TRANSACTIONAL SCRIPTS IN CONTRACT STACKS

Shaanan Cohney[*] *& David A. Hoffman*[**]

Abstract

In conventional transactions, written contracts usually memorialize the terms of the commercial exchange. For deals in which some of the goods being transferred and the forum for the trade are digitized—as in the case of cryptocurrencies—parties may use computer code rather than a written contract to record their terms. Such pieces of code are sometimes called "smart contracts" because they perform many of the same functions as contracts but are expressed in a computing language. Coded exchange embodies a potentially revolutionary contracting innovation. But it is difficult to assimilate into traditional contracting terminology, conceptual framing, and doctrine.

This Article begins by distilling the central legally and practically significant category of smart contracts—what we call "transactional scripts." It then develops an account of how these scripts are created, including significant costs of storage, and how they produce errors of legal significance. This account, in turn, allows us to more rigorously and accessibly situate transactional scripts in existing legal doctrine.

Commentators are enthusiastic about scripts because, the story goes, they lack some of the features that require adjudication. Yet we show that optimism to be unfounded by documenting how scripts, like ordinary contracts, can result in misunderstanding, frustrated intent, and failure.

When code misdelivers, disappointed parties will seek legal recourse. We argue that jurists should situate scripts within other legally operative statements and disclosures, or contract stacks. Precision about the relationship between script and stack sustains a novel framework, rooted in old doctrines of interpretation, parol evidence and equity, that will help jurists compile answers to the private law problems that digitized exchange entails.

[*] Ph.D., Postdoctoral Research Associate, Center for Information Technology Policy, Princeton University

[**] Professor of Law, University of Pennsylvania Law School. We thank Alexander Altieri and Chrissy Pak for research assistance, and Bridget Fahey, James Grimmelmann, Greg Klass, Raina Haque, Drew Hinkes, Gabe Kaptchuk, Max Raskin, Jeremy Sklaroff, Tim Swanson Andrea Tosato, Alec Webley, Kevin Werbach and participants in the "Transactional Scripts and Legal Order" workshop for comments. This work was supported in part by a grant from the Ripple Research Fund at the Wharton School.

INTRODUCTION

In early 2019, a group of people founded Edgeware, a blockchain-based platform to host software development.[1] Edgeware made potential users a deal: if they agreed to temporarily sequester some cryptocurrency (essentially, placing an initial investment in escrow), they'd gain governance rights over the platform at a later date.[2] The mechanism for that investment was a piece of carefully-audited software, called "Lockdrop," written in the Solidity programming language and deployed on the Ethereum blockchain.[3]

By July of 2019, investors committed $300,000,000 to the project. Then someone looked with particular care at the following piece of code:[4]

```
assert(address(lockAddr).balance == msg.value);
```

For a series of technical reasons, this line had the potential to prevent the transactions from occurring as they were represented and could thus permanently impound investor assets.[5] Luckily, this error was discovered before it caused harm. But, as the coder who discovered the vulnerability put it, "smart contracts are software. Even carefully audited, well tested software will (almost always) contain bugs. Therefore, and despite our best efforts … Smart contracts will (almost always) contain bugs!"[6] The question we ask in this paper is simple: do these bugs have legal consequences? The answer is likewise simple: they do.

---

[1] Commonwealth Labs, *Edgeware: An Adaptive Smart-Contract Blockchain*, Draft v0.99 (Feb. 25, 2019), https://arena-attachments.s3.amazonaws.com/3850782/1928e31873075de95992d4987eb14a2e.pdf?1552432633.

[2] *Id.* at 5; Brady Dale, *Lock Up Ether, Get Free Crypto: Twist on Airdrops Attracts Top VCs* (Mar. 20, 2019), https://www.coindesk.com/1confirmation-canaan-partners-back-startup-pushing-new-spin-on-airdrops. For an introduction to cryptocurrencies, *see* Shaanan Cohney, David A. Hoffman, Jeremy Sklaroff & David Wishnick, *Coin-Operated Capitalism*, 119 COLUM. L. REV. 591, 603-605 (2019); *see also* DAVID FOX & SARAH GREEN, CRYPTOCURRENCIES IN PUBLIC AND PRIVATE LAW (Oxford 2019).

[3] Quantstamp, *Certificate of Smart Contract Audit for Edge-lockdrop* (Apr. 08, 2019) , https://arena-attachments.s3.amazonaws.com/4282493/a155dc84aa1dfba4cfd3dc6be1e1ebdc.pdf?1557965252.

[4] Neil McLaren, *Gridlock (A Smart Contract Bug),* MEDIUM (Jul. 1, 2019), https://medium.com/@nmcl/gridlock-a-smart-contract-bug-73b8310608a9.

[5] Essentially, anyone with access to the internet could easily frustrate the mechanism, freeze the sequestering process, and prevent previous senders from extracting their assets If the attacker designated cryptocurrency to the next lock address, the total balance would exceed the amount sent by Lockdrop, causing the check to fail.

[6] *Id*. The Edgware project itself has continued to have problems. https://finance.yahoo.com/news/edgeware-blockchain-launch-gets-hijacked-152544998.html

But to their promotors, even buggy "smart contracts" like Lockdrop are the vanguard of a revolution, heralding an age in which code will depose both contract theory and practice.[7] Enthusiasts boast that "smart contracts" are a potential solution to an astounding variety of business and social problems. They may transform insurance,[8] financial derivatives,[9] consumer protection,[10] corporate governance,[11] tax filing,[12] voting,[13] supply chain management,[14] bankruptcy,[15] property rights,[16] and repossession through the internet of things.[17] But there's more. Jurisprudence—in the sense of the fundamental utility of contract doctrine—is on the chopping block.[18] For many, smart contracts are the first transformative legal innovation of the millennium.[19]

Perhaps inevitably, "smart contracts," a term that connotes money, computers, and intelligence has invited a stampede of commentators to speculate about a wide variety of possible

---

[7] Primavera De Filippi & Samer Hassan, *Blockchain Technology as a Regulatory Technology: From Code Is Law to Law Is Code*, FIRST MONDAY (Dec. 5, 2016), https://firstmonday.org/ojs/index.php/fm/article/view/7113/5657 ("[S]mart contracts are actually meant to replace legal contracts."); Mark Verstraete, *The Stakes of Smart Contracts*, 50 LOY. CHI. L. J. 743 (2019) ("By consensus, smart contracts are a revolution in private ordering.").

[8] *Smart Contracts: 10 Use Cases for Business*, AMBISAFE, https://ambisafe.com/blog/smart-contracts-10-use-cases-business/ (last visited Nov. 9, 2019).

[9] *See generally* Int'l. Swaps & Derivatives Ass'n., *Legal Guidelines for Smart Derivatives Contracts: Introduction* (2019), https://www.isda.org/2019/01/30/legal-guidelines-for-smart-derivatives-contracts-introduction/.

[10] *See generally* Joshua Fairfield, *Smart Contracts, Bitcoin Bots, and Consumer Protection*, 71 WASH. & LEE L. REV. ONLINE 35 (2014).

[11] Fiammetta S. Piazza, *Bitcoin and the Blockchain as Possible Corporate Governance Tools: Strengths and Weaknesses*, 5 PENN STATE J. L. & INT'L. AFFAIRS 262, 282-286 (2017).

[12] Valentyn Vishnevsky & Viktoria Chekina, *Robot vs. Tax Inspector, or How the Fourth Industrial Revolution Will Change the Tax System: A Review of Problems and Solutions*, 4 J. TAX REFORM 6, 20 (2018).

[13] *See* Tsui S. Ng, *Blockchain and Beyond: Smart Contracts*, A.B.A. BUS. L. TODAY (Sept. 2017), https://www.americanbar.org/groups/business_law/publications/blt/2017/09/09_ng/ ("Governments may use smart contracts to manage . . . e-voting.").

[14] *See, e.g.,* Horst Treiblmaier, *The Impact of the Blockchain on the Supply Chain: A Theory Based Research Framework and a Call for Action*, 23 SUPPLY CHAIN MGMT. INT'L. J. 545 (2018).

[15] *See generally* Alan Rosenberg, *Automatic Contracts and the Automatic Stay: A Primer on "Smart Contracts" in Bankruptcy*, 38 AM. BANKR. INST. J. 18 (2019).

[16] Michael Casey, *Could Blockchain Technology Help the World's Poor?,* WORLD ECON. FORUM AGENDA (Mar. 9, 2016), https://www.weforum.org/agenda/2016/03/could-blockchain-technology-help-the-worlds-poor.

[17] For the canonical description, *see* Jeremy M. Sklaroff, *Smart Contracts and the Cost of Inflexibility*, 166 U. PA.L. REV. 263, 271-78 (2017).

[18] For a lucid review, *see* Marco Dell'Erba, *Demystifying Technology: Do Smart Contracts Require a New Legal Framework? Regulatory Fragmentation, Self-Regulation and Public Regulation*, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3228445

[19] *See generally* Alexander Savalyev, *Contract Law 2.0: "Smart" Contracts as the Beginning of the End of Classic Contract Law*, 26 INFO. & COMMC'N. TECH. L. 116 (2017). See also *What is Ethereum*, ETHERSCIPTER, https://etherscripter.com/what_is_ethereum.html (last visited Nov. 9, 2019) (describing popular smart contract Ethereum as a "a new kind of law" that can be "perfectly observed and enforced").

contracting technologies.[20] Many marvel at the innovation as some kind of utopian simulacra:[21] a self-executing,[22] irreversible,[23] transparent exchange,[24] occurring through "consensus," that is "self-executing"[25] or "automated."[26] With that capacious definition in mind, is the chip in your Visa Card part of a smart contract network? The code comprising the Venmo App? Your Metrocard?[27] If we can't understand the scope of this phenomenon, how can we fairly evaluate the risk it poses, or the benefits it promises, to our commercial and social life?

Smart contracts' flabby meaning is not this Article's precise target.[28] Rather we offer a focused description of the most celebrated aspect of the underlying technology and its relationship to legal order. We'll start by introducing a new term that we think captures what most people in this field think of when they consider "smart contracts" as they are currently

---

[20] The fish rots from the head. *See* NICK SZABO, SMART CONTRACTS: BUILDING BLOCKS FOR DIGITAL MARKETS (1996) (defining smart contracts as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises"). In recent years, Szabo has vociferously defended the term. *See, e.g.* Nick Szabo, @NickSzabo4, TWITTER (Oct. 14, 2018, 6:38 PM), https://twitter.com/NickSzabo4/status/1051603179526270976 ("'Smart contract' is a very useful concept & phrase. 'Smart' as in 'smart phone' (shorthand for computerized phone), 'contract' meaning it does some important things we previously relied on contracts to do for our deals, especially controlling assets & incentivizing performance.").

[21] Frank Pasquale, *A Rule of Persons, Not Machines: The Limits of Legal Automation*, 87 GEO. WASH. L. REV. 1, 24-25 (2019).

[22] Werbach and Cornell argue that smart contracts are distinctive because "juridical forums are powerless to stop the execution of smart contracts—there is no room to bring an action for breach when breach is impossible." Kevin Werbach & Nicolas Cornell, *Contracts Ex Machina*, 67 DUKE L J 314, 332 (2017). A different leading account focuses on decentralized execution as well. "[S]mart contracts are digital contracts allowing terms contingent on decentralized consensus that are tamper-proof and typically self-enforcing through automated execution."; Lin Wiliam Cong & Zhiguo He, *Blockchain Disruption and Smart Contracts*, 32 REV. FIN. STUD. 1754 (2019).

[23] Jeffrey J. Lipshaw, *The Persistence of "Dumb" Contracts*, 2 STAN. J. BLOCKCHAIN L. & POL'Y. 1, 22 (2019) (saying that smart contracts' key characteristics "in addition to consensus, include immutability and finality from the time they are created and going forward").

[24] Adam Kolber, *Not-So Smart Blockchain Contracts and Artificial Responsibility*, 21 STAN. TECH. L. REV. 198, 208 (2018) ("easy to verify program execution").

[25] Amy J. Schmitz & Colin Rule, *Online Dispute Resolution for Smart Contracts*, 2019 JDR 103, 113 (2019)113 (describing smart contracts as "self-enforcing," "self-governing," and with "no ambiguity around the parties' obligations").

[26] For an example, consider this oft-cited definition: "A smart contract is an automatable and enforceable agreement. Automatable by computer, although some parts may require human input and control. Enforceable either by legal enforcement of rights and obligations or via tamper-proof execution of computer code." Christopher D. Clack, Vikram A. Bakshi, & Lee Braine, *Smart Contract Templates: Foundations, Design Landscape and Research Directions*, ARXIV: 1608.0071v3, 2 (2017). *See also* Max Raskin, *The Law and Legality of Smart Contracts*, 1 GEO. L. TECH. REV. 305 (2017) ("A smart contract is an agreement whose execution is automated.").

[27] Kolber, *supra* note 24, at 208.

[28] *Cf.* Ed Felten, *Smart Contracts: Neither Smart nor Contracts?*, FREEDOM TO TINKER (Feb. 20, 2017), https://freedom-to-tinker.com/2017/02/20/smart-contracts-neither-smart-not-contracts/.

deployed. We name and describe the *transactional script*.[29] Here is a parsimonious definition (that we'll unpack later):[30]

> *A transactional script is a persistent piece of software residing on a public blockchain. When executed as a part of an exchange, the code effectuates a consensus change to the state of a ledger.*

We stress that our focus is narrower than all digitized exchanges,[31] or even all deals accomplished through blockchain-style ledgers.[32] That is, transactional scripts sit at the core of the rapidly expanding group of things called "smart contracts," but do not encompass the whole field. Crucially, transactional scripts are exchanges that operate in *public*—indeed, they are valuable in large part because their resolution must be agreed to by multiple different entities, jointly operating on an operating system with fixed and translucent rules.

Transactional scripts are a striking legal innovation and much of the interest in smart contracts, we will show, is in fact in transactional scripts.[33] But law (and lawyers') role in the creation and execution of scripts is unclear. Even apart from the

---

[29] For a definition of smart contracts that in some ways tracks with our transactional script, *see* Carla L. Reyes, *If Rockefeller Were a Coder*, 87 GEO. WASH. L. REV. 373, 383 (2019) ("The term 'smart contract' refers to decentralized computer code that runs on a DLT protocol and manifests some combination of the following characteristics: exerts some control over assets digitally recorded on a DLT protocol, takes some action upon receipt of specified data, is often, but not always, part of a DLT-based application, guarantees execution, and writes the resulting state change from the operation of the smart contract into the DLT's ledger.") (cleaned up).

[30] *Cf.* Peter G.L. Hunn, *Smart Contracts as Techno-Legal Regulation*, 7 J. ICT STANDARDIZATION 269, 275. (2019) (focusing on "a deterministic state machine" and a "consensus protocol" that provides agreement "on the same sequences of operations.")

[31] For a tremendous survey of that topic, *see* Harry Surden, *Computable Contracts*, 46 U.C. DAVIS L REV. 629 (2012). Surden discusses "autonomous computable contracting" only briefly. *Id.* At 695.

[32] The reader would benefit from reading the following excellent works on scripts (broadly defined). Sklaroff, *supra* note 17; J.G. Allen, *Wrapped and Stacked: "Smart Contracts" and the Interaction of Natural and Formal Language,* 14 EUR. R. CONTRACT L. 307 (2018); Werbach & Cornell, *supra* note 22; Karen E. C. Levy, *Book-Smart, Not Street-Smart: Blockchain-Based Smart Contracts and The Social Workings of Law*, 3 ENGAGING SCI. TECH. & LAW 1 (2017); James Grimmelmann, *All Smart Contracts Are Ambiguous,* 2 J. L. & INNOVATION 1 (2019); PRIMAVERA DE FILIPPI & AARON WRIGHT, BLOCKCHAIN AND THE LAW: THE RULE OF CODE (Cambridge 2018); Jonathan Rohr, *Smart Contracts in Traditional Contract Law, Or: The Law of the Vending Machine*, 67 CLEVELAND ST. L. REV. 67 (2019); Edmund Schuster, *Cloud Crypto-Land*, 83 MODERN L REV., ___, at 23-25 (forthcoming 2020).

[33] *See Smart Contracts Market Research Report - Global Forecast to 2023*, MARKET RESEARCH FUTURE (Nov. 2019), https://www.marketresearchfuture.com/reports/smart-contracts-market-4588 ("The global smart contracts market is expected to reach approximately 300 USD Million by the end of 2023 . . . ."). *Cf. Venture Capital Firms Go Deep and Wide with Blockchain Investments*, 2 DIAR 39 (2018) ("[I]n just . . . three quarters of 2018, blockchain and crypto companies have raised nearly 3.9Bn through traditional VC – 280% more [than] last year . . . .").

performative techno-libertarian claims about law's abnegation, many scripting projects are missing many of the accoutrements of transactional law. More plainly put, currently deployed transactional scripts—those that are exposing real people to financial and personal risks—often rely on the code alone as their primary risk-allocation mechanism. And the code errs.

In the Edgeware project, none of the governance promises were expressed in a natural language "contract" as we understand that term, even in the digital sphere, as in a click-wrap agreement. They existed rather in a "white paper"—a self-published document with no standard form and untested legal effect.[34] That white paper states that users will receive "Downside protection" in the case of "malicious attack/exploitation," and describes the lockdrop contract as a failsafe technique.[35] Moreover, it articulates the belief of the promotors that it is impossible to falsely claim ownership for an address.[36] Needless to say, these representations do not match the reality of what the code delivered. But, equally obviously, since the code was public, its latent errors were also theoretically knowable, at least to sophisticated counterparties.

Other examples of coding errors and oversights abound—including the now infamous DAO hack, which we will discuss later.[37] In many cases, the losers of coding errors have paid off the winners to undo transactions.[38] Such settlements occur in the very indistinct shadow of law: there simply isn't a body of cases that directly address the question of what happens when transactional scripts go wrong.[39] The academic literature too has largely downplayed the role and capabilities of law in resolving transactional scripts gone wrong.[40]

---

[34] Commonwealth Labs, *Edgeware: An Adaptive Smart-Contract Blockchain* 9-15 (drft. v0.99 2019) (https://certificate.quantstamp.com/view/edge-lockdrop), (hereinafter " Edgeware Whitepaper").

[35] Edgeware Whitepaper, *supra* note 34, at 6-7.

