-stake systems, it may be difficult severely punish egregious transgressions without requiring too much stake to be posted for normal operation.

LICS '16, July 05-08, 2016, New York, NY, USA
Copyright © 2016 held by owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4391-6/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2933575.2934579

2.2 Threat Models

For conventional concurrent data structures, the environment is typically assumed to be benign: concurrent threads make method calls that examine or modify the data structure. To prove correctness, there are well-developed tools and techniques for reasoning about interleavings across layers of abstraction, perhaps in the presence of certain kinds of faults. By contrast, for blockchain data structures, the environment is complex and hostile: sometimes clients must be blocked from bad behavior, but other times it is enough just to provide incentives for them to behave well. Sometimes these incentives take the form of immediate reward (coinbase transactions), and sometimes the threat of later punishment (criminal charges).

Aiyer et al. (2) provide an elegant taxonomy for describing different kinds of client roles. Clients are divided into three classes:

- *Byzantine* clients are capable of arbitrary behavior. This class models malicious or malfunctioning nodes. For example, Bitcoin must guard against Byzantine clients that may propose double-spending transactions.
- *Altruistic* clients can be trusted to follow the protocol exactly, and do not require incentives or policing. For example, today, Bitcoin makes essential use of a peer-to-peer layer whose participants receive no explicit reward. (Indeed, some have claimed that Bitcoin itself runs on altruism (25).)
- *Rational* clients attempt to maximize a known objective function. As long as good behavior yields the best maximization strategy, rational nodes will behave. For example, Bitcoin miners are given incentives to correctly append blocks to the blockchain. They may also display bad behavior such as *selfish mining* (8) or *mining cartel attacks* (22) because doing so may increase their profits, and Bitcoin provides no disincentive for such behavior.

This *BAR* taxonomy was expounded in the context of a shared backup system, where participants fall naturally into these three categories. Blockchain-based applications may be more complex. For example, a single client may be considered altruistic with respect to one observer, but rational or Byzantine with respect to another. Consider a securities trading system run by a financial institution. From the institution's point of view, an internal node is considered altruistic because it is trusted to approve only well-formed transactions. From a suspicious customer's point of view, however, that node would be considered rational if the customer is concerned that the node might covertly inject "front-running" transactions before the customer's stock purchases, allowing the institution to profit from its knowledge of the customer's activities.

In the same way, when considering liveness properties, it often makes sense to view certain clients as rational. For example, Bitcoin miners have an incentive to mine blocks, and therefore the Bitcoin system as a whole is expected to progress. When considering safety properties, however, it often makes more sense to view clients as Byzantine: a malicious miner controlled by a hostile government and therefore unconcerned with profits should not be able to corrupt the Bitcoin ledger.

In *proof-of-stake* protocols (3), agents post a financial bond, and any agent caught cheating forfeits that bond. Such protocols accept as an article of faith that participants are rational, their objective functions known, and that retaining the bond is more important to the agent than any benefit that may follow from corrupting the consensus protocol. Nevertheless, it makes sense to consider the implications of Byzantine failures: if an agent's laptop is hacked, or the owner is threatened by organized crime, how much damage could the agent do? We will have more to say on this topic in Section 5.

2.3 Abstraction

At a concrete level, it can be helpful to think of blockchain specification, implementation, and use as a concurrent data structure problem. So far, however, many blockchain-based systems are defined only by reference implementations, not by any kind of formal specification. Very informally, a blockchain is a concrete data structure that tracks a sequence of *transactions*[1], whose cumulative effects are typically summarized as an abstract data structure we will call a *persistent concurrent ledger*. Here, *persistent* (6) means that the object's past states remain accessible: new information may be appended to the object's state, but old information is never overwritten. *Concurrent* (14) means that the ledger must behave "correctly" when accessed by concurrent, possibly competing agents. Finally, a *ledger* is a table that records a system's state changes, providing ways to query a system's current state, its past states, and in general how it got that way. Traditionally, a ledger keeps track of financial account balances, but here we use the term in a broader sense, keeping track of application-specific state transitions.

While correctness proofs necessarily focus on underlying implementations, the correctness properties to be proved should be expressed in terms of the abstract object being implemented. Proving correctness of a concurrent FIFO queue may require examining a linked-list data structure in detail, but the desired FIFO property is expressed in terms of the abstract queue, not the underlying list. For blockchain-based applications, however, it often remains a challenge to make make such clean separations between layers of abstraction. We would like to specify properties like "Bitcoin miners cannot force excessive reorganizations", or "trading firms cannot inject front-running stock purchases", either deterministically or with high probability, in terms of formal properties of the abstract ledger, not the underlying blockchain implementation, but we don't know how to do that in a clean and rigorous way.

3. Logics of Authorization

Blockchains have a natural role in *authorization*, establishing whether an agent has a right to do something, as well as subsidiary tasks such as *authentication*, identifying one agent to another, and *delegation*, conveying rights from one agent to another. Indeed, there is a rich literature on logics of authentication (4), authorization (11; 18; 12), and delegation (19; 24). Blockchains, however, add substantial new challenges in the form of *smart contracts* (29), defined by Wikipedia to be "computer protocols that facilitate, verify, or enforce the negotiation or performance of a contract, or that make a contractual clause unnecessary."