[36] *Id*. at 11-12 ("Upon inspection [of the transaction hash], a verifier should have enough proof that if the owner of the . . . account did not also own the recipient Edgeware account (represented as the target address . . .), then they would not issue such a transaction.").

[37] *See infra* at text accompanying notes 231 through 242.

[38] The literature has noted that often firms price bug bounties too low when compared with their after-market (and illegitimate) use. Lorenz Breidenbach, Philip Daian, Florian Tramer & Ari Juels, *Enter the Hydra: Toward Principled Bug Bounties and Exploit-Resistant Smart Contracts*, PROCEEDINGS OF THE 27TH USENIX SECURITY SYMPOSIUM, 978-1-939133-04-5 (2018). It is difficult to know how common self-help is in the world of transactional scripts. Victims have little incentive to publicize their failure to write better code, while successful attackers may wish to avoid public renown (if not the tax authorities).

[39] *See, e.g*. Lauren Henry Scholz, *Algorithmic Contracts*, 20 STAN. TECH. L. REV. 128, 141 (2017) (discussing the application of contract and agency law to algorithmic contracts and noting lack of caselaw).

[40] *Cf.* Dell'Erba, *supra* note 18, at 21 ("It could be that there may be a bug in the code or that the parties may reconsider what they want.")

Even the most sophisticated treatments in the literature largely focus on what happens when scripts accomplish their promised aims.[41] Some argue that the technology is simply incapable of the sorts of error that law cares about: the "[c]ode typically entails no ambiguity, and no variant interpretation is possible."[42] Others lament that because the blockchain code is self-contained, it contains no place within it for "default law" to exist.[43] Worse, even if there are places for law's tools to have purchase, jurists "may not be able to hypothesize a reasonable human's interpretation of a given smart contract."[44]

Such handwaving is an unwarranted, and ultimately unworkable, surrender of the law's role in adjudicating disputes. Code that embodies commercially-significant scripts will inevitably contain ambiguities, and disappointed parties will ask judges to adjudicate their rights. At the basic level, as James Grimmelmann has recently observed, the "meaning of any specific program rests on a foundation of some prior agreement about how to interpret some larger class of programs."[45] He concludes that while there may be fewer superficial examples of interpretative gaps in formal code, "when the bottom drops out, it can really drop out."[46] It's then that law will be asked to step in and provide rational and predictable solutions, which will almost certainly be developed by analogy to off-chain exchanges.

But analogies can deceive. Whereas scholars and jurists who confront questions about ordinary written contracts have a deep working knowledge of how such exchanges function, transactional scripts are functionally innovative and the legal community has yet to coalesce around even a basic understanding of what they are, let alone what they do. What's needed, then, is a working knowledge of the ways in which the vocabulary and functioning of the code itself can create disconnects between the intent of the humans coding the transactional scripts and the code's function.[47] Describing the plumbing of these phenomena is

---

[41] *Compare* Werbach and Cornell, *supra* note 22, at 350-364 (things going well) with *id.* at 365-367 (discussing problems in algorithmic expression).

[42] Wulf A. Kaal & Craig Calcaterra, *Crypto Transaction Dispute Resolution*, 73 BUS. LAW. 109, 136 n.94 (2017-2018).

[43] Usha R. Rodrigues, *Law and Blockchain*, 104 IOWA L. REV. 679, 682 (2019) ("Because the smart "contract" is code alone, there is no gap, in the sense of an entry point, for the law to step in to fill.")

[44] Kaal & Calcaterra, *supra* note 42, at 136; Schmitz & Rule, *supra* note 25, at 104 ("Those with no coding background cannot easily interpret a smart contract in its rawest form.")

[45] Grimmelmann, *supra* note 32, at 12.

[46] *Id*. at 20. He continues that the community can "redefine the programming language in a way that radically alters the meaning of programs written in it."

[47] On the problems caused by blockchain jargon, *see* Angela Walch, *Blockchain's Treacherous Vocabulary: One More Challenge for Regulators*, 21 NO. 2 J. INTERNET L. 1 (2017).

the first contribution of this Article, and occupies Part I. We make two fundamental observations.

First, the coding language development environment of almost all extant transactional scripts has features (and entails uncommon practices) which make coding errors, and gaps between coders' intent and expression, both likely to occur and difficult to resolve.[48] *Second*, the dominant platform for scripts—Ethereum—assesses a tax on complex programs. This fee makes transactional scripts unsuited for many knotty contracting problems, unless their drafters make parts of the code non-public, thus stripping scripts of much of what makes them an elegant solution to problems of trust in exchange.

With a better grasp of how scripts operate, Part II uses case examples to argue—contrary to the dominant account—that these scripts fail in ways that are legible to traditional contractual frameworks. Put simply, they fail to accomplish their parties' intent. Those examples start with "tokens" issued in "initial coin offerings," continue with decentralized exchanges, and finally consider so-called "oracles." We describe both the coded and natural language promises accompanying these projects. Each such category of transactional script has been touted as a revolutionary financial innovation. Each, as we'll show, have already produced errors with real legal consequences.

In Part III, we ask how law ought to respond to the sorts of problems occasioned by such systematic failures of transactional scripts. We argue that the layering of transactional script and semantic agreement is best understood as producing a *contract stack*. Contract stacks are inevitable when parties come together to accomplish commercial ends, even if they try their hardest to make the code the final answer to all their problems. Collecting the meager extant caselaw, and deploying old fashioned rules of interpretation, we offer a novel framework through which courts ought to compile such stacks and thus make sense of these new forms of commercial exchange.

I. DESIGNING EXPENSIVE, BUGGY SCRIPTS

We begin with a precise account of the creation and function of transactional scripts. To the extent that our discussion requires

---

[48] *See* Elaine Ou, *Blockchain is Littered with Smart Contracts Gone Bad*, INS. J. (Nov. 16, 2017), https://www.insurancejournal.com/news/international/2017/11/16/471387.htm ("Last week, more than $150 million worth of ether, the platform's currency, ended up stuck in the wallets — forever . . . . Only a few months earlier, a bug . . . allowed hackers to run off with $32 million. Shortly before that, a Canadian exchange accidentally trapped $13 million in its own broken smart contract.")

struggling with new jargon, our petard has been well-hoisted.[49] Our goal is not merely to demystify this technology, but also the world and social practices of coders whose work increasingly matters to law and transactional practice. We start, then, with the functional question of how coders go about their work.

A. How (Commercial) Coding Works

The process of taking a programmer's intent from code to execution involves multiple steps, each of which can and does introduce error. Usually a programmer starts with a human-driven goal. This might be to make a lottery algorithm that randomly chooses a user, and transfers an amount of cryptocurrency, or another that takes an upfront payment in return for later repayment with interest. A programmer then chooses a language in which to code. Typically, coders will choose a high-level language that abstracts away the details of the machine's hardware and allows programming in syntax closer to natural language.

Transactional scripts are commonly coded in Solidity, the language conceived alongside of, and for use on, Ethereum, the platform on which most scripts operate.[50] It is syntactically similar to the popular, web-development language, Javascript and thus looks familiar to many non-smart-contract coders. To create a transactional script in Solidity, a coder creates a text file with contents that conform to the publicly available specification of the Solidity language, and that capture in programmatic form the design goal.[51]

Like traditional contract drafters, transactional script developers freely use boilerplate. A significant fraction of code in open source software projects is derived from existing sources of similar code, whether in the same or other projects.[52] This practice is also prominent within the transactional scripts world, with one study finding in 95% of implementations that performed a

---

[49] *Cf.* ELIZABETH MERTZ, THE LANGUAGE OF LAW SCHOOL: LEARNING TO "THINK LIKE A LAWYER" (2007) (the classic text on how law students are taught to think and argue in distinctive ways).

[50] For a good overview of the coding ecosystem, *see* Raina Haque, Rodrigo Seira, Brent Plummer and Nelson Rosario, *Blockchain Development and Fiduciary Duty*, 2 STAN. J. BLOCKCHAIN L. & POL. 139, 146-51 (2019).

[51] For more on the process of open source development in blockchain generally, *see* Angela Walch, *Open-Source Operational Risk: Should Public Blockchains Serve as Financial Market Infrastructures?* in David Lee, Kuo Chuen and Robert D Deng (eds), HANDBOOK OF BLOCKCHAIN, DIGITAL FINANCE, AND INCLUSION Vol. 2, 252-254 (Elsevier Academic Press 2017)

[52] Developers report a mean 30% of functionality is derived from reused code, *see* Sojer and Henkel, *Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments*, JOURNAL OF THE ASSOCIATION FOR INFORMATION SYSTEMS https://pdfs.semanticscholar.org/84f7/9b045d79c6af2f20aab35547205011743f0d.pdf

common function, coders had used identical syntax.[53] Reusing or retrofitting code for new ends serves multiple purposes for developers. Reuse decreases development time, may provide ready-made solutions for difficult coding problems, may provide code known to be secure, and may ease compatibility concerns. Code reuse can also cause problems, making developers reliant on code they may not understand, propagating bugs, and increasing development time where reused code is difficult to understand or adapt.[54]

Knitting together these pieces of boilerplate with bespoke code is an iterative process, and programs are created across multiple sessions consisting of coding and testing. To keep track of changes, and the contributions of many individuals,[55] software projects use version control systems that themselves capture a log of each change.[56] To add a change to a version control system, coders are typically required to explicitly identify the change/s they wish to add, describe it in a "commit message" and send it to a server that tracks the full set of changes across users.[57]

---

[53] *See* Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey, *Erays: Reverse Engineering Ethereum's Opaque Smart Contracts*, , USENIX Security 2018, Table 1. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-zhou.pdf

[54] https://consensys.github.io/smart-contract-best-practices/general_philosophy/ ("A smart contract system from a software engineering perspective wishes to maximize reuse where reasonable.")

[55] Programmers will sometimes pair program, wherein a reviewer assesses each line of code as the primary programmer is typing. The reviewer suggests changes, spots errors and often drives the strategic direction of the code. Williams, Laurie, Robert R. Kessler, Ward Cunningham, and Ron Jeffries, *Strengthening the case for pair programming*. IEEE software 17, no. 4 (2000): 19-25.

[56] Distributed version control systems have much in common with blockchains. They record sequences of changes to a common, replicated record using a chained hash structure that ties together each new change with the complete past history. They do not however solve consensus problems, relying on individuals to determine the outcome of conflicts.

[57] While decentralized version control systems (DVCS) such as git can in theory operate without a centralized server, the convenience of a host that is always online and authoritative ensures that most deployments use such a server. Common commercial services providers of these servers include Github, Gitlab and Bitbucket.

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

   Simplify serialize.h's exception handling

   Remove the 'state' and 'exceptmask' from serialize.h's stream
   implementations, as well as related methods.

   As exceptmask always included 'failbit', and setstate was always
   called with bits = failbit, all it did was immediately raise an
   exception. Get rid of those variables, and replace the setstate
   with direct exception throwing (which also removes some dead
   code).

   As a result, good() is never reached after a failure (there are
   only 2 calls, one of which is in tests), and can just be
replaced
   by !eof().

   fail(), clear(n) and exceptions() are just never called. Delete
   them.
```

Figure 1: A well written commit message from the Bitcoin core repository explaining the changes made and the reasoning behind them.[58]

When added to the system, the set of changes is termed a "commit." The commit log generated by a version control system thus contains evidence of the drafting process, both in code and human readable form. The log and code are available to all developers on a project and may optionally be stored on a publicly accessible server.

Large firms enforce discipline over this commit log system: commit messages must capture the intent and effect of a change. In smaller development outfits, programmers often fail to include meaningful commit messages and the log may thus poorly capture intent.[59]

Another prominent practice common to well-disciplined development teams is *code review*. Before a server or team will accept a commit, it passes through human review. The review is performed by other team members, who comment on a platform integrated with the version control system. The reviewers assess all elements of the commit—the message, the quality of the code implementing the change and the intent of the change—and either accept, reject, or send the commit back for further review.

Once a program is ready to be tested, it must be converted from the high-level language to machine instructions. The

---

[58] https://web.archive.org/web/20200116162012/https://github.com/bitcoin/bitcoin/commit/eb0b56b19017ab5c16c745e6da39c53126924ed6.patch

[59] Short-form commit messages can range from the insightful, "Simplify serialize.h's exception handling," to the banal, "fuck fuck holy shit fuck I think I finally fixed my shitty git fuck." *See* Ramiro Gómez, *Exploring Expressions of Emotions in GitHub Commit Messages*, https://geeksta.net/geeklog/exploring-expressions-emotions-github-commit-messages/

conversion is done through the aid of a compiler, a secondary program developed for this express purpose.[60] Compilers, as software themselves, are imperfect. Rarely, they contain bugs that cause a mistranslation from the high-level language to the target language, further obscuring the link between programmer intent and program.

In his famous 1984 paper, UNIX co-author Ken Thompson described how to build a malicious compiler that secretly inserts backdoors into low-level code.[61] Solidity compilers, with their relative closeness to blockchain based assets, make a similarly attractive target for attacks.[62] Yet, there is more to worry about than maliciously designed software. The Solidity Foundation, the corporate entity that oversees development of the protocol, maintains a list of security bugs that have historically affected their compiler.[63] For example, a "high severity" bug from an early version of the compiler permitted an attacker (under specific circumstances) to overwrite a value she did not control, by manipulating the value of one she did.[64] The recommended remediation was to "[d]eactivate, remove funds from, or upgrade already deployed contracts."[65]

However, if all goes well, the output from the compiling process is bytecode, a low-level representation of the high-level program that the computer can execute more directly. To provide some concrete detail, we now turn to the blockchain platforms of interest.

B. An Introduction to the Blockchain Platform

Blockchains are already the subject of a large legal literature. In broad strokes, they are platforms for distributed data

---

[60] There are two primary ways programs are executed: directly, or through an interpreter. Directly executed programs undergo multiple translation (compilation) steps taking the program from humane-readable source code, to less-human readable assembly code, to machine readable instructions. See https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Compiled_Programs. Interpreted programs generally undergo a preliminary form of compilation but are not themselves converted into machine instructions. Rather, an interpreter executes the preliminary form by a set of rules acting on a virtual machine. See https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Interpreted_Programs.

[61] Ken Thompson, *Reflections on Trusting Trust, Turing Award Lecture*, 27 COMMUNICATIONS OF THE ACM 761 (1984).

[62] The reference compiler distributed via a single official channel is a centralized point of failure for the Ethereum transaction script ecosystem.

[63] *List of Known Bugs*, SOLIDITY (last visited on Nov. 9, 2019) https://solidity.readthedocs.io/en/v0.5.12/bugs.html.

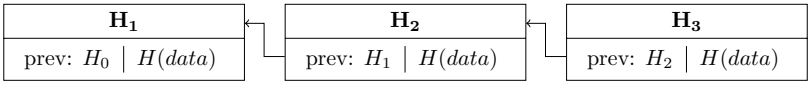
[64] This was possible with "contracts containing two or more contiguous state variables where the sum of their sizes is less than 256 bits and the first state variable is not a signed integer and not of bytesNN type." https://blog.ethereum.org/2016/11/01/security-alert-solidity-variables-can-overwritten-storage/

[65] *Id*.

processing that create incentives for users to agree on, and store, outcomes of computation.[66] A blockchain generally consists of two components: storage structures that track changes in the system and algorithms that ensure consistency of data across storage locations.

A blockchain identifies data records by hashes: the output of an algorithm called a cryptographic hash function.[67] This easy-to-compute equation takes potentially voluminous data as input and produces a short, fixed-length, output—the hash. Critically, there are no known methods to easily perform the reverse computation.[68] It is similarly hard to find two different sets of data that when fed into the function both produce the same hash.[69] This creates a tie between the input data and the hash. The hash acts as the "name" of the block, uniquely identifying it by its contents.

Data is organized into blocks, with each block linking to a set of stored data. Each block contains a hash (below in bold) corresponding to the storage records linked to the block and the hash of the previous block (shown as "prev").



A block's hash is computed by feeding the contents of the block into the hash function. The hash therefore is strongly tied to the data within the block, and also ties the block to those that come before. This property ensures an ordering to the blocks, creating the chain aspect of a blockchain. Updates (proposed or accepted) to a blockchain are generally contained within "transactions", small sets of machine instructions or transactional records that comprise the data within blocks. A block is associated with transaction data through an additional hash stored within the block, in our figure this hash is depicted as H(data).[70]

---

[66] *See generally* DE FILIPPI & WRIGHT, SUPRA note 32, at 33-49; Theophanis Stratopoulos & Jesús Calderón, *Introduction to Blockchain* (2019), https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3395619; Nat'l. Inst. Of Standards & Tech., *Blockchain Technology Overview*, IR8202 (2018), https://doi.org/10.6028/NIST.IR.8202.

[67] Nat'l. Inst. Of Standards & Tech., id., at 7-13; Stratopoulos & Calderón, *id*., at 34-40.

[68] The reader should take care to note that cryptographic hash functions used for systems such as blockchains bear additional security properties. *See* JONATHAN KATZ AND YEHUDA LINDELL, INTRODUCTION TO MODERN CRYPTOGRAPHY 128-130

[69] Even with the considerable computation power of the Bitcoin blockchain, it would take around 720 times the age of the universe to find a collision for its hash function. *See* https://hackernoon.com/a-physicists-journey-into-cracking-bitcoin-4631e57158cc

[70] The hash over the data is computed with the aid of an additional structure known as a Merkel Tree. The properties of a Merkel Tree allow one to easily notice if the contents of a particular transaction have changed without checking the entirety of the data.

Using a chain as a public and trusted record requires a mechanism to achieve consensus on the contents of the record. Participants known as validators follow a well-defined set of rules that allow them to agree both on the validity of a particular chain, and the ordering (and acceptability) of any proposed updates to the chain[71]. Participants connect to one another over the internet, forming a subnetwork within the larger network. Participants send and receive messages in a set format to indicate their responses to proposed changes to the blockchain.

The blockchain records each update that has ever happened to the data it stores. By viewing the record of changes, a viewer can access the historical state of the chain. The contents of a blockchain are fixed only so long as the participants agree about what constitutes the set of previous transactions.[72] Moreover, while the record of past transactions may be unchanged, a future transaction may modify the ledger to make the most recent contents identical in substance to the contents at a prior time (differing records of the past notwithstanding).

Protocols commonly feature *validators* who vie for the scare opportunity to submit the next block to the network.[73] Those that succeed at adding a block claim a reward.[74] A cap on the amount of data that can be stored in a single block, in combination with the restricted opportunities to add blocks limits the number of transactions that can be processed in a unit of time.[75]

An individual wishing to transact via the network submits their proposed transaction to the peer-to-peer network and indicates how much they are willing to pay (in cryptocurrency) to have their transaction processed.[76] Validators will process such user-submitted transactions, claiming associated processing fees. The size of the fee is inversely correlated with how long the network will take to process the transaction, as a larger fee more strongly incentivizes validators to include that transaction within the next block.

---

[71] These rules are the protocol governing the system.

[72] Forks serve as an "ever-present escape valve" from majority consensuses with which a minority of blockchain participants disagree. Haque *et al, supra* note 50, at 164; *cf.* Grimmelmann, *supra* note 32, at 16 ("A user community can always collectively change or ignore [consensus protocols].").

[73] In such protocols, a subspecies of validators, miners, are also given the opportunity to mint a new unit of the corresponding cryptocurrency, updating the ledger to grant them ownership over the new coin. Haque et al, *id.*, at 149.

[74] Opportunities to add blocks are allocated according to the consensus scheme, the two most popular of these being Proof-of-Work, which probabilistically allocates opportunities based on amount of computational effort spent, and Proof-of-Stake which probabilistically allocates opportunities in proportion to a miner's staked cryptoasset.

[75] Dubbed the scaling problem, maximizing transaction throughput is a highly active area of research. Proposed solutions include moving certain transactions off-chain, and other novel protocol designs.

[76] In some systems, portions of the transaction fee may be optional, but this may result in the network failing to ever process a transaction.

Blockchain ledger entries are associated with identifiers known as *addresses*, which determine who controls particular ledger entries.[77] Digital keys that grant control over assets (or data) stored in a given set of entries are stored in "wallets" which are files or programs containing these keys. Colloquially, some equate control with ownership, which is why you'll hear about a wallet's "owners." Of course, access and the ability to modify doesn't necessarily mean ownership, unless that apartment key you gave your dog walker was more significant than you thought. At its core, a cryptoasset is nothing more than an entry in the ledger, secured with a set of fancy tools that mean only those people who know the access digits can change its characteristics.

Finally, public blockchains are typically designed so that validators or other users can unilaterally join and leave the network. *Permissioned* blockchains take a fundamentally different approach, admitting only preapproved validators to the consensus forming process. While potentially useful for a consortium of known parties, or for use within an individual business, permissioned blockchains rely on users' trust in the list validators, preapproved by the entity operating the blockchain.[78] This limits their use in the low-trust marketplace for which transactional scripts are most often touted.[79] We therefore focus our analysis on the permissionless ecosystem.[80]

---

[77] These are normally derived from a users' cryptographic keys or the result of a hash function applied to relevant data.

[78] *See* Sklaroff, *supra* note 17, at 276-77 fn.50 (noting that "the advantages of [permissioned] blockchains exist in tandem with reliance on offline identity" and that as a result, "participants on permissioned blockchains are typically bound by off-chain, real-world agreements") (internal quotation omitted).

[79] Most commenters agree with Ian Kane, COO of TERNIO, in thinking that "permissioned blockchains have their place specifically in an enterprise environment." Shehryar Hasan, *Private Blockchains Are Bullshit, Expert Says*, BLOCKPUBLISHER (Jan. 25, 2019). See also Justin O'Connell, *What Are the Use Cases for Private Blockchains? The Experts Weigh In*, BITCOIN MAG. (June 20, 2016) (finding the value of private blockchains in their ability to "provide interesting opportunities for businesses to leverage [their] trustless and transparent foundation for internal and business-to-business use cases"). *Cf.* THE EU BLOCKCHAIN OBSERVATORY & FORUM, BLOCKCHAIN AND THE GDPR 16 (Oct. 16, 2018), https://www.eublockchainforum.eu/sites/default/files/reports/20181016_report_gdpr.pdf (suggesting that permissioned blockchains might need permissionless blockchains in order to be globally interoperable).

[80] The computer science literature explores a variety of scenarios in which network participants have greater or lesser trust in other participants. These works occupy the space between assuming an overwhelming majority of participants are honest, and assuming preexisting trusted relationships with other network participants (permissioned models). Such constitute a middle ground between permissioned and permissionless approaches to distributed computation, and generally represent a trade-off between scalability and trust. While laudable efforts reduce the level of trust that it is necessary to sacrifice for substantial gains in scalability, these subtleties are not the focus of our work. *See Arbitrum: Scalable, private smart contracts*, Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg and Edward W. Felten for a technique for scaling blockchain computations and for a discussion on other approaches. USENIX 2018.

C. Ethereum and Scripting

Bitcoin is an awkward commercial platform, mostly useful for recording and facilitating the flow of bitcoin transactions.[81] Realizing the utility of performing more complex operations on a blockchain, Vitalik Buterin proposed and developed Ethereum, a blockchain based computing platform, with an associated cryptocurrency, *ether*. The protocol's explicit goal was to permit enhanced scripting—more complicated logical operations than recording ownership—on a blockchain. Ethereum uses the *Ethereum Virtual Machine* (EVM)—a software system with predefined rules and operations, which you can think of as a simulated computer.[82]

The Ethereum virtual machine enables programs to store data on the Ethereum blockchain and defined how the original data could be modified.[83] By the nature of the blockchain, all data is public and therefore replicable; the EVM, however, allows developers to restrict how the data may be modified. Storage and control, in turn, created the platform for transactional scripts. Scripts are programs operating on Ethereum which when executed (or "called") affect the ledger itself.

The instruction set provided by Ethereum is as powerful and expressive as any other programming language. If limits are not imposed on transactional script running time and resource requirements, a malicious actor could force validators to perform never ending computations, halting all useful work on the chain. Ethereum therefore imposes what we term a *complexity tax*: transaction fees proportional to the computation required by a transaction. This fee is known as gas and is paid in fractional

---

[81] While Bitcoin can be scripted to perform sophisticated tasks, doing so is challenging and coders wishing to do so must work against the limited functionality of the platform.

[82] Kolber, *supra* note 24, at 208. Solidity, the most common language used to program Ethereum Transactional scripts is Turing complete, meaning it can in theory perform any program. This is limited only by the restrictions the protocol places on complexity to mitigate denial of service attacks. *See* Niharika Singh, *Turing Completeness and the Ethereum Blockchain,* HACKER NOON (Feb. 16, 2019), https://hackernoon.com/turing-completeness-and-the-ethereum-blockchain-c5a93b865c1a. The EVM draws its inspiration from common models of computer architecture with modifications that ensure the integrity of scripts' code. While this architecture is foreign to the computers on which the software runs, it is simulated by way of the virtual machine. See https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369

[83] For a more complete discussion on the limits of the Bitcoin model, *see* ConsenSys, *Thoughts on UTXO by Vitalik Buterin*, MEDIUM (Mar. 9, 2016), https://medium.com/@ConsenSys/thoughts-on-utxo-by-vitalik-buterin-2bb782c67e53. As agents able to manipulate the records stored on the blockchain, transactional scripts are able to transfer value contingent on those records. This is the true source of their utility over other types of programs that interact with distributed databases: the tight integration of an asset storage mechanism (cryptocurrencies and cryptoassets) with a programmatic way to transfer ownership (transactional scripts). The coupling of the two ensures that the value of the asset is conditional on playing by the rules of the game – which in turn provides certain assurances that the contracts will be executed.

amounts of ether.[84] The Ethereum protocol specifies a hard limit on how much gas may be consumed by a single transaction.[85] Thus, the *amount* of gas paid per transaction is determined by the Ethereum protocol, while the *exchange rate* is determined by the user. The user must also pre-pay gas that in their estimation will sufficiently cover costs. If this pre-payment is too low, and there is insufficient gas to completely execute the script, the script will terminate without a refund.[86]

There are 70 different operations understood by the EVM, each of which is associated with a cost, based on the amount of time and energy it takes a validator to execute the operation. Here are only a few, with their associated costs.

Operation Name	Max Cost (Gas)	Cost ($)/1 million operations[87]	Effect
ADD	3	~$30	Adds two numbers together
MUL	5	~$50	Multiplies two numbers
SSLOAD	200	~$2000	Load a single number from permanent storage[88]
SSTORE	20000	~$200,000	Store a single number into permanent storage

The most expensive operation, SSTORE, updates data that constitute the Ethereum ledger's state. As it adds to the long-term cost of storing the ledger, the protocol imposes a substantial up-front fee.

Data storage on a public ledger forms a key component of many proposed blockchain use-cases,[89] but is financially costly in

---

[84] For a detailed explanation of the internal mechanisms of Ethereum, *see* Preethi Kasireddy, *How Does Etherum Work, Anyway?,* MEDIUM (Sep. 27, 2017), https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369.

[85] As of writing, this was set at approximately 8 million gas (with an upcoming change to 10 million gas), equivalent to ~$2 USD at a gas price of 1 Gwei (billionths of an ETH).

[86] Existing literature describes the complexity tax in general terms without elaborating on its exact costs. *See, e.g.* Thibault Schrepel*, Is Blockchain the Death of Antitrust Law? The Blockchain Antitrust Paradox,* 3 GEO. TECH. L. REV. 281, 292 (noting that Ethereum rewards successful miners with transaction fees); Sklaroff, *supra* note 17, at 293 n.139 (describing the "supply and demand dynamic" created and its policing effects on "buggy or infinitely recursive code"), id. at 295 n.143 ("[T]his solution would be prohibitively expensive from a transaction fee perspective").

[87] These figures are calculated at the default gas price of $\mathbf{3 \times 10^{-8}}$ gas/ETH and an exchange rate of $295 USD/ETH with data sourced from https://hackernoon.com/ether-purchase-power-df40a38c5a2f rounded up to one significant figure

[88] The maximum value on which the EVM operates on in a single operation is **2^256**. Outside of technical reasons for this choice, this limit prevents coders from storing all their data represented as one huge number in an attempt to pay lower gas costs.

[89] Even for systems that purport to store the bulk of data off-chain, the cost of storing references to the off-chain data necessitates a cost-benefit approach to assessing the

comparison to general purpose storage (which is available from commercial providers on a yearly basis at approximately one-ten-millionth of the cost of Ethereum based storage).[90] Similarly, the processor in a typical laptop performs on the order of billions of operations per second, which would equate to hundreds of dollars' worth of computations on Ethereum.[91] Gas costs don't stem from the regular unit cost of storing a chunk of data, or performing a computation.[92] They are artifacts of the replicated work and storage used to maintain and validate consensus.[93]

This complexity tax has limited, but real implications. True: storage and complex programming aren't what blockchains are for, and such special purpose tools shouldn't be evaluated in comparison with traditional computers. While this objection has merit, it's important to recognize that for some proposed uses of transactional scripts—such as encoding semantic frameworks like discretion, good faith, and best efforts—blockchain solutions incur significant costs.[94]

The result of such costs is that it is practically impossible to run some kinds of scripts (as we've defined them) in public. Many real-world computational tasks (searching and sorting large amounts of data, machine learning, optimization) require non-trivial amounts of computational power and/or data storage. For example, though it would be potentially useful to write a script that used an algorithm to determine if a worker had used her best efforts, and then pay her for her time, that kind of computation is not practical other than by delegating the computation to an agent "off-chain." Current scripts thus generally contain only the

---

viability of a blockchain based solution. *See* Medicalchain Whitepaper (records themselves are stored off chain, but each update to a record necessitates storing a new value on the blockchain), https://medicalchain.com/Medicalchain-Whitepaper-EN.pdf; Arman Jabbari and Phillip Kaminsky, *Blockchain and Supply Chain Management*, http://www.mhi.org/downloads/learning/cicmhe/blockchain-and-supply-chain-management.pdf (envisioning large-scale supply-chain data storage on the blockchain.)

[90] https://aws.amazon.com/s3/pricing/

[91] Some sites modern processors at the tens of billions of floating-point operations per second, that on the scale of our illustrations, are comparable with the basic arithmetic operations of the EVM. *See, e.g.,* https://asteroidsathome.net/boinc/cpu_list.php.

[92] Noted blockchain researcher Emin Gün Sirer correctly notes that costs of storing the blockchain itself are negligible on a per node basis (*See* https://twitter.com/el33th4xor/status/1200477778463907841). Our concern is how this cost is magnified by current consensus protocols.

[93] These particular costs are distinct from the computational cost incurred by Proof-of-Work style consensus algorithms, that establish consensus in the first instance. Rather, the need to check the validity of the output from computations and maintain copies of the output (to allow failure recovery), impose a cost that is paid ex ante by a tax on expensive computations.

[94] While Sklaroff, *supra* note 17, notes the potential high ex ante cost of developing contracts flexible enough to incorporate legal frameworks, we here quantify those costs, focus on the ongoing taxes imposed by the blockchain paradigm, and expands on coding realities that further drive development costs

simplest of if-then type logic.[95] A useful analysis of the legal significance of our scripts requires a clear-eyed evaluation as to the likely future uses and limitations of the form.

There *is* a robust technical literature exploring mechanisms to avoid the complexity tax. But most proposals to mitigate costs trade-off between the security and transparency of the on-chain model, and the efficiency attainable under frameworks that assume more on the part of other protocol participants.[96] While there are some improvements that can be made to efficiently scale blockchains without compromising the promise of truly trustless exchange,[97] the most effective solutions will inevitably delegate the bulk of computation and storage to smaller subsets of the network or even to non-participants off chain[98]. While there are known on-chain techniques to reduce the cost of most computations, finding a way to minimize blockchain data storage costs while ensuring the data is accessible on demand is an open problem.[99] Firms considering off-chain solutions must therefore ask whether the game is worth the candle.

That is, public blockchains have important virtues: primarily, they enable trustless exchange between pseudo-anonymous counterparties. Alternatives may not have those virtues, and firms may not choose intermediate blockchain solutions for exchange.

D. *Coding on Ethereum*

We now return to our coders' work and focus on how they write a transactional script. Suppose a team of coders have authored high-level code and run that high-level code through the

---

[95] The quarrelsome reader may note that all *imperative paradigm* programming adheres to an if-then paradigm. We merely note the paucity of sophistication in scripts currently deployed. For a brief discussion on different programming paradigms see https://medium.com/@jingchenjc2019/a-brief-survey-of-programming-paradigms-207543a84e2b

[96] *See* Gudeon Moreno-Sanchez, Roos, McCorry, Gervais, *SoK: Off The Chain Transactions*, IACR ePrint 2019/360 for an analysis of various designs facilitating off-chain computation by a smaller number of nodes, reducing the cost imposed by large scale replication of work and Kyle Croman, Christian Decker, Eyal Adem, Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, Roger Wattenhofer, *On Scaling Decentralized Blockchains.*

[97] One proposal that succeeds in maintaining a trustless network while improving substantially on scaling is the Avalanche Protocol, which proposes an alternative to proof-of-work. *See* Kyle Croman, Christian Decker, Eyal Adem, Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, Roger Wattenhofer, *Scalable and Probabilistic Leaderless BFT Consensus through Metastability*, 1905 ArXiv 08936 (2019). However even protocols that minimize computational costs (Avalanche and Arbitrum, *supra* note 79) still don't realize the promise of effectively free transactional scripts.

[98] Importantly for our analysis of buggy scripts in Part III, even where off-chain scaling techniques are adopted, the code remains publicly accessible.

[99] *Supra* note 96.

compiler, producing bytecode that can now be executed on the EVM. If they are responsible, they must now test the program; only after testing will it be deployed. During development programmers will go through many, many cycles of coding, compiling and testing, finding bugs and areas that need improvement. Logically separable elements of the program will often be developed separately, by different people, and over a span of days to months. The final code therefore resembles more a pastiche of many separate thought processes rather than a distillation of discrete intent.

To make the discussion more concrete, we present the *Staged Contract* script, written in Solidity, and explain how it is processed by the virtual machine. It illustrates a simple deal: the owner of the script agrees to pay for work in increments of 1ETH

```
contract StagedContract {

  // THE JOB IS WORTH 5ETH
  int256 public transferlimit = 5;
  int256 public amounttransferred = 0;
  int256 public discount = 0;

  address payable public owner =
      0x2BcEB4B315Fd823Bd93cb9065A97C7A7d0174E93;
  address payable public recipient =
      0x1161B70D1ddc964785189ab7CFf5006cbbEfab4E;

  // ALLOW THE OWNER TO APPROVE A DISCOUNT
  function approveDiscount() public {
    if(msg.sender == owner){
      transferlimit = transferlimit - discount;
    }
  }