For example, here is how one could construct a blockchain-based collection of smart contracts for governance of a newly-founded company. Each event is recorded in a blockchain.

- *January:* The company's initial (genesis) block states that the company's board of directors consists of Alice, Bob, and Carol, and that a majority vote of the board is necessary and sufficient to make all governance decisions, including awarding stock options.
- *February:* Carol resigns from the board, and Alice and Bob vote to replace her with Dave.
- *March:* Alice and Dave vote to appoint Ellen as chief compensation officer, and delegate to her the authority to issue stock options.
- *April:* Ellen issues $10000 in stock options to Fred, an employee.

[1] Blockchain transactions should not be confused with database-style transactions used to group multiple operations into a single atomic step.

- *May:* The makes Gina chief compensation officer, replacing Ellen.,
- *June:* Fred exercises his stock option.

As this example shows, smart contracts encompass a kind of *introspection*: a proof that Fred now owns that stock must show not only that the rules in effect at the time were followed at every step, but that the rules themselves were legitimate, having been set in place in accordance with earlier rules, eventually tracing back to the authorizations dictated by the genesis block.

Smart contracts may also make use of programs expressed in a high-level language such as Solidity (7), yielding structures that are potentially far more complex than those produced by conventional access control systems.

Because contracts exist to be legally binding, there is a compelling need to be able to reason about blockchain-based smart contracts. For example, if the board of directors passes such-and-such a rule, are there contingencies where one director can force out the others? Can all the company's obligations be discharged on transfer of control?

Juels *et al.* (17) analyze the feasibility of *criminal smart contracts* (CSCs) that offer rewards for key theft, web site defacement, and similar crimes. In the near future it may be possible to offer rewards for even more serious crimes, such as arson or assassination. One suggested countermeasure against CSCs is to empower a trusted authority (or quorum of authorities) to remove CSCs from the blockchain. A logic of authorization could allow us to express and prove (or disprove) properties such as "this contract remains valid unless it is disabled by a cryptographic warrant signed by appropriate authorities".

This area is bound to be important, and it presents difficult challenges.

4. Logics of Concurrency

For conventional concurrent objects, it is necessary to define what can happen when multiple clients call the object's methods at the same time. One common answer is to say that the method calls are *linearizable* (15): they appear to take place in some one-at-a-time order.

Most blockchain-based distributed ledgers are effectively linearizable. Underlying the ledger, there is a linear list of blocks, where each block contains a set of transactions (method calls). A distributed consensus protocol is repeatedly called to append new blocks to the list, establishing one-at-a-time order in which the transactions appear to take place. (Proof-of-work blockchains, such as Bitcoin's, are more complicated, because one block might be displaced by another, although the probability of such displacement falls rapidly with time.)

For conventional concurrent data structures, linearizability does not specify how concurrent method calls are ordered. For blockchain-based concurrent ledgers, however, the order of concurrent operations may be important. For example, a client for a stock-trading ledger may be concerned that other, more powerful customers' orders might be given preferential scheduling, or, as in the example given above, that the broker may be injecting competing, "front-running" orders ahead of the client's order.

A system like our hypothetical stock trading application must not only be fair, for some application-appropriate notion of fairness, it must be able to *prove* to its customers that it is fair. In recent work (13), we described how to modify the Tendermint (28) ledger to (proactively) prevent certain kinds of unfairness, and to (reactively) detect other kinds of unfairness after the fact. Accountability mechanisms ensure that badly-behaving nodes can be punished. This redesign is based on a few simple principles:

- Each non-deterministic choice dilutes accountability by presenting an opportunity to cheat. As much as possible, non-deterministic choices should be replaced with deterministic rules, such as a pseudo-random number generator whose seed is not under the control of participants. Nodes are held accountable for following all such rules, so violations can be detected, either proactively or reactively.
- When non-deterministic choices cannot be avoided, nodes are held accountable for those choices, ensuring that choices cannot be repudiated later, and that unusual statistical patterns can be detected.

Here is the challenge for the formal methods community: how can we go from common-sense arguments such as those listed above to a formal proof that that a system incorporating these principles guarantees (some form of) fairness? How can we go from simple postulates of the form "A signed x", to a proof that "the broker cannot get away with favoring another client over you"?

5. Logics of Incentives

Many blockchain-based applications rely on *incentives* for liveness (and sometimes even safety) properties. A sound logic of incentives is a difficult problem, perhaps even an impossible one. One conspicuous difficulty stems from the *small-game fallacy* (27): the often-unquestioned assumption that players' objective functions are completely known by system designers.

Eyal and Sirer (8) have observed that under some circumstances, a Bitcoin mining consortium that postpones revealing when it has mined a new block can earn a disproportionate share of the network's mining rewards. Such *selfish mining* illustrates the small-game fallacy: selfish mining would not be a threat if miners only wanted to maximize short-term profits. Nevertheless, a large mining consortium could use selfish mining to threaten smaller consortiums, forcing them to merge, until eventually the consortium controls more than half of the system's computing power. Selfish mining resembles "dumping", where a manufacturer sells its products below its cost of production to drive out competition.