  // ALLOW THE RECIPIENT TO DISCOUNT THEIR WORK BY A FIXED
      AMOUNT
  function offerDiscount(int256 _amount) public {
    if(msg.sender == recipient){
      discount = _amount;
    }
  }

  function completeStage() public {
    // CHECK IF RECIPIENT IS RUNNING THIS,
    // AND THEY HAVEN'T BEEN PAID THE ENTIRE AMOUNT YET
    if(amounttransferred < transferlimit &&
    msg.sender == recipient) {
      revert();
    } else {
          // PAY THE RECIPIENT 1ETH FOR EACH STAGE OF THE
              WORK
          // THEY CAN CLAIM IT AT ANY TIME
          amounttransferred = amounttransferred + 1;
          recipient.transfer(1);
    }
  }
```

Figure 2: Transactional Script in Solidity

as the recipient completes stages of some off-chain job.[100] The recipient can claim payment for each stage without the intervention of the owner up to the total value of the contract.

StagedContract also allows the recipient to offer a discount through the discount() function which the owner can approve via the approveDiscount() function. Illustrative of the complexity tax, at time of writing, it cost $1.30 to deploy the script and $0.08 to execute the completeStage() function once.

Like most computer programs, Solidity requires the availability of fundamental computational structures:[101] basic arithmetic operations, operations to access and modify some form of memory (the "state"), a way to input/output data, and crucially, a mechanism to choose between different possible execution paths based on the state of the computer. These are the key elements implemented by the EVM.

Even this very simple script requires expertise to conceive of, implement and deploy. It requires still more expertise to recognize that it contains a serious bug. Noting that if an integer grows or shrinks beyond the limit set by the EVM (overflow and underflow respectively), the integer resets, a sneaky recipient could mount the following attack:[102]

1. The recipient offers a discount that reduces the cost of the job to nothing, which the greedy owner readily accepts.
2. The recipient offers an additional discount in advance of future work and waits until just before the owner approves it.
3. The recipient quickly engorges their discount to the maximum size permitted by the EVM.
4. The unsuspecting owner fails to notice the change and approves the oversized discount, reducing the sum available for payment so much that it underflows—resetting the amount available for payment to the *maximum* value permitted by the EVM.
5. The recipient now executes completeStage() repeatedly, emptying the script of its ether, without running into the limit.

This is but one of many ways in which the specifics of the EVM can trip up otherwise code-literate individuals

The problem is that scripts like these which are already present on the blockchain can't be easily modified. Therefore, wise

---

[100] Not depicted is a mechanism for the owner to increase the amount of payment available over time.

[101] These, along with the ability to execute loops, are also the elements that permit the EVM to support a Turing complete language. Singh, *supra* note 82.

[102] While the scenario is admittedly contrived, it serves to illustrate the difficulty of writing correct code

developers would first try it out on "test-nets," small scale blockchains replicating the behavior of their larger siblings, but without the goal of preserving immutability or value long term. Testnets are also used to trial changes to the core protocol of a blockchain, without impacting existing users. Testnet transactions are generally free, allowing developers to avoid the cost associated with deploying multiple versions of their transactional scripts.

When a developer is ready to test their transactional script, either on the real network or on the testnet, they submit a special transaction to the network. The transaction includes the bytecode of the transactional script, an amount of gas to pay for the deployment, and a wallet from which the gas will be transferred out. The transactional script is then stored on the blockchain, and its component functions and storage can be accessed.

Modern programming languages, including Solidity and the Ethereum EVM instruction set, separate segments of code that perform discrete operations into "functions," each of which require a developer using the function to provide certain inputs. Once a function finishes executing, it produces an output which is provided to either the user or, in instances where a function is executed ("called") within another function (the "caller"), the output is available for use within the caller. To call a function, the caller must also include gas sufficient to pay for the execution of the callee (and in a recursive fashion for any functions the callee might call).

This all assumes the coders have implemented each step of this process correctly. But, as the *StagedContract* example illustrates, errors can be subtle. There simply is no foolproof method to generate software that matches its initial specification.[103] Further, even as software is patched to remove old bugs many new bugs creep in, and software does not converge to a bug-free state.[104]

Penetration testing and security auditing are other important components of sophisticated software development. Specialized security engineers attempt to find and exploit security flaws and assess the quality of code as relevant to security.[105] However, security failures bedevil even the best

---

[103] *See* Wenbo Guo, Dongliang Mu, Xinyu Xing, Min Du, Dawn Song, *DEEPVSA: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis*, USENIX SECURITY SYMPOSIUM 2018.

[104] *See* Saender A. Clark, *The Software Vulnerability Ecosystem: Software Development in the Context of Adversarial Behavior,* Unpublished Dissertation University of Pennsylvania, ProQuest Dissertations Publishing, 2017. 10242540.

[105] Due to the level of specialization required, such engineers or testers are contracted from boutique firms, with fees comparable to those of high-end lawyers. This puts many well performing service firms outside the price range of the majority of transactional script

audited software packages.[106] The real test for code, particularly when designed to operate adversarial environments, comes only when it is deployed and used. It is often only when the code meets the road that developers find the bulk of bugs, improving on their code by constant iteration.[107] Even then, vulnerabilities may remain latent for long periods of time prior to discovery.[108]

These characteristics of the vulnerability life cycle pose challenges for a transactional script developer, where the opportunities for safe testing on a live blockchain are limited, and the ability to patch significantly hampered. Further the semantics of the Solidity coding language in which almost all transactional scripts are written are substantially different from traditional software development, leading to overconfident developers ignoring potential pitfalls such as reentrancy vulnerabilities that are not present in most coding environments.[109]

The need for security also imposes a second, more metaphysical "complexity tax:" the more complex a transactional script , the harder it is to preserve both the coders intention (and the intention of the person hiring the coder and so on), while not creating any insecurities; more development time must be expended to shore up the code.[110] The bigger the transactional script ship, the more effort must be expended to patch the leaks.

---

developers.
https://www.reddit.com/r/ethdev/comments/6pdgvd/how_much_does_a_smart_contract_audit_cost/

[106] OpenSSL one of the most scrutinized software packages (and which underlies much of the cryptography used to secure the internet) disclosed over 11 high-severity vulnerabilities since 2014, despite having been created prior to 2002 https://www.openssl.org/news/vulnerabilities.html

[107] Some experts estimate the density of bugs is 10-50 per KLOC (1000 lines of code), and as few as that even Microsoft with its sophisticated development practices still releases code with a bug density of ~0.5 per KLOC. The is equivalent to tens-of-thousands of bugs in a modern operating system compiled from tens-of-millions of lines of code. *See* STEVE MCCONNELL, CODE COMPLETE.

[108] *See* Clark *supra* note 104.

[109] Reentrancy vulnerabilities are a class of flaw wherein a function calls another function that is external to the given script, and that callee unexpectedly calls the original function before the original function has finished executing. *See* https://consensys.github.io/smart-contract-best-practices/known_attacks/. Programming language considerations that contribute to the difficulty of developing secure and correct Solidity code include the potential for "integer overflow" (where the size of numbers exceeds the space available to store them), a lack of support for decimal numbers and incomplete formal specification of the language. For a further description of these and others see https://www.reddit.com/r/ethdev/comments/7rdocn/what_are_the_main_security_problems_associated/

[110] *See* Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne, *Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities,* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 37, no. 6 (2010): 772-787.

E. Summary

A recent survey of script developers suggests that coders are becoming increasingly aware of the problematic real-world consequences of how Solidity interacts with development have real world consequences.[111] Developers, many of whom were working for free,[112] had ideological motives: to "create a decentralized currency that cannot be manipulated by a central authority."[113] That is, "removing power from banks and governments."[114] These were motivated, committed, blockchain proponents.

But the survey respondents noted that the security concerns we've discussed and stakes made coding difficult. "[In] most [non-blockchain projects] when a bug appears, it will be fixed and soon forgotten. But in blockchain projects some bugs can be very costly and never forgotten."[115] Similarly, some complained that erroneous ledger entries are "almost impossible" to fix. These unique blockchain problems, when coupled with the decentralized VM on which software operates, "makes it difficult to build robust software. Unreliable connections, unexpected latency, and malicious nodes create a hostile production environment."[116] There are also few production-ready tools to work through errors in scripts, particularly a "reliable and user-friendly decompiler."[117]

In sum: it's simply impossible to create perfect software the first time through, and existing tools to pre-test scripts before deployment are inadequate or extremely costly. Irreducible features of Ethereum (and other blockchains designed using the same logics) will render transactional scripts buggy. One recent study, looking only at the very simple ecosystem of scripts on Ethereum, found 100 errors per 1000 lines of code.[118] This error

---

111 Amiangshu Bosu, Anindya Iqbal, Rifat Shahriyar & Partha Chakroborty, *Understanding the Motivations, Challenges and Needs of Blockchain Software Developers: A Survey*, 24 EMPIRICAL SOFTWARE ENGINEERING 2636 (2019).

112 *Id*. at 11-14.

113 *Id*. at 15

114 *Id.*

115 *Id*. at 17.

116 *Id*. at 18.

117 *Id*. at 25.

118 Jakub J. Szczerbowski, *Place of Smart Contracts in Civil Law. A Few Comments on Form and Interpretation*, PROCEEDINGS OF THE 12TH ANNUAL INT'L. SCI. CONFERENCE NEW TRENDS 2017 (Nov. 9, 2017), https://ssrn.com/abstract=3095933; Ivica Nikolic, Aashish Kolluri, Ilya Sergey, Prateek Saxena, & Aquinas Hobor, *Finding the Greedy, Prodigal, and Suicidal Contracts at Scale* at 1, ARXIV.ORG (Mar. 14, 2018), https://arxiv.org/abs/1802.06038 (finding a 3.5% vulnerability rate across an analysis of one million scripts); Giuseppe Destefanis, et al., *Smart Contracts Vulnerabilities: A Call for Blockchain Software Engineering?,* BLOCKCHAIN ORIENTED SOFTWARE ENGINEERING (IWBOSE) 2018 International Workshop 19-25 (2018) (stating "[t]he feeling of many software engineers about such huge interest in Blockchain technologies and, in particular,

rate likely will increase as developers pursue ever-more-ambitious Ethereum projects. As bugs accrue and create real-life losses, parties will turn to tribunals and to the law for recourse. In the next section, we offer some concrete examples of this insight.

II. TRANSACTIONAL SCRIPTS IN THE REAL WORLD

Now that we have in hand a better understanding of what writing a transactional script entails—and where error might creep into that process—let's consider three typical use cases of scripts in the current blockchain ecosystem.

A. Tokens

A basic use of a transactional script is to change a blockchain record to debit a cryptocurrency from a single address' entry and credit the entries of other addresses. However, the flexibility of the EVM allows for more sophisticated types of trades, both of ether (the Ethereum-defined base currency) and script-defined cryptoassets.

ERC-20 is the Ethereum technical standard that provides a template for creating a fungible, tradeable asset, known as a token.[119] Such tokens are the most common cryptoasset. Token balances are not stored by the owner of the token. Instead, the transactional script that created the asset an internal ledger of addresses and their corresponding token balances. Below we provide some of the code that creates such tokens.

In the following script excerpt the variable *balances* serves as the scripts' internal ledger of account balances. While any human can manually inspect the ledger, an Ethereum script can only access the balances via the balanceOf() which outputs the balance for a given address. This includes the token script itself, which uses balanceOf within the transfer function to check if there is sufficient balance available to debit, before subtracting that amount from one address and crediting to another. These transfer and balance functions are depicted in the graphic below.

By limiting direct access to the balances, the script functions like a metaphorical sealed vault of a novel commodity, with a corresponding public ledger determining ownership over the

---

on the many software projects rapidly born and quickly developed around the various Blockchain implementations or application, is that of unruled and hurried software development.")

[119] The standard requires a compliant script to include a method to determine the total number of such tokens in circulation, the number of tokens owned by a given address, and functions that facilitate the transfer of tokens. *See* https://theethereum.wiki/w/index.php/ERC20_Token_Standard

contents to different parties. The vault allows users to query and alter the ledger only by the mechanisms controlled by a series of

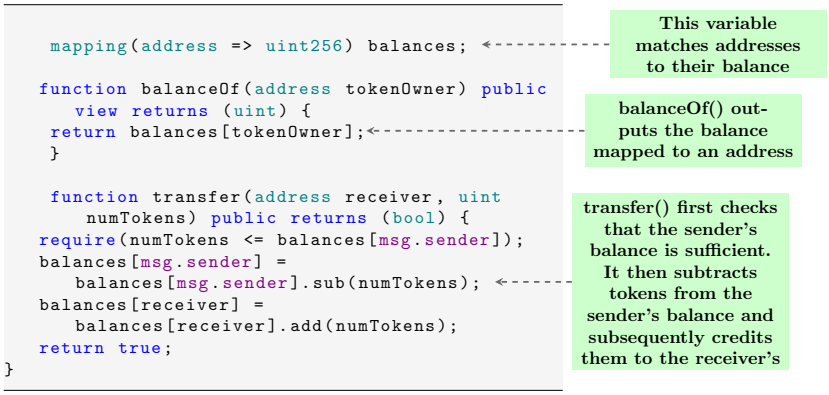


Figure 3: Token Script

buttons on the vault's exterior. The buttons correspond to the functions that a given script makes available, and illustrate how even with a transparent vault, control over and ownership of the commodity ultimately lies in the mechanism underlying the buttons.

Likewise, exchanges of a token can only be effectuated through the interface provided by its parent script. In the above example this is captured by the "transfer" function.

While a secondary script may layer on supplementary terms of an exchange, the actual transfer of ownership is mediated through the script that maintains the balances variable for that asset—no matter the rules or rituals one constructs around the operation of the aforementioned vault, the ledger remains under the sole control of the button mechanisms. Any flaw in the mechanism is therefore propagated to all users of the asset.

Consider what would happen were the coder to forget the check ensuring that adequate funds were available. For one, a user could transfer more tokens than present in their balance, allowing them to accrue more tokens, i.e., cryptoassets, than they would otherwise be entitled. Additionally, any other script using the asset would also be affected as all token transfers are mediated by this contract that maintains and controls the balance variable.

Of course, tokens are not merely technological artifacts. They take their value from social consensus: their holders must think they will eventually provide some utility (even if merely being trading instruments). To obtain that consensus, tokens are typically described and marketed with natural language text, written by people who may, or may not, have coded the tokens'

scripts. In previous work, we examined the ERC-20 tokens created as a part of initial coin offerings in 2017.[120]

Such offerings, loosely modeled on initial public offerings, typically involve the exchange of bitcoin or another form or cryptocurrency for a set of rights embodied in a transactional script. For example, an organization called Kik raised $98M in 2017 by offering for sale some of 10 trillion "Kin" tokens it had created. According to Kik's White Paper, thirty percent of the total sale proceeds were earmarked for "startup resources, technology, and consideration in exchange for a covenant to integrate with the Kin cryptocurrency and brand."[121] Kik could, and did, embed these promises in a transactional script. We found that on a variety of measures, Kik's marketing documents and code matched exactly.[122]

Tokens thus pose at least two sorts of problems for jurists. First, what if the code itself is somehow flawed, meaning that their buyers receive something different than they expected? Second, what if the code fails to match the natural language promises that purport to describe it?[123]

B. *Exchanges*

Sometime on November 15, 2018, someone placed a buy-offer for a token called "Free Coin" on the TokenStore decentralized cryptocurrency exchange at ten times the prevailing market rate. This created a substantial arbitrage opportunity for an enterprising trader who subsequently purchased Free Coin at the market rate and resold it at the inflated rate, realizing a profit of 0.79ETH or $267 USD, while paying a complexity fee of $5.00.[124] Wishing to ensure that the entire sequence of trades was completed, the second trader batched the transactions using a script guaranteed to complete both the buy and sell trades, or neither.

This series of events is normal on digital marketplaces that trade cryptocurrency. Such marketplaces today are a major source of liquidity for cryptoassets, and consequently the most practically important public face of the transactional script

---

[120] Cohney *et al*., *supra* note 2.
[121] Cohney *et al*., *supra* note 2, at 628-29.
[122] *Id*. at 673. That is not to say Kik is in the clear: it remains enmeshed in a fight with the SEC about whether its tokens were securities.
[123] *See* Complaint, Sec. & Exch. Comm'n v. REcoin Group Foundation, LLC, No. 17-CV-5725 (RJD) (RER) (E.D.N.Y. Sept. 29, 2017) (alleging securities liability when white paper made representations about charitable giving for which "there is not program code").
[124] See Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li and Xueyuan Zhao , Iddo Bentov, Lorenz Breidenbach, Ari Juels, *Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges*. The fee was 113,265 gas at a price of 134 Gwei, or about $5.

commercial ecosystem.[125] Indeed, they may be the only fora where it is obvious that transactional scripts are pragmatically important mechanisms of exchange. Cryptocurrency exchanges take multiple approaches to custody, settlement, and order matching, which we now explore.

Unlike the "Free Coin" trade, the majority of cryptocurrency trades currently occur on *centralized* exchanges. Users send either fiat currency or cryptocurrencies to an account controlled by the exchange, and in return, the exchange promises to (and normally does) promptly transfer an equivalent amount of a requested asset. Such trades almost always occur off-chain and are settled using the exchange's internal ledgers.[126] Exchanges do generally offer custodial "wallets" that store the keys to a users' cryptoassets for easy trading.

These centralized exchanges could, but in most cases do not, use transactional scripts (as we define the term). Rather, they act as a market maker, matching orders where the seller's price is lower than a buyer's price and often facilitate trades between two parties for whom it is the custodian. There is thus good reason to assume that trades between such parties are governed by ordinary contracts law. Later, we'll discuss one such example, which resulted in the *Quoine* case.

The trend, however, is towards decentralized models for cryptocurrency exchanges.[127] Collapses of centralized exchanges have left users without access to balances stored on the exchange.[128] Centralized exchanges have also suffered for lack of liquidity across rarer asset types as exchanges compete for order flow.[129] These flaws, combined with the blockchain community's ideological opposition to centralization, provided fertile ground for the development of decentralized exchanges[130] or DEXes.

TokenStore, a DEX, did not automatically match trades in their order book. Rather, sellers of assets would post an order and buyers would digitally sign their intent to match the order, forwarding the transaction to the DEX's on-chain contract which

---

[125] *See generally* Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Toneli, and Michel Marchesi, *A Massive Analysis of Ethereum Smart Contracts Empirical Study and Code Metrics*, IEEE Access, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8733785 (most TS are exchanges)

[126] Note these are not ledgers in the blockchain chain, but merely the exchanges' own internal record keeping mechanism.

[127] While trades on decentralized exchanges are still of comparatively low volume, the bulk of new exchanges are adopting decentralized models.

[128] Mt. Gox., notable for its 2014 collapse, handled as much as 70% of all Bitcoin transactions prior to the event. *See* https://techcrunch.com/2019/02/06/the-plot-to-revive-mt-gox-and-repay-victims-bitcoin/

[129] *See* https://media.consensys.net/state-of-decentralized-exchanges-2018-276dad340c79

[130] Despite the allure, for a long time decentralized exchange was somewhat of a misnomer as the earlier generations still relied on a single contract for market making and settlement.

provided for a 0.3% payment allotted to TokenStore. This is a common setup for DEX systems today.[131] The service provided in this instance can be viewed as two separable components:[132] a listing service for open orders, and a platform to automatically consummate signed trades. Each component is mediated by a different piece of software. The order book is maintained by a centralized database, and interaction with it is by a website and its accompanying interfaces, both controlled by TokenStore.

The consummation component is managed by a minimal transactional script. To perform a trade, users first place cryptoassets in the custody of the transactional script using a deposit function. A trader wishing to match an order then uses the web interface to generate a signed transaction and submits it to the trade() function in the transactional script. The script performs a number of checks to ensure that the trade is valid and then updates the balances of both users.

In the case of our poor trader, once the trade was entered into the order book, any counterparty could force its execution. TokenStore appears to lack either a whitepaper or any substantive formalization in natural language of their system architecture. Its website contains no link to a terms and conditions, and even their medium blog is sparely populated with only eight posts, with fully half being merely launch announcements.

The natural language content surrounding TokenStore is limited to a handful of tweets, blog posts, commit messages, and code comments, which yield only modest insight into the purported offering. A medium post entitled "Advantages of token.store ETH — Summarized" touts the security of the platform claiming "token.store doesn't hold any of your funds; the trader deposits his funds into a smart contract…This makes the experience safe and secure by its very nature" and directs users to a link to "check the code". On Twitter, the project bragged that "funds at http://token.store ETH and EOS are held in smart contracts: only users who hold the private key to the wallet which deposited them can withdraw them."[133]

---

[131] An explanation of similar architectures and the costs they impose is provided by Iddo Bentov, Lorenz Breidenbach, Phil Daian, Ari Juels, Yunqi Li, and Xueyuan Zhao, *The Cost of Decentralization in 0x and EtherDelta*, http://hackingdistributed.com/2017/08/13/cost-of-decent/

[132] This separation is justified by noting that a DEX could build a platform compatible with a rival's signed transactions, using its own contract to consummate trades. TokenStore itself appears to have added support for transactions originating from "0x", another popular DEX. See https://github.com/tokenstore/contract/pull/11

[133] https://twitter.com/TokenDotStore/status/1142470315777363968. Of note, the script is upgradable via an opt-in process, meaning that while TokenStore currently has no way to access user assets, a future version of the contract certainly could. Though TokenStore

Of direct relevance to trading error, the code controlling trade execution is preceded by the following code comment:

```
// Note: Order creation happens off-chain but the orders are
signed by creators,
// we validate the contents and the creator address in the logic
below
```

The notion that TokenStore "validates" the contents of an order prior to fulfillment leaves open the question of what validation a non-code-reading user ought to expect. We return to these issues in Section III.

C. Oracles

In their default setting, transactional scripts are unable to interact with events occurring or data outside of the blockchain. Consider a transactional script that pays a shipper so long as the temperature within the shipping container stays below a certain threshold. While the logic is simple (if the temperature never went above X, pay the contract price), the quandary for the coder is how to ensure that the transactional script knows what the temperature inside the container is. This problem is solved using an "oracle," a computerized agent that periodically submits the needed external data to the blockchain to be used within contracts.[134]

An oracle normally consists of two parts: a script and a way to populate the script's data store.[135] The figure below excerpts from the oracle for a currency exchange called Synthetix, showing a portion of the function that incorporates updates to currency exchange rates onto the script's storage.

As shown above, to execute the update function, the program must provide as input the data it wishes to incorporate. Notably, with each update a gas fee must be paid to store the new data. Oracles are thus not immune from the challenges associated with the complexity tax.

---

has delisted a number of tokens from its order book and user interface, individuals who have delisted tokens stored in the transactional script can still access them by manually submitting a withdraw transaction to the blockchain. See https://twitter.com/TokenDotStore/status/1063126115290615808

[134] Incorporating external data directly onto the chain undermines the consensus mechanism as there is no principled way for network participants who do not have direct access to the external data to validate its correctness. *See* Alexander Egberts, *The Oracle Problem - An Analysis of how Blockchain Oracles Undermine the Advantages of Decentralized Ledger Systems*, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3382343

[135] Oracles can exist in software or hardware, with the latter purportedly offering some security guarantees stemming from the increased difficulty of compromising hardware

The data from oracles is typically provided by a third-party. This highlights a significant trade-off: oracles require trust in the data source (and further in the soundness of the oracle script).[136] For some, the intermediation and trust reintroduced by oracles highlights the "oracle paradox": the more relevant the oracle data, the less one should be willing to trust the provider.[137] There are a

```
function internalUpdateRates(bytes4[]
    currencyKeys, uint[] newRates, uint
    timeSent) internal returns(bool) {
require(currencyKeys.length ==
    newRates.length, "Currency key array
    length must match rates array length.");
require(timeSent < (now +
    ORACLE_FUTURE_LIMIT), "Time is too far
    into the future");

// Loop through each key and perform update.
for (uint i = 0; i < currencyKeys.length;
    i++) {
    // Should not set any rate to zero ever,
        as no asset will ever be
    // truely worthless and still valid. In
        this scenario, we should
    // delete the rate and remove it from
        the system.
    require(newRates[i] != 0, "Zero is not a
        valid rate, please call delegateRate
        instead.");
    require(currencyKeys[i] != "sUSD", "Rate
        of sUSD cannot be updated, it's
        always UNIT.");

    // We should only update the rate if
        it's at least the same age as the
        last rate we've got.
    if (timeSent <
        lastRateUpdateTimes[currencyKeys[i]])
        {
        continue;
    }

    newRates[i] =
        rateOrInverted(currencyKeys[i],
        newRates[i]);

    // Ok, go ahead with the update.
    rates[currencyKeys[i]] = newRates[i];
    lastRateUpdateTimes[currencyKeys[i]] =
        timeSent;
}

emit RatesUpdated(currencyKeys, newRates);