Could formal methods provide a systematic way to detect when such perverse incentives exist? In the conventional world, one effective way to find bugs in a finite program or circuit is to construct a model, and then to use a model checker to verify that desired predicates hold in all executions. Consider the following fantasy technology. Suppose we could construct a finite model for something like Bitcoin mining, complete with incentives, along with a model checker that could verify that desired predicates hold in all executions. One could check predicates of the form "a miner that withholds newly-mined blocks always decreases its own profit" (probably true), or "a miner that withholds newly-mined blocks decreases its own profit at least as much as it decreases the profits of others" (false). Of course, we have no idea how to define such a model, how to characterize and track incentives, or how to check predicates that combine states with incentives. Nevertheless, model checking complements theorem proving by providing counterexamples: if there are circumstances where incentives lead to perverse behavior, only model checking will find them, while an automated proof system would simply fail to prove their absence.

References

[1] Digital asset holding. `https://digitalasset.com/`. Retrieved 6 April 2016.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 45–58, New York, NY, USA, 2005. ACM.

[3] Bitcoinwiki. Proof of stake. `https://en.bitcoin.it/wiki/Proof_of_Stake`. Retrieved 6 April 2016.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, Feb. 1990.

[5] CoinDesk. 8 banking giants embracing Bitcoin and blockchain tech. `http://www.coindesk.com/8-banking-giants-bitcoin-blockchain/`. Retrieved 6 April 2016.

[6] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, Feb. 1989.

[7] Ethereum. Trustless, decentralised, smart contracts for Ethereum. `https://ethereum.github.io/solidity/`. Retrieved 6 April 2016.

[8] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[9] Forbes. As ransomware crisis explodes, Hollywood hospital coughs up $17,000 in Bitcoin. `http://www.forbes.com/sites/thomasbrewster/2016/02/18/ransomware-hollywood-payment-locky-menace/#7db76f5c75b0`. Retrieved 6 April 2016.

[10] Forbes. The inside story of Mt. Gox, Bitcoin's $460 million disaster. `http://www.wired.com/2014/03/bitcoin-exchange/`. Retrieved 6 April 2016.

[11] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. A linear logic of authorization and knowledge. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 297–312, Berlin, Heidelberg, 2006. Springer-Verlag.

[12] V. Genovese, L. Giordano, V. Gliozzi, and G. L. Pozzato. A constructive conditional logic for access control: A preliminary report. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 1073–1074, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[13] M. Herlihy and M. Moir. Enhancing accountability and trust in distributed ledgers. Submitted for publication, Feb. 2016.

[14] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[15] M. Herlihy and J. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990.

[16] IBM. IBM blockchain is open for business. `http://www.ibm.com/blockchain/`. Retrieved 6 April 2016.

[17] A. Juels, A. Kosba, and E. Shi. The ring of gyges: Investigating the future of criminal smart contracts. Cryptology ePrint Archive, Report 2016/358, 2016. `http://eprint.iacr.org/`.

[18] C. Lesniewski-Laas, B. Ford, J. Strauss, R. Morris, and M. F. Kaashoek. Alpaca: Extensible authorization for distributed services. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 432–444, New York, NY, USA, 2007. ACM.

[19] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, Feb. 2003.

[20] R. C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 369–378, London, UK, UK, 1988. Springer-Verlag.

[21] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[22] RHorning. Mining cartel attack. `https://bitcointalk.org/index.php?topic=2227.0`, Dec. 2010.

[23] I. Ripple. Welcome to ripple. `https://ripple.com/`. Retrieved 6 April 2016.

[24] C. Ruan and V. Varadharajan. Dynamic delegation framework for role based access control in distributed data management systems. *Distrib. Parallel Databases*, 32(2):245–269, June 2014.

[25] E. G. Sirer. Bitcoin runs on altriusm. `http://hackingdistributed.com/2015/12/22/bitcoin-runs-on-altruism/`. Retrieved 6 April 2016.

[26] Stellar. Money could move like email. `https://www.stellar.org/`. Retrieved 6 April 2016.

[27] N. Szabo. `http://unenumerated.blogspot.com/2015/05/small-game-fallacies.html`. Retrieved 30 March 2016.

[28] Tendermint. `http:/https://github.com/tendermint/tendermint/wiki`, Oct 2015. Commit c318a227.

[29] Wikipedia. `https://en.wikipedia.org/wiki/Smart_contract`. Retrieved 30 March 2016.

[30] Wikipedia. Satoshi nakamoto. `https://en.wikipedia.org/wiki/Satoshi_Nakamoto`. Retrieved 6 April 2016.

[31] Wired. The rise and fall of silk road. `http://www.wired.com/2015/04/silk-road-1/`. Retrieved 6 April 2016.

[32] Wired. The schism over Bitcoin is how Bitcoin is supposed to work. `http://www.wired.com/2016/02/the-schism-over-bitcoin-is-how-bitcoin-is-supposed-to-work/`. Retrieved 6 April 2016.