// Now update our XDR rate.
updateXDRRate(timeSent);
```

The function takes as input a set of exchange rates and currency pairs, along with the current time

The function performs the following process for each currency pair:

1. **Check if the last update was far enough in the past to require a new update**
2. **Invert the rate if required**
3. **Update the script with the new rate and reset the update timer**

Figure 4: Oracle Script from Synthetix

[136] Federated oracles attempt to tackle this trust problem by distributing it: they require that a small number of parties independently provide the same information. The transactional script validates that the information it received is consistent across each parties' submission, before it permits the data to be used. Such an approach is no guarantee that a series of bugs won't cause the overall failure of the system.

[137] See https://hackernoon.com/the-middleman-of-trust-the-oracle-paradox-and-five-protocols-that-can-bring-external-data-into-the-df39b63e92ae

number of ingenious protocol designs that achieve to tweak these trade-offs.[138]

But oracles break down despite the best intentions of all involved. For example, on June 25, 2019, one of the commercial data feeds from which Sythnetix receives its USD/KRW (Korean Won) exchange rate "began to intermittently report a price 1000x higher than the current rate."[139] An earlier outage had taken one of the other two data sources offline, and the code to disregard outliers averaged the remaining feeds was also buggy. For a script operating a market for synthetic cryptoassets tied to real world assets, a mismatch between the synthetic asset and the underlying asset was the worst-case scenario.

A bot was able to detect the mismatched exchange rate and took advantage of the arbitrage opportunity, acquiring tokens with a market value of $1b with a mere $1000 investment.[140] Fortunately for Synthetix, the bot owner reversed the trades in return for a bug bounty payout. The chain indicates the reversal was achieved through a subsequent trade at an exchange rate between an sETH (tokenized ETH) and sKRW (tokenized KRW) many orders of magnitude below the actual rate.[141] This suggests cooperation between Synthetix (which would have been able to manually adjust the exchange rate) and the bot owner who willingly converted their sETH below market rate. While the on-chain transactions suggest that the bot owner lost their investment (along with their profits) in the reverse transaction, this was likely remediated through a side payout.

We previously showed a script by which Synthetix incorporated various exchange rates. As with our previous examples, that script was accompanied by natural language text, which provided the opportunity for gaps between intention and outcome. The top of the file included the following description of its function:

---

[138] Decentralized oracle protocols (such as Chainlink) provide networks of individual oracles along with incentive structures that promote trust in the network. An alternative solution operated by Augur maintains a prediction market which syncs to off-chain events by maintaining a fee pool paid out to participants that report on the state of the off-chain world. Amusingly a participant can disrupt the market by placing a self-referential bet on Augur itself.

[139] *See Synthetix Response to Oracle Incident*, https://blog.synthetix.io/response-to-oracle-incident/

[140] The exchange of ETH used to take advantage of the arbitrage opportunity is accessible at https://etherscan.io/tx/0x6f6ee43ee07013503df786532493a3c405465f91e3ce8bb4ba8717a715db1caa

[141] This was deduced by following the chain of transactions ending at https://etherscan.io/tx/0xc3fc19c63e1090eb624212bad71a27cd3dc7afcd0cf9063d24bfc47b5d036ae2 with an exchange of 37m sETH for 362 sKRW.

> [This is a] contract that any other contract in the Synthetix system can query for the current market value of various assets, including crypto assets as well as various fiat assets.
>
> This contract assumes that rate updates will completely update all rates to their current values. If a rate shock happens on a single asset, the oracle will still push updated rates for all other assets.

Alongside extensive code commentary, Synthetix makes representations as to the functionality of its platform in the README accompanying the code, in the help section of its webpage, and in its marketing materials.[142] None of those commentary documents references the possibility of an error at the data source. The README file, however, notes that the fees are governed by an exchange rate that is "derived by looking at a basket aggregate of currencies," that the rates will be not be "stale" and that the oracle will be "trusted." And the MIT License for the Synthetix software, which is provided as a file on its Github page, states that the software is provided "AS IS."[143] How to compile these various statements is the subject of our next Section.

III. SCRIPTS AND STACKS

As we explored above, it simply is not practical to create scripts that perfectly embody their coders' intent. A 2019 decision issued by the Singapore International Commercial Court, applying the common law of Singapore (itself derived from English common law), offers a unique window into how jurists might resolve the contractual consequences of code gone awry.[144]

The case involved a lopsided trade of cryptocurrency. Quoine, a Singapore-registered firm, operated a centralized currency exchange platform that primarily enabled trading of cryptocurrencies. B2C2 was an "electronic market marker" based in England and Wales, which had developed a trading algorithm written by its president several years before the events of the case. In April 2017, Quoine, as it always did, had a program in

---

[142] The webpage does include a link to "terms and conditions" but they govern use of the website ("By accessing the website at http://synthetix.io, you are agreeing" …"arising out of the use or inability to use the materials on Synthetix's website") rather than the blockchain platform.

[143] https://github.com/Synthetixio/synthetix/blob/8f3b95d1205f2b4d6b62124bd07f593773800743/LICENSE

[144] B2C2 v. Quoine Pte Ltd, 2019 SGHC (I) 03 (2019).

place to monitor users who had borrowed collateral with which to trade. When Quoine's program identified an imbalance in the reserves, it forced the sale of collateralized assets at the best available price.

Unfortunately, the program that Quoine had written to ensure a liquid market temporarily failed. The result was its prices were out of sync with the global market. B2C2, whose goal was to capture returns from the bid-ask spread,[145] offered to fill seven particular orders at a price around 250 times that available on the broader market, making it a quick profit of several million dollars.[146]

The next day, Quoine reversed the trades in its order book. B2C2 sued, arguing that the reversal violated Quoine's terms and conditions, which provided that "once an order is filled, you are notified via the Platform and such an action is irreversible."[147]

Quoine offered two principal defenses to its supposed obligation to complete the trades with B2C2.

First, it pointed out that its risk disclosure statement, uploaded prior to the trade but not expressly incorporated into the terms and conditions, permitted a reversal of trades if "market circumstances shift dramatically or something else happens."[148] The court held that because the statement was not expressly incorporated into the contract (that is, into the terms and conditions), it did not override the express language of the contract itself.[149] This decision seems perfectly sensible on its face—an application of the usual idea that an integrated agreement ought not be contradicted by extrinsic statements.[150]

Second, and more interestingly, the court considered whether the facts established a *mistake*, allegedly because the parties would not themselves have executed the trade in a hypothetical world where they talked about it in person in real time. That is, the parties' agents—the programs—had executed a deal which, though an accurate expression of the code's instructions, somehow failed to capture their "real intent."[151] As Quoine argued, the platforms were "really complex platforms", in which "a lot of things [could] go wrong."[152] The mistake in question was not a

---

[145] *Id*. at 5.
[146] *Id*. at 12.
[147] *Id*. at 55.
[148] *Id*. at 64.
[149] *Id*. at 73.
[150] For this point, the tribunal cited a British treatise, *id*. at 69 and supporting Singaporean authorities, but the result would be no different in most US jurisdictions. *See infra* at text accompanying notes 227 through 228.
[151] *See generally* TIMOTHY MURPHY, 1 CORBIN ON CONTRACTS: FORMATION OF CONTRACTS, REV. ED SEC. 4.11 (2018) (describing old cases on telegraphs and mistakes in transmission of messages)
[152] B2C2 v. Quoine Pte Ltd, 2019 SGHC (I) 03, 30 (2019).

typographical error in the code, but rather an "oversight in the design of the system."[153] Quoine asked the court to apply a "pragmatic and judicious stance" and void a "clearly erroneous trade."[154]

The tribunal, adopting a narrower reading of unilateral mistake derived from British common law, held that to avoid liability, the person who was not mistaken must have actually known of her counterparty's error "was shutting her mind to the obvious."[155] In deciding whether B2C2 had that requisite state of mind, the court noted conceptual difficulties:

> "[A]pplying the law to. . . algorithmic trading. . . raise[s] new questions. What mistakes have been made and to what extent are they fundamental? How does one assess knowledge or intention when the whole operation is carried out by computers acting as programmed? Whose knowledge is relevant? At what date is this knowledge to be assessed?"[156]

Given that the program executed a fixed trade—i.e., the design of the code bespoke the programmer's intention to strip it of any discretion—the court analogized the program to a "mere machine carrying out actions which in another age would have been carried out by a suitably trained human. [It is] no different [from] . . . a kitchen blender relieving a cook of the manual act of mixing ingredients."[157]

The court thus held that for the purposes of ascertaining mistake, it would focus on the intent of "the person on whose behalf the computer placed the order in question [B2C2]."[158] This, the court explained, required examining the "state of mind of the programmer of the software of that program at the time the relevant part of the program was written."[159] Assessing the relevant evidence, the court decided that the programmer had not inserted code intending to trade at lopsided rates, or knowing that doing so could only result from the counterparties' error. Thus, it rejected the claim of mistake.

Quoine is not a case about a transactional script gone wrong. But it does offer a few glimpses into the future of how scripted exchanges will be resolved, or at least one possible approach to such problems.

---

[153] *Id*.
[154] *Id*. At 79.
[155] *Id*. at 80; *cf*. Rst 2nd Contracts 151 (knew or should have known).
[156] *Id*. at 86.
[157] *Id*. at 89.
[158] *Id*. at 87-88.
[159] *Id*. at 89-90.

The first is the importance of the role that old-fashioned contract law—and old-fashioned contracts—will play in the disposition of the parties' legal rights. The *Quoine* tribunal privileged the natural language contract embodied in the terms and conditions over the code. Though it ultimately declared that the risk disclosures outside the terms and conditions were not binding, it apparently would have enforced those disclosures, had they been incorporated, over software code that enabled the trade. It was only in the absence of contract terms governing the deal that the court turned to what the code permitted, and why. Thus, the coded rules of exchange were, in the court's view, largely irrelevant.

Second and relatedly, the *Quoine* court's focus on the intent of the programmer is a natural outgrowth of contract caselaw. Interestingly, *Quoine* focused on intent at the time of the original programming. Knowing what B2C2 intended required the court only to take the testimony of one coder, and to apply the normal judicial tools (assessing demeanor, consistency with prior statements and the documentary record) to determine the "truth" of the coder's intent.[160] Such a simple story is unlikely to replicate when we interrogate more complex coding problems.

We think that the *Quoine* decision offers a rough guide to the sorts of problems that transactional scripts will raise, and a sense of how common law courts will be motivated to resolve them. Indeed, we think the tribunal got it mostly right in context, but that its reasons won't scale. Given what we've learned about the technological environment generating scripts, we make two basic arguments about how the law ought to consider problems related to scripting.

First, judges should not necessarily privilege "contract" over "code" but rather ought to ascertain and harmonize meaning across a *contract stack*. Code—when read with its natural language comments and commit logs—has communicative meaning that courts should seek to ascertain and enforce. Second, conditional on integrating the stack, the search for meaning should focus on expressions that best reflect the best public-facing account of the parties' shared intent at the time that they committed to the deal. We work through these principles by suggesting a list of canons of scripted law.

A. The Canonical Stack

As Jason Allen has recently argued, transactional scripts are the latest in a series of "contractware, i.e., technological artefacts

[160] *Id.*

designed to embody and perform contracts."[161] A chip in a credit card embodies the concept well. There is a natural language credit agreement between you and your card company, updated and modified at the issuers' will against the background of regulation, which define the circumstances under which credit may be extended. Those terms parallel ones agreed to between the issuer (or its agents) and merchants, which enable your card (the artefact resulting from your natural language agreements) to effectuate a pending sale by insertion into the merchant's reader. That is, the card/reader performs contracts that hover in the air around them.

All contractware has this property: it is wrapped within an ordinary, legal contract. Absent a legitimate connection to those contracts, use of a credit card can still affect the world—your account will be debited, the merchant's credited—but such changes can be quickly reversed. Of course, not all contractware requires physical manifestations. Allen argues that transactional scripts are an example of contractware in which "the subject matter of the contract is an immaterial object which can be manipulated directly by the [code.]"[162]

Allen concludes that transactional scripts are layers in the "contract stack." The stack is a useful metaphor to describe the various elements of "contract as a complex entity."[163] A stack might include, perhaps, an oral contract (or other indicia of social agreement), a written text, and the legal rules which give effect to the relationship between the two. In a transactional script, the "written text," i.e. natural language terms like the statement of terms and conditions in Quoine, is "complemented (or supplanted) by code which is also, incidentally, wholly or partially executable by a machine."[164] Each additional step—the compiling of machine-readable code from human readable code—adds another layer of complexity to the stack, but the stack is intended to operate, and therefore ought to be read, as a whole.[165]

Many working in this space acknowledge that contract stacks will be the mechanism through which commercial projects will deliver scripted performance.[166] This is the apparent impetus, for example, for the International Swaps and Derivative Association's attempt to insert explicit references to scripts which operationalize interest payments into the ISDA master

---

[161] Allen, *supra* note 32, at 5.
[162] *Id*. at 9.
[163] *Id*. at 18.
[164] *Id*.
[165] *Id*. at 19.
[166] DE FILIPPI AND WRIGHT, SUPRA note 32, at 77 (arguing that "hybrid" contracts will be useful especially when particular components cannot be reduced to code).

contract.[167] But other sophisticated projects, like the OpenLaw cooperative, are explicitly developing stacked deals.[168]

But we'd go further. *All litigated scripts will exist in contract stacks*. That's not to say that there will always be a 100-page master agreement, or even clicked-through terms and conditions of sale. Rather, our claim is that there will *always* be *some* non-code statements—ranging from the highly formalized terms and conditions, to less formalized white papers and code commentary, to quite informal promises (twitter)—which will inform tribunals' understandings of what the parties intended to exchange. This is true both as a matter of practice, and a matter of logic. Code is not self-descriptive. Any scripts that have practical relevance will have some non-code language surrounding them. Any set of communications relevant to the exchange that are visible to the parties are at least presumptively a part of the stack. (Whether they all count equally is a harder question.)

Thus, although some in the literature have asked if a "smart contract" is really a contract, standing alone, we doubt the practical relevance of that question.[169] The code standing alone may not fully specify an executory contract, unliteral or otherwise, because it simply accomplishes performance. But, unlike the cheese, at least at the moment, the code never stands alone. As our examples in Part II illustrate, it is typically deployed in a social context.

Though the *stack* is a relatively recent term, the idea that transactions are accompanied by many sorts of potentially legally-operative promises is not. You buy a car, led to the dealership by a commercial, and see a price at the entrance to the lot. The dealer makes representations to you. You reply with your own admissions. Finally, you sign a written agreement, which often seeks to exclude the prior representations as outside of the operative deal. The whole exchange is a contract stack. Some of it is legally operative, and some is not. When you buy a cup of coffee at the store after seeing a sign at the door, the process is the

---

[167] Int'l. Swaps & Derivatives Ass'n., Legal Guidelines for Smart Derivatives Contracts: The ISDA Master Agreement 7 (2019).

[168] *See* OpenLaw FAQ, https://app.openlaw.io/faq#first_draft__automate ("On OpenLaw, you can execute smart contract code by embedding a smart contract call in any template.")

[169] Some have argued that the parol evidence rule would make it difficult "for the parties to prove their intent to contract by pointing to other circumstances, such as prior dealings or negotiations." Anna Duke, *What Does the CISG Have to Say About Smart Contracts: A Legal Analysis*, 20 CHI. J. INT'L. L. 141, 159 (2019). This is extremely puzzling, as the rule itself only applies to fully integrated agreements. Others suggest that when the parties have "signed and verified that the contract had been accurately translated into computer code," it will be difficult for them to later argue that there were additional terms. Alan Cohn, Travis West & Chelsea Parker, *Smart After All: Blockchain, Smart Contracts, Parametric Insurance, and Smart Energy Grids*, 1 GEO. L. TECH. REV. 273, 280 (2017). This too is confusing, since the typical way that parties would verify that the contract is correct would be to agree to that stipulation in a natural language agreement.

same—though if you use an app to fulfill the purchase, some of the code that accomplishes payment is obscure to you and is, perhaps, not part of the contract. The point is that what counts as the "contract" is a post-hoc legal construction, which sometimes maps onto a single document, but often does not.

We thus suggest our first canon—a maxim—to help courts make sense of the adjudication of disputes arising from scripted exchange. Like all of the rules that we suggest, it builds on existing caselaw. It is also a default rule: the parties can (and as we explore below) have changed it by contract.

Canon 1: All shared communications of intent, including the code, comprise the legally-operative stack.

In working through this canon, an example may help. Consider a token white paper makes a promise about governance rights, but the script contains no reference to that promise.[170] That was the norm in the 2017 ICO craze. In previous work, we conducted an exhaustive audit of the match between semantic disclosures (including those in White Papers, Twitter, Instagram, Reddit, and Medium) and scripted code in tokens. We found that most programs had not, in fact, created code that conformed to their promises. For over 20% of ICOs in our sample where promoters promised cryptoasset supply restrictions, and 35% of promised token burning, we could not observe corresponding restrictions written into transactional scripts. Worse, we did not find code-based vesting restrictions in twenty-five of the thirty-six ICOs where promoters promised to adhere to such restrictions. Finally, of twelve ICOs for which our audit revealed that a central party could modify the functionality of the cryptoasset's code, only four disclosed that ability in their promotional materials.[171]

To the extent that an action for legal breach of such a contract stack is brought, how do we know what was promised? Some might argue that the only legally operative promises are those made in natural language documents outside of the code; others, that the code provides the only relevant set of rules. In a way, this is quite similar to start of a conventional parol evidence problem where multiple documents are candidates for inclusion as the litigated "contract."[172] When the bounds of contract are

---

[170] *Cf.* Cohney *et al.*, *supra* note 2, at 640-44 (discussing market solutions to missing disclosures).

[171] *Id.* at 639-40.

[172] *See* Gregory Klass, *Parol Evidence Rules and the Mechanics of Choices*, 20 THEORETICAL INQUIRIES IN LAW 457, 465 (2019) ("a writing is integrated if an only if the parties together intended that it would serve as a final statement of some or all terms of their agreement.").

fuzzy—that is, in the absence of integration—the stack of contractware is capaciously constructed. This problem is well-illustrated by the conventional 2-207 problem, where the parties have exchanged non-matching forms and the court must compile a "contract" *ex post.*[173]

Terms and conditions, because they clearly indicate the parties' intent to contract, are almost certainly part of the stack. So too are most published white papers, which make numerous promises about what's to be delivered, even though in some ways they are offers directed to the world. The reason is obvious: white papers are the best evidence of what the code supplier intends to create, and the way that counterparties are enticed to invest, contribute funds, or otherwise transact.[174] It might be (as we will explore) that terms and conditions provide better evidence of intent than do white papers, when both are present, but as a starting point, the white paper ought to be considered as part of the set.

More transient promises, like those made on social media posts, on reddit, or via video, are also plausible candidates for the stack. Jurists will ask if they make promises sufficiently definite and certain to enable a fact finder to generate the grist for obligation.[175] This will turn on the nature of the promises made.[176] Does a post on reddit stating the precise amount of a capped supply count?[177] We think so. But one that states that a project "discussed burning its tokens" without more detail seems to be too vague to generate obligation.[178]

The hardest problem is the contractual status of the script, including its commentary. It's hornbook law that the parties can use ciphers to express themselves, and that courts ought to enforce such coded meanings "however we may marvel at the

---

[173] *See* John D. Wladis, *The Contract Formation Sections of the Proposed Revisions to U.C.C. Article* 2, 55 SMU L. REV. 997, 1012 (2001) ("The exchange of non-matching records rarely involves parol evidence issues because there is usually no one record that is a final expression of the parties' agreement.")

[174] *See generally* TIMOTHY MURPHY, 1 CORBIN ON CONTRACTS: FORMATION OF CONTRACTS, REV. ED SEC. 2.12[1] (2018) (describing "webpage whose contractual nature is not obvious" as one where courts will ask if "the recipient actually knows or has reason to know of it when she assents to it.")

[175] *See generally* ID., Sec. 1.1 (2018) (defining offer "as an act whereby one person gives to another the legal power of creation the relation called contract")

[176] For examples of the sorts of marketing promises made in ICO disclosures, *see* Cohney *et al., supra note* 2, at Appendix C, https://columbialawreview.org/content/coin-operated-capitalism-appendix-c/

[177] *See ICON Technical Q&A Summary*, r/hellocin, Reddit (Sept. 18, 2017), https://www.reddit.com/r/helloicon/comments/70t56h/icon_technical_qa_summary/ [http://perma.cc/GXW4-F2K4].

[178] *See* Aetrnty, *Comment in Burning Token*, r/Aeternity, Reddit (Sept. 11, 2017), http://www.reddit.com/r/Aeternity/comments/6za07b/burning_token/ [https://perma.cc/483H-EQV3].

caprice."[179] In deciding whether to give effect to private meanings, courts traditionally engage in a hypothetical inquiry: *if* the parties had been queried at the moment of contracting about the meaning of a particular term, what would they have jointly said? Courts will thus self-consciously adopt the parties' expressed "vernacular."[180] Unilateral and uncommunicated meaning bear as little on the problem of contractual interpretation as the public meanings that the parties meant to cast aside.[181]

A traditional requirement about non-traditional communications is that the party against whom they would operate reasonably understands them to be contractual in nature.[182] Context matters: the more impermanent the medium, the less obvious it should be that a bargain results from the posting.[183] This is why, in an old case, a court found that the language on the back of coat check tickets was unenforceable: such scraps "did not arise to the dignity of a contract."[184] Promises made in terms and conditions feel like contracts (or more or less), while those made on Twitter may not.[185]

But how about the code? The primary objection to including scripts as presumptive source of contractual intent is that the lay understanding of code is that it *does*, not *promise to do*. It therefore communicates little more than any operating machine. True, those in the know can learn the coder's sophistication from her script's elegance and economy, just as connoisseurs of font will make inferences based on this Article's typesetting. But that's not the same as the sort of promissory communication that normally generates obligation.

Moreover, because the code is inherently buggy—as we have explored—including it into the stack necessarily means that the stack will potentially always contain ambiguous evidence of what

---

179 MARGARET N. KNIFFIN, 24 CORBIN ON CONTRACTS: INTERPRETATION OF CONTRACTS, REV. ED SEC. 24.9 (1998); Smith v. Wilson, 3 B. & Ad. 728 (K.B. 1832) (holding that "parol evidence was admissible to sh[ow] that... the word thousand, as applied to [the contract], denoted twelve hundred"); Hurst v. W. J. Lake & Co., 16 P.2d 627, 629 (Ore. 1932) (holding the term "minimum 50%" to encompass "as low as 49.5%").

180 ID. at 24.13.

181 ID. at 24.6.

182 TIMOTHY MURPHY, 1 CORBIN ON CONTRACTS: FORMATION OF CONTRACTS, REV. ED. SEC. 2.12 [1] (2018) (discussing non-contractual documents not enforced for lack of inquiry notice).

183 In some contexts, statements directed at the whole world will be deemed to be advertisements. PETER LINZER, CORBIN ON CONTRACTS 2.4 (JOSEPH M. PERILLO, ED., 2010). But since white papers and the like aren't considered in the abstract here, but rather when accompanying the scripts which put them into effect, the better analogy is the supermarket circular distributed at the grocery itself. ID. at Sec. 2.7.

184 Healy v. New York, C. & H. R.R. Co., 153 A.D. 516, 519-520 (1912).

185 *See generally* Berkson v. Gogo LLC, 9 F. Supp 3d 359, 382 (2015) (arguing that "[electronic contracts] require clearer notice than do traditional retail"). *But cf.* Kristen Chiger, *When Tweets Get Real: Applying Traditional Contract Law Theories to the World of Social Media*, 3 ARIZ. ST. SPORTS & ENT. L.J. 1 (2012) (finding that tweets can create contractual obligation).

the parties "really" mean to accomplish. As our discussion above makes clear, however, the code is festooned with natural language comments and commit logs, which do, implicitly or explicitly, state what the coders are trying to accomplish. In light of the social conventions of the scripting industry today, it is reasonable to conclude that participants *believe* that the code-and-comments create obligations, i.e., that the script is contractual in nature. The very term "smart contracts" justifies this conclusion. The rhetoric of exchanging in *contracting* should be constitutive.

This conclusion is strengthened in light of our focus on public blockchains, where all of the relevant parties have access to the scripted rules, can inspect them, and can read the associated code commentary. By definition, we have examined scripts that are open to bilateral inspection. It would be as if we were considering a contract about the importation of chicken from abroad, between parties from different countries, that expressly incorporated a foreign language dictionary, and then denied that dictionary's relevance in understanding the parties' meaning. Course of performance is typically seen as a strong signal of meaning: the script is yet a clearer signal. It is the expressed performance itself, fixed contemporaneously with agreement.

Incentives provide a final reason to include code in the stack. If lawyers are on notice that the code has legal relevance, that it can create or destroy obligation, or help jurists to interpret them, they will begin to pay attention. The scripting industry at the moment is covered by few specific rules, because of uncertainty about the nature of the underlying asset type. Private regulation through lawyering—having lawyers work to understand exactly what the script says and inquire how it might go wrong—is a way to help to civilize this wild west. That is, bringing the script into the contract stack will motivate lawyers to care about coding, and coders, and consequently reduce the likelihood of promissory fraud.

B. *Tensions Within the Stack*

What happens if there are differences in what's promised between elements of the stack? Allen argued that questions of intent are particularly difficult when interpreting code, as the formal language might have meta-logics—internal and intentionally chosen goals (like, for example, compactness).[186] He thus suggests that courts may need to modify traditional canons of interpretation to think about how to best capture the parties'

[186] Allen, *supra* note 32, at 27.

meaning when working through layers of a stack.[187] He gives no further details, and this next part elaborates on the problem.

Courts will often say that interpretation ought to make sense of the "contract as a whole," that is, "the entire deed, and not merely upon disjointed parts of it."[188] That's particularly true when the parties express their agreements in multiple documents.[189] In determining meaning when there is ambiguity, extrinsic evidence—that is, anything other than the contract itself—is commonly admitted.[190] This leads us naturally to a canon seeking harmonization.

Canon 2: Where Possible, Interpret the Stack to Harmonize Meaning.

We start by returning to the ICO example. At least until recently, it was common for white papers to make promises about governance in ICO "smart contracts," but neglect to write actual scripts containing such coded rules.[191] In our view, the stack as a whole ought to be read to be making a promise: the absence of protection in the code should not be dispositive. If, for example, the projects' founders were to take assets in violation of a textual vesting promise, an action ought to lie for breach. That's true even though an informed reader with an understanding of Solidity would have readily observed that the tokens contained no promises. (Most, in fact, were essentially the unmodified ERC-20 code.) The point is that a *legally* informed reader could believe those promises to be enforceable based on the white paper alone, and their repetition within the script unnecessary.

Conversely, when the natural language disclosures say nothing about the ability to modify rights described as having been created in a token, but the script appears to permit modification, we might conclude that the contract stack permits

---

[187] Allen, *supra* note 32, at 27.

[188] KNIFFLIN, *supra* note 169, at 24.21 (quoting Blackstone).

[189] *See* Restatement (Second) of Contracts: Rules in Aid of Interpretation § 202(2) (Am. Law Inst. 1981) ("All writings that are part of the same transaction are interpreted together.")

[190] Note that New York courts, like many others in a modern formalist trend, take a different approach. *See, e.g.,* Gilbane Bldg. Co./TDX Constr. Corp. v. St. Paul Fire & Marine Ins. Co., 31 N.Y. 3d 131, 137 (2018).

[191] For example, the Monaco project promised that its supply would be capped in a transactional script: "The MCO smart contract will stop accepting commitments at 888,888ETH hard cap" Monaco, *Monaco Whitepaper, White Paper Database* 8, https://whitepaperdatabase.com/wp-content/uploads/2018/03/Monaco-MCO-Whitepaper.pdf [https://perma.cc/YXS9-5GQY] (last visited Jan. 26, 2019). But our audit disclosed no such scripted commitment. Cohney *et al.*, *supra* note 2 at Appendix B.

modification.[192] In the search to understand what the contract stack promises, neither code nor script ought to prevail over the other, again unless the parties otherwise indicate. The goal is to determine what the parties intended, expressed in whatever cipher they chose. We should not *a priori* dismiss rights provided in code.

The TokenStore problem offers another setting for the harmonization principle. Recall that TokenStore had a vestigial legal wrap in place: a handful of twitter and medium posts, making vague gestures about the exchange's commitment to being a hands-off-enterprise. The code permitted trader errors—indeed, it did nothing to prevent them. However, the code *commentary* stated that "we [i.e., TokenStore's operators] validate the contents and the creator address" of the "orders."

It's not clear what the writers of this comment intended it to mean. In programming communities, validate can take on a narrow meaning—someone can enter an "input in a form that is not expected," leading in "arbitrary control flow, arbitrary control of a resource, or arbitrary code execution."[193] But it can also take on a broader meaning, i.e., that the code produces commercially reasonable results.[194] And perhaps such narrow programming meanings are only relevant in commercial markets, so that "validate" ought to take on an ordinary meaning, i.e., "to make legally valid." This would mean that the exchange bears the risks of obvious errors. Such a reading would be helpful to a trader's action for rescission based on mistake (which will turn in part on what the contract says about risk) as well as that for ordinary breach of contract. In our view, the correct approach again would seek to make sense of the gestalt project, treating the code and its natural language comments as guides to parties' joint intent.

Harmonization becomes difficult when parts of a contract conflict.[195] When pieces of a deal counterpose, courts traditionally first seek to ascertain the parties' principal purpose, and then to advance it by deciding out which pieces of evidence to privilege. For example, handwritten terms prevail over typewritten terms, and specially typed provisions control over pre-printed forms.[196]

---

[192] *See generally* Cohney *et al., supra* note 2. at 630-34. Of course, a court could find that, in light of an industry-wide practice of describing rights as "immutable," silence in one layer of the stack should be interpreted against a commercial background denying modifiability. *Id*. at 615, n.114.

[193] Common Weakness Enumeration, *CWE-20: Improper Input Validation*, https://cwe.mitre.org/data/definitions/20.html

[194] The Open Web Application Security Project, *Data Validation: Business Rules*, https://www.owasp.org/index.php/Data_Validation

[195] *Cf*. Surden, *supra* note 31, at 657 (noting an "unresolved tension . . . in future scenarios where there is both a written and data-oriented representation of the same contractual expression, with interpretations that differ.")

[196] *See generally* 5 CORBIN ON CONTRACTS § 24.23.

Courts foreground those provisions that they believe are best indicators of what the parties "really" meant.

Scripted exchange will sometimes also pose problems of inconsistent intent. Given that pieces of the stack are written at different times, by authors with distinct professional backgrounds, and intended for different readers, we should anticipate conflicts in meaning. As a default rule, we'd propose a hierarchy of meaning, which privileges natural language wrapping text over code when they conflict.

> **Canon 3:** In cases of conflict, privilege natural language promises over coded ones: i.e., wrapping text, commitment messages and code commentary over code, high level code over byte code.

By wrapping text, we mean to include any text that is outside the code itself, but within the stack. When such text conflicts with code, we think the parties' contract—what they can sue on—most likely turns on their natural language expression.[197] This is a pragmatic choice.[198] As with the *Quoine* tribunal, most judges are going to have a natural affinity for text that they can read without the aid of an expert translator.[199] They will be motivated to believe that "no one reads smart contracts."[200] (The fact that no one reads regular contracts is equally true, though it feels easier to blame them for it.)[201] Of course, just like contract text, code *can* be "read," and often results from a similar iterative drafting process as old-fashioned contracts.

Why, then, privilege English over Solidity? One reason sounds in the classic worries about opportunism and bad faith that drives many judicial treatments of adhesion contracts. If parties could avoid a contractual promise by negating it in code that you had no reason to think would be read, we would rightly worry about promissory fraud, or other forms of bait-and-switch behavior. Though today, many users of transactional scripts are sophisticated—even to access scripts, you usually install specialized software on your computer—that may not be the case

---

197 *See* Rohr, *supra* note 32, at 81 ("[C]ode [that automates a larger agreement] is likely to be viewed as a component of performance that one party will attempt to prove is nonconforming.")

198 For a defense of a search-costs based theory of parol evidence and other rules that limit "idiosyncratic understandings", *see* Joshua Fairfield, *The Search Interest in Contract Law*, 92 IOWA L. REV. 1237, 1265-67 (2012).

199 In consumer-facing transactions, courts also may worry about exploitation when parties use language that is hard for adherents to understand. *See, e.g.*, Frostifresh Corp. v. Reynoso, 274 N.Y.S.2d 757 (Dist. Ct. 1966).

200 Cohney *et al., supra* note 2, at 598.

201 *See generally* Tess Wilkinson-Ryan, *A Psychological Account of Consent to Fine Print*, 99 U. IOWA L. REV. 1745 (2014).

going forward. Given that English is easier to read than Solidity, courts will likely privilege natural language promises wherever they can.

But at a deeper level, our intuition is that courts imagine they are looking for something they call "real" intent, but which is really more like what the parties expressed about their intent to the world.[202] Just as with other forms of commercial transactions drafted and entered in stages, discerning real intent is often a fool's errand.[203] That inquiry falsely implies that the parties gave the problem some thought. When courts speak about intent, they are engaging in a hypothetical and imaginative exercise, which entails significant degrees of analytic freedom. But imaginative exercises must be explainable in public judicial opinions, and thus rely to a degree on text that can be read by the widest audience, and which is susceptible to the cheapest judicial oversight.[204]

As we've shown, code is irreducibly buggy, and the normal ways that coders handle error—by iterating better versions—may not translate well to scripted exchange. Code simply isn't a very straightforward way to express the parties' intent. By contrast, parties have had hundreds of years of experience contracting in English (or French, or Esperanto, turning on the court's and parties' native tongue). Unless there is good evidence that a particular line of code was made salient—for example, if it is referred to by line number in the natural language contract itself—courts should conclude that text trumps code.[205] A combination of realism and efficiency, at the end of the day, will privilege publicly accessible meaning.[206]

Whether the text wrapping layer should displace case *commentary* and *commit logs* is a much harder problem. Here, the concerns about publicly accessible meaning drop away, since code commentary is generally written in English (or at least a coding dialect that can be grokked). The remaining issue is whether commentary intending to explain a cipher is as good evidence of

---

[202] *See* Rohr, *supra* note 32, at 78.

[203] *See* Douglas G. Baird & Robert Weisberg, *Rules, Standards, and the Battle of the Forms: A Reassessment of 2-207*, 68 VA. L. REV. 1217, 1219 (1983); GRANT GILMORE, DEATH OF CONTRACT 47 (2d ed. 1995) ("[I]f "the actual state of the party's minds" is relevant, then each litigated case must become an extended factual inquiry into what was "intended," "meant," "believed" and so on.")

[204] An analogy is the courts' treatment of disputes about meaning based on differences in the parties' respective native tongues. In such cases, the courts may adopt the broadest and widest-shared meaning available. Frigaliment Importing Co., Ltd. v. B.N.S. Int'l. Sales Corp., 190 F.Supp. 116 (S.D.N.Y. 1960).

[205] Rohr points out that courts analyzing vending machine contracts were drawn to "meeting of the mind" analogies, even when they plainly were inapt. Rohr, *supra* note 32, at 80.

[206] For a different defense of publicly accessible meaning, see generally Aaron D. Goldstein, *The Public Meaning Rule: Reconciling Meaning, Intent, and Contract Interpretation*, 53 SANTA CLARA L. REV. 73 (2013) (arguing that extrinsic evidence should be limited to public and shared meaning to avoid gamesmanship).

what the exchange was intended to accomplish as the wrapping text's more legalistic frame.

On the one hand, the commit logs and commentary is integral to the code itself, expressed contemporaneously with its fixation and with the goal of revealing its intent.[207] It is the "crown jewel" of the code, laying bare its "inner secrets."[208] Code commentary and commit logs are thus like the definition section of an ordinary contract: the very best evidence of meaning.[209] Consequently, commentary and commit logs should be privileged over the code it explains.

That said, some might worry that code commentary (and, to a lesser extent, commit logs) in Solidity is intended to be disposable[210]–it might signal what a coder hoped to achieve but is not likely to have been written with particular care.[211] For example, as we discussed above, most open source code today is reused from script to script. Thus, it's not necessarily (or even usually) the case that the commentary was written with a singular's project's goals in mind. Blindly adopting such commentary as gospel risks being misled as to what the parties wanted, just as (for example) adopting boilerplate can, over time, lead parties to using terms that even they do not understand.[212]

Even considering these risks, commentary and commit messages should have the same interpretative weight as natural

---

[207] Daniela Steidl, *et al., Quality Analysis of Source Code Comments*, in 2013 21st INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION 83, 83 (Inst. of Elec. & Elec. Eng'rs., 2013) ("A significant amount of source code . . . consists of comments, which document the implementation and help developers to understand the code . . . . Comments are the second most-used documentary artifact for code understanding, behind only the code itself.").

[208] Jeffrey Sullivan & Thomas Morrow, Practicing Reverse Engineering in an Era of Growing Constraints Under the Digital Millennium Copyright Act and Other Provisions, 14 ALB. L.J. SCI. & TECH. 1, 17 (2003) ("Reverse engineering does not lay bare a program's inner secrets. Indeed, it cannot. The inner secrets of a program, the real crown jewels, are embodied in the higher levels of abstraction material such as the source code commentary and the specification").

[209] There may be examples where coders make an explicit attempt to synthesize the semantic contract within the code. *See, e.g.,* Demo Lexon, http://demo.lexon.tech/apps/editor/ (embedding human readable semantic contract within compliable code that exports to Solidity).

[210] Andrew Johnson-Laird, *Software Reverse Engineering in the Real World*, 19 DAYTON L. REV. 843, 857 (1992) ("[Source code commentary] is the equivalent of marginal annotations and is intended to assist the original programmer or those that follow in understanding why the program was crafted in a particular way, or to explain a particularly complex flow of logic. There are no restrictions on what must or must not be written in comments, but inevitably they are the repository of all the knowledge that the programmer has in his or her head as the code is being created. One also frequently sees a certain irreverence in the commentary which is a by-product of the exuberance of programmers and is best not taken too seriously . . . .")

[211] *Cf* Haque *et al, supra* note 50, at 183 ("Another [developer] may constantly contribute a high volume of lines over a long period of time, but still be a functionary whose work is wholly non-essential and could easily be replaced by others.")

[212] *See generally* Stephen J. Choi, Mitu Gulati & Robert Scott, *The Black Hole Project in Commercial Boilerplate*, 67 DUKE L. J. 1 (2017) (discussing pari passu clauses).

language contract terms. True, they might not provide clear evidence of meaning. But the same objections can be made about boilerplate that travels from deal to deal. Moreover, though it's true that many coders treat commentary and logs as disposable, well-counseled projects, knowing the rule that we propose, would be well-positioned to clean up the commentary before deploying a script, and impose discipline over commits that are merged into the project. The result could be another opportunity to surface and correct bugs, while aligning the parties' expectations with what they receive. And, of course, this is just a default rule: the parties may express a different rule by contracting for it.[213]

Finally, we think that the source code generally is a better source of meaning than the compiled byte code. That's so because the source code is, in broad strokes, readable by humans with the exercise of reasonable effort, meaning that all parties to a transaction can gain some insight as to what they've agreed. The alternative, which holds that byte code is the "real" contract, seems likely to lead to embarrassing results. For example, consider this "online user agreement" which a law student confronted:

[213] *Cf.* Surden, *supra* note 31, at 652 (describing "data-meaning threshold agreement" to give "specific interpretations" for computable contracts).

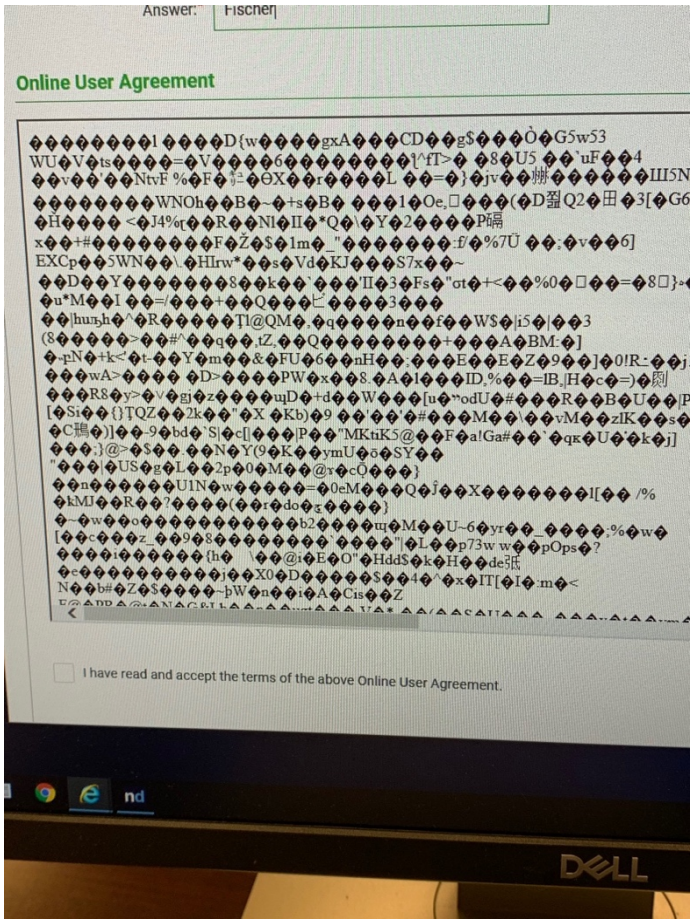


Figure 5: Online User Agreement[214]

This is not bytecode, but rather an erroneous encoding of the bytes into the glyphs that represent them. Obviously, clicking on the agree box wouldn't bind anyone—the gibberish communicates no meaningful information to humans (even if some computer, somewhere, could make sense of it). Byte code is likewise at the very bottom of our interpretative hierarchy, as decoding it requires the exertion of effort and expertise likely beyond the capacity of the contracting parties.

These interpretive principles ought to give way depending on context. Thus, we imagine that ephemeral contractual promises—a stray tweet by a project manager promising a particular outcome of the script—would likely not displace a well-curated GitHub repository. Or, a piece of code commentary that makes a joke. Here, again, our argument rests on a pragmatic judgment as to what courts will do, which in turn relates to the parties'

---

[214] Samuel P. Morse (@SamuelPMorse), Twitter (Sept. 13, 2019, 9:10 AM), https://twitter.com/SamuelPMorse/status/1172497664799363075.

reasonable, commercially-informed expectations.[215] The more permanent, considered and reliable the evidence of a promise—the more, in other words, it would be reasonable to rely upon it—the more a court is likely to consider it a reliable basis of the bargain.

Finally, consider a problem of interpretation that has no easy non-scripted analogue. What if the programmers' intent is internally contradictory *and is recorded as such*. Of course, courts will often say that a parties' private meaning ought to be discounted when ascertaining the shared intent of the transaction. Thus, simply because one lawyer on a team thought that "black" means "white" doesn't mean that the term takes on that meaning, unless that lawyer communicated her meaning to the other side in a way that seemed authoritative.[216] This is why some have argued that visible metadata ought to bear on meaning.[217]

Evidence of inconsistent drafter intent in the transactional script context is different. Given the use of version control systems, both the expression of code and the identity of who wrote each line of human-readable code may be knowable at the moment the counterparty inspects the terms, and certainly at the moment of formation. So is the commentary. Because the code is public, all parties can see the conflicting evidence of meaning at the moment the contract is entered: it is like the parties final executed document included visible redlined changes.[218] Thus, a party might encounter a piece of code that has multiple drafters who appear to be communicating different goals. What to do in this scenario?

The simplest answer—harmonization with the rest of the agreement and with the social context—is probably best. But its application is subtle and rooted in the sociology of the operative coding community. We ought to prefer (and render operative) the meanings of coders who advance the larger agenda of the project, to the extent it can be reconstructed.[219] Failing that, we should prefer later in time to earlier in time pieces of code, and code which hews closely to the commentary that surrounds it. These

---

[215] Timothy Murphy, 1 Corbin on Contracts: Formation of Contracts, Rev. Ed Sec. 4.12 (2018) (role of reasonable interpretations in determining meaning).

[216] *See* Cendant Corp. v. Commonwealth Gen. Corp., No. 98C-10-034 HLA, 2002 WL 31112430, at *6 (Del. Super. Ct. Aug. 28, 2002) (holding that the drafting history created issue of material fact).

[217] *See, e.g.* Thomas H. White, *Parol Metadata: New Boilerplate Merger Clauses and the Admissibility of Metadata under the Parol Evidence Rules*, 4 CASE W. RES. J.L. TECH & INTERNET 237, 267 (2012) ("If the metadata is visible on the final version of the contract, this should be conclusive weight in favor of its admissibility.")

[218] For an argument that non-resolved tracked changes ought to be included as part of the integrated document, see Elizabeth A. Janicki, *Note, Contracts as Speech Acts: Bringing Jakobson to the Conversation*, 107 GEO. L.J. 201, 218 n.113 (2018).

[219] Hunn, *supra* note 33, at 281.

match the background rules of contract interpretation, which generally seek to find and give priority to the best available evidence of the parties' expressed and integrated intent.

Overall, this interpretative hierarchy, which promotes supervised coding and last-in-time syntax, arises out of the same intuitions that generate the parol evidence rule. Recall Corbin's famous, though not universally accepted, rationale for the rule.[220] It was not, he argued, primarily about controlling self-serving frauds by excluding convenient ex post evidence of meaning.[221] Rather, the rule was intended to give priority to the parties' fixed agreement: to discard those earlier agreements, negotiations and understandings which had not made it into the final and binding contract.[222] On the question of whether the parties intended a particular expression to be the final expression, "no relevant evidence, parole or otherwise, is excluded."[223] But even if admitted, courts could clearly weigh such evidence and find that "the more bizarre and unusual an asserted interpretation is, the more convincing must be the testimony that supports it."[224]

On this understanding, if a transactional script's promotors seem to advance multiple potential purposes at odds with one another, a court ought to identify those terms that best match the *reasonable* expectations of the parties at the time the moment that the counterparty committed to the particular exchange. The existence of prior conflicting terms would still be potentially relevant, but they ought to be deemphasized.

In applying this canon, consider the Synthetix example discussed above. One party deployed a bot that took advantage of a corrupted third-party oracle to execute a trade which would have garnered it potentially a billion dollars in profit. Presumably because that amount would have been uncollectable, the hacker settled for some undisclosed bug bounty, paid off-chain. Neither party tested what contract law would have had to say. But we can speculate somewhat as to its contents.

Canon 1 reminds us to take up all the sources of meaning. A license, deployed by Synthetix, disclaimed all warranties as to the Code's correctness. But its commentary promised that the quotes were the "current market value." Thus, if a court were to ask if

---

[220] *See generally* Arthur L. Corbin, *The Interpretation of Words and the Parol Evidence Rule*, 50 CORNELL L.Q. 161, 187 (1965) ("[W]hen the words of . . . a contract are plain and clear and unambiguous, no interpretation is required and no extrinsic evidence is admissible—evidence of antecedent history and surrounding circumstances and purposes and usages of the parties. The judge looks at the words of the written document alone and knows their true meaning by virtue of judicial notice.").

[221] Linzer, *supra* note 182, at 25.2.

[222] *Id*.

[223] *Id*. *But cf*. Klass, *supra* note 172, at 464 (describing some theorists' view that integration also controls interpretation evidence).

[224] Linzer, *supra* note 182, at 25.2.

the parties intended this result, the answer would turn, we think, on the hierarchy in *Canon 3*. In our view, as between two kinds of wrapping natural language, the commentary to the code ought to take precedence over the ambiguous license, meaning that Synthetix would have a difficult time arguing that whatever the oracle delivered was, for the parties' purposes, actionable "market values." At the very least, Synthetix should have born a heavy burden of proving an alternative.[225] Thus, Synthetix, which presumably would have argued mistake, probably fairly bears the burden of a third-party data source risk.

Canon 4: Where Natural Language Contracts Refer to Code, Integration Clauses Should Be Read Narrowly.

We have suggested a set of default rules for interpreting scripts. But what if the parties want to vary such rules by agreement: should we give these scripted integration clauses force?[226] Offline, courts have generally permitted integration clauses to control which pieces of prior or contemporaneous contracting are included within the litigated deal, at least between sophisticated firms.[227] But courts have sometimes been dubious about attempts to integrate fuzzy stacks and to limit evidence of meaning that appears otherwise relevant.[228]

The problem here is that it is almost impossible to imagine an integration clause in a wrapping contract that contains no references to the script that effects the exchange. Indeed, the state of the art today encourages parties to directly refer to the addresses where the script resides within the "contract." In such cases, we think it impossible to exclude the script entirely—in other words, we don't think that script can be both referred to and treated as extrinsic evidence. That would be much like saying that an addendum to a contract, which contains a key description of the relevant subject matter, is not a part of the deal.

What parties might do is to try to use the natural language contract to determine meaning. Natural language terms and conditions would state that the only operative promises are those found in the natural language itself, and that parties should not

---

[225] The case would have been different with an appropriate disclaimer in the terms of use, which we have not yet found.

[226] *See generally* Gregory Klass, *Intent to Contract,* 95 VA L. REV. 1437, 1442-43 (2009); Klass, *supra* note 172, at 466-71 (discussing the circumstances in which integration clauses are enforceable).

[227] *See* Klass, *supra* note 172, at 475-478 (arguing for a "hard express integration rule for firm-to-firm negotiated contracts).

[228] *Cf.* 6 CORBIN ON CONTRACTS § 25.7 ("If we accept the premise that the primary factor in contract formation is the parties' mutual intent, the essence of integration is whether they intended a document to be the final word, and evidence of this intention should be found from all sources, not just the words of the contract.")

read the commentary in the script to make promises about what the code accomplishes. This would not deny that the code is a part of the bargain, but rather would attempt to limit what can be inferred from the natural language text it contains. Or, the text might state that the script's execution controls over any other possibilities—that is, the natural language may deny its own efficacy.

The choice of whether to defer to these attempts to control meaning turns on whether the court generally adopts a more contextual or more formalist approach to interpretation.[229] Contextualism, which discourages opportunistic drafting and thus protects consumers, has much to commend it in markets where sharp dealing is more prevalent. Fraud has defined many blockchain products to date, as has incoherent transactional lawyering. This is not a space producing formalism's best factual predicates.

To the extent these issues seem fanciful, consider the DAO hack.[230] The DAO was a token-mediated platform that allowed small investors to enter jointly into a venture capital pool.[231] The entity's "terms," apart from disclaiming various legal rights, stated that the "use of The DAO's smart contract code . . . carries significant financial risk, including using experimental software."[232] However, it also stated:

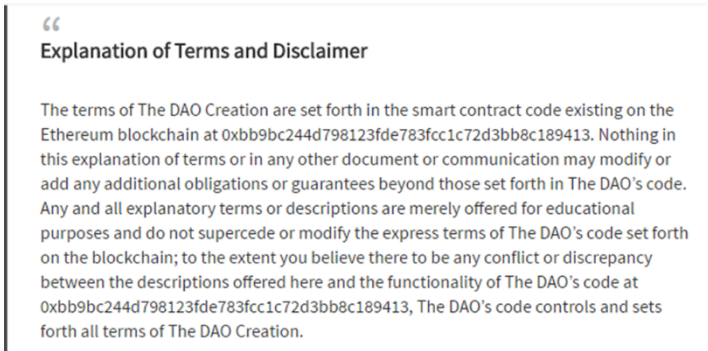
---

[229] *Compare* Wells Fargo Bank v. Cherryland Mall Ltd. P'ship, 812 N.W.2d 799, 810 (Mich. Ct. App. 2011) (admitting usage of trade notwithstanding contractual clause that "no trade practice . . . shall be used to contradict, vary, supplement or modify any term of this guaranty agreement") *with* Southern Concrete Services, Inc. v. Mableton Contractors, Inc., 407 F. Supp. 581, 584 (N.D. Ga. 1975) ("The court recognizes that all ambiguity as to the applicability of trade usage could be eliminated by a blanket condition that the express terms of the contract are in no way to be modified by custom, usage, or prior dealings."). *See generally* Lisa Bernstein, *Custom in the Courts*, 110 NW. U. L. REV. 63, 72 (2015) ("the enforceability and effectiveness of a general clause opting out of all trade usages is at best unclear"); Joshua M. Silverstein, *Contract Interpretation Enforcement Costs: An Empirical Study of Textualism versus Contextualism Conducted Via the West Key Number System,* 47 HOFSTRA L. REV. 1011 (2019).

[230] *See*, e.g., Haque *et al, supra* at 50, at 167-71; Laila Metjahic, *Deconstructing the DAO: The Need for Legal Recognition and the Application of Securities Laws to Decentralized Organizations*, 39 CARDOZO L. REV. 1533 (2018) (analyzing the corporate legal theories under which the creators of The DAO might be held liable for the 2016 hack).

[231] *See* Vitalik Buterin, *Bootstrapping a Decentralized Autonomous Corporation: Part I*, BITCOIN MAG. (Sept. 20, 2013), https://bitcoinmagazine.com/articles/bootstrapping-a-decentralized-autonomous-corporation-part-i-1379644274 ("What is a corporation, after all, but a certain group of people working together under a set of specific rules?").

[232] *Explanation of Terms and Disclaimer*, DAOHUB, https://daohub.org/explainer.html, archived at INTERNET ARCHIVE, https://web.archive.org/web/20160704190119/https://daohub.org/explainer.html (captured on Jul. 4, 2016) (last visited Nov. 4, 2019).



"

Explanation of Terms and Disclaimer

The terms of The DAO Creation are set forth in the smart contract code existing on the Ethereum blockchain at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413. Nothing in this explanation of terms or in any other document or communication may modify or add any additional obligations or guarantees beyond those set forth in The DAO's code. Any and all explanatory terms or descriptions are merely offered for educational purposes and do not supercede or modify the express terms of The DAO's code set forth on the blockchain; to the extent you believe there to be any conflict or discrepancy between the descriptions offered here and the functionality of The DAO's code at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413, The DAO's code controls and sets forth all terms of The DAO Creation.

Figure 6: The DAO's Terms

The code repository contained an even more incoherent set of disclaimers, including a readme file that claimed that using the software "does not, in and of itself, create a legally binding contract," and that "in order for you to form a legally binding contract you shall seek legal advice from an appropriately qualified and experienced lawyer."[233] This set of disclaimers appears to be an attempt to shield the entity from liability by at once embracing and rejecting contract law.[234]

After users had contributed funds, but before the DAO's own investments had begun, someone noticed a flaw in its code which allowed siphoning of $55 million (of around $170 million total assets) out of the pool. Ethereum's then developers promulgated a proposed software update to the entire blockchain – a hard fork – which was adopted by some, but not all, holders of the original tokens.[235]

The DAO's creators had some warning of the vulnerability.[236] Before the hack, a commentator posted about the vulnerability (titled, "Protect against recursive withdrawRewardFor attack") and suggested a seemingly easy change (reversing the ordering of

---

[233] Stephan Tual, *Updated Readme*, GITHUB: SLOCKIT/ DAO (Apr. 11, 2016), https://github.com/slockit/DAO/commit/aceec3efcc8afd4277396ebc42628f2e5ca8dff2#diff-04c6e90faac2675aa89e2176d2eec7d8.

[234] For a trenchant analysis of these issues, *see* Drew Hinkes, *A Legal Analysis of the DAO Exploit and Possible Investor Rights*, BITCOIN MAG. (June 21, 2016).

[235] Reyes, *supra* note 29, at 388; Vitalik Buterin, *Notes on Blockchain Governance*, Vitalik Buterin's Website (Dec. 17, 2017), https://vitalik.ca/general/2017/12/17/voting.html; The Ethereum Classic Declaration of Independence, Ethereum Classic (Aug. 11, 2016), https://ethereumclassic.org/assets/ETC_Declaration_of_Independence.pdf (declaring the holders' intent to "continue the original Ethereum blockchain" and decrying the hard fork as a violation of the blockchain's "core tenets").

[236] *See, e.g.,* Matthew Leising, *The Ether Thief*, BLOOMBERG (June 13, 2017), https://www.bloomberg.com/features/2017-the-ether-thief ("Gün . . . had already been tracking and publicizing flaws in the DAO's design. . . [He] appears to be the first to pinpoint the flaw that put the money in jeopardy.").

two lines of code) which would close it.[237] The DAO made the change, reassuring users that "The important takeaway from this is . . . this is NOT an issue that is putting any DAO funds at risk today."[238] The updated version was called The DAO 1.1 "milestone." In the code, the in-line comment preceding the new revision on line 580 stated its explicit purpose:[239]

```
    // we are setting this here before the CALL()
value transfer to
    // assure that in the case of a malicious
recipient contract trying
    // to call executeProposal() recursively money
can't be transferred
    // multiple times out of the DAO
```

Assurance notwithstanding, someone then executed the famous hack—in part because The DAO's fix was incomplete—transferring money multiple times out of The DAO.

Thus, here we again have a stack of meaning about what the parties to the contract—The DAO's creators and its investors—expected.[240] But the relevant documents are contradictory. The actual code did not accomplish what the comment or white paper promised, a fact that soon became obvious to all. Moreover, the organizers of The DAO had specifically told users that the code governed, even as they disclaimed the legal enforceability in the code itself.[241] Let's use the canons to offer a solution to the private law contracting problems that the DAO occasioned.[242]

As *Canon 1* instructs, we ought to consider all the relevant pieces as evidence of meaning, and of what was promised. The code, terms and conditions and readme files all are a part of the stack.

---

[237]LefterisJP, *Posting to Slockit/DAO*, GITHUB (Jun. 12, 2016), https://github.com/slockit/DAO/commit/f01f3bd8df5e1e222dde625118b7e0f2bfe5b680?diff=split.

[238] Stephen Tual, *No DAO Funds at Risk Following the Ethereum Smart Contract 'Recursive Call' Bug Discovery*, Slock.It (June 12, 2016), https://blog.slock.it/no-dao-funds-at-risk-following-the-ethereum-smart-contract-recursive-call-bug-discovery-29f482d348b.

[239]LefterisJP, *supra* note 219, at line 580, https://github.com/slockit/DAO/blob/d48ee5c49f9dc3b9548623aa6985cbc3c9528b67/DAO.sol#L580.

[240] The creators said at the time that they expected no financial benefit from its success. Stephen Tual, credited as a primary organizer, said: "No one benefits from it except the people that support it. Even we, the ones who invented it, get nothing." *See* Tanaya Macheel, *The DAO Might Be Groundbreaking, But Is It Legal?,* AM. BANKER (May 19, 2016, 3:12 PM), https://www.americanbanker.com/news/the-dao-might-be-groundbreaking-but-is-it-legal.

[241] *See generally* Kolber, *supra* note 24, at 218, 221 ("Note: Although the word "contract" is used in The DAO's code, the term is a programming convention and is not being used as a legal term of art. The term is a programming convention, not a representation that the code is in and of itself a legally binding and enforceable contract.").

[242] There are many public law and regulatory aspects to the story, but they are not ours to tell.

Canon 2 suggests that the DAO's counterparties reasonably could have believed that they would not face the risk of recursive money transfer.[243] That promise, embodied in a marketing announcement and in the code comments itself, is, it is true, absent in the operational code. It's as if a door which contained on its front face a sign stating "Private Property: Locked Door" was freely openable with a key hanging nearby. If someone were to have been harmed by relying on that set of statements, when suing in fraud or tort they would face questions about how reasonable their precautions had been.[244] But in a contract lawsuit, *Canon 3* teaches that code should bow to comments and commit logs in a way that best embodies what the programmers intended to accomplish and therefore to what they ought to be held.[245] *Canon 4* tells us to discount the attempts at non-integration as, at best, confused.

We thus disagree with The DAO's attacker, who argued that the fork violated its rights because those actions were literally permitted by the code.[246] Yes, code drafters ought to bear the interpretative risk of error.[247] But the non-drafting counterparties whose funds were taken could not reasonably be expected to know that the code had that bug, given the commentary promising the opposite result. Had they not received their money back, The DAO's investors should have been able to bring an action for breach against its developers, or even, perhaps, against the attacker if its action amounted to participating in the nexus of contracts that The DAO had proposed.[248]

---

[243] Rohr, *supra* note 32, at 85 arrives at a similar conclusion, noting analogies to vending machine cases to find that "the agreement includes only those terms that were reasonably available to DAO Token holders prior to purchase," which Rohr impliedly concludes would include the Terms of Use but not the permissive code.

[244] One problem we leave to future work is the relationship between putting statements inside the contractual stack and the tort-contract line in litigation.

[245] Thus, we reject the idea that the only operative promises are those written in code. *Cf.* Larry Lessig, *Code Is Law: On Liberty in Cyberspace*, HARV. MAG (Jan.-Feb. 2000), https://harvardmagazine.com/2000/01/code-is-law-html ("[C]ode . . . determines how easy it is to protect privacy, or how easy it is to censor speech. It determines whether access to information is general or whether information is zoned. . . . In a host of ways that one cannot begin to see unless one begins to understand the nature of this code, the code of cyberspace regulates.").

[246] A Guest, *An Open Letter*, PASTEBIN (June 18, 2016, 5:21 AM), https://pastebin.com/CcGUBgDG (claiming that the fork "would amount to seizure of my legitimate and rightful ether, claimed legally through the terms of a smart contract.").

[247] In this context, perhaps all participants, including investors and programmers, could be considered simply partners. *See* Steven Palley, *How to Sue a Decentralized Autonomous Organization*, COINDESK (Mar. 20, 2016, 3:17 PM), http://www.coindesk.com/how-to-sue-a-decentralized-autonomous-organization/. In that event, agency law would presumably add complexity to the interpretative defaults.

[248] Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: The DAO, Exchange Act Release No. 81,207, at 7-8 (suggesting curators faced potential liability); Drew Hinkes, *A Legal Analysis of the DAO Exploit and Possible*

C. Recapitulating the Canons: Quoine and Non-Public Scripts

We have confined our analysis to the definition of transactional scripts we offered in the introduction. Such scripts present a particular set of problems for interpretation. Because the scripts are public, and participants typically are knowingly participating in a specialized form of commerce, the sorts of concerns we might have about knowledge, opportunism, and black-box contracts are, by and large, muted.[249]

How to apply the canons when parties are not relatively sophisticated and cannot view code at the moment of the exchange, presents a distinct and difficult set of questions. In the *Quoine* case, for example, it is not obvious that either side had access to each other's code (certainly nothing in the decision turns on that question explicitly). Where parties cannot access code, it can't communicate meaning, no matter how well drafted its commentary. Thus, it can't be part of the stack that comprised the grist for the bargain. Similarly, as we discussed above, script interpretation might need recalibration where counterparties are ordinary consumers, with as little interest in the hash function's elegant resolution to problems of trust as they have how packets move across the internet itself.

That is, we do not intend to offer answers to all the problems produced by the gap between contract and code. To the extent that new forms of exchange arise that obscure code, permit immutable instructions outside of blockchain, or change the economic incentives for ferreting out error, so too should the law evolve to offer solutions that are pragmatically grounded.

IV. THE FUTURE OF THE CONTRACT STACK

To date transactional scripts haven't delivered revolutionary change to either the world or our small, legal, corner of it. In the big picture, the ecosystem is marginal: billions of dollars of investment in a trillion-dollar world economy. And yet the intellectual footings of the script project are expanding at an astounding rate. Every day, new projects (like Facebook's Libra, or JP Morgan's fiat coin) launch with scripted roots, and the technical community gains experience and competence with each

---

Investor Rights, BITCOIN MAG. (June 21, 2016), https://bitcoinmagazine.com/articles/a-legal-analysis-of-the-dao-exploit-and-possible-investor-rights-1466524659.

[249] In a forthcoming paper, Greg Klass discusses these issues in depth. *See* Greg Klass, *How to Interpret a Vending Machine,* Draft Available from Authors.

failure. We simply have no idea what the future of transactional scripts will look like.[250]

The relationship between this burgeoning, but still highly speculative, ecosystem and law are typically described as antagonistic. Thus, for noted commentator Nick Szabo, the primary virtue of "smart contracts" is that they ostensibly don't need law.[251] For others, scripted deals "will subject the provision to justice to market forces and break the state's monopoly over the court system."[252] This is an ideological call to arms against the civilizing and constraining role that contract jurists have traditionally had in commercial life.

Our approach is different. First, unlike some skeptics, who think that transactional scripts are toys with no real use cases, we believe that they are a potentially valuable new contracting technology. It might be that scripts will reduce back-end transactional costs by reducing the need for transactional lawyering.[253] In certain settings, the benefits from cutting enforcement costs will be worth the costs of upfront specification.[254] By reducing monitoring costs on the margin, transactional scripts may make it more likely for parties to enter into the exchange. Similarly, in regimes where institutional trust is at a nadir and centralized trading repositories are unreliable, scripts can provide significant value.[255]

And yet, this value will have natural limits, defined by a tradeoff between the value of trustless computing and the added costs imposed by the complexity tax, whose scope we are the first to make concrete. Most solutions to the complexity tax require either the intermediation of third parties or consensus protocols

---

[250] Allen, *supra* note 32, at 11 (warning against too heavily discounting the likelihood of successful deployment).

[251] *See, e.g.,* Nick Szabo, @NickSzabo4, Twitter (Oct. 14, 2018, 6:51 PM), https://twitter.com/NickSzabo4/status/1051606530108190720 ("Worrying about whether a smart contract is "legally enforceable" reflects a profound misunderstanding. The main relation of smart Ks to traditional courts is that smart Ks control burden of lawsuit."); Nick Szabo, @NickSzabo4, Twitter (Oct. 14, 2018) (deleted), https://medium.com/cryptolawreview/whys-szabo-afraid-of-smart-contract-critiques-669ef9e63fc0 ("The parties can if they choose write a traditional K to backstop a smart K, although in many situations where a smart contract is useful the exercise would be pointless because the ex ante burden of lawsuit is higher than its added performance incentive benefits.").

[252] Raskin, *supra* note 26, at 335.

[253] *See* Sklaroff, *supra* note 17, at 275-77 (noting that transactional scripts can reduce accounting, due diligence, monitoring, and enforcement costs by cheaply and effectively ensuring data integrity, "minimizing reliance on intermediaries"); Werbach & Cornell, *supra* note 22, at 322, 348 (noting that the ability of transactional scripts to automatically verify, facilitate, and remedy).

[254] *Cf.* Sklaroff, *supra* note 17.

[255] Eric Tai, *Force Majeure and Excuses in Smart Contracts*, 26 EUR. REV. PRIVATE L. 787, 787 (2018).

running across a smaller set of validators.[256] Thus, at least for now, complex organizational solutions will remain within "real" contracts, while particular, discrete, computable aspects can be put on public blockchains. This conclusion fits well with the most sophisticated guidance currently available.[257] But even given that relatively narrow scope, we have shown how and why errors will be difficult to prevent. Legal scholarship about computable contracts simply hasn't fully grappled with the irreducibly buggy nature of coding.

Errors in scripts will result in the parties' outcomes failing to match their goals. In our view, the persistent of error, and the hazards of determining intent, makes recourse to third-party decisionmakers inevitable. Transactional scripts are not, and will not be, self-executing, at least not all the time. At that point, such decisionmakers will be well-advised to look at traditional contract principles to help resolve disputes.

Our approach would bring scripts within the traditional world of contract law through a constitutive legal act: the compilation of a contract stack. This solution is not merely useful for the current form of transactional script. It has relevance for other sorts of digital and algorithmic contracting, whether now contemplated or yet to be imagined. Code can communicate executory intent to contract and can thus be the grist for legal analysis.

But software is imperfect. Code-mediated transactions will therefore sometimes fail to achieve what their promisors intend, even as they are surrounded by communications in "real" languages, intended to be relied on by real people. In such cases, law will confront—and must surmount—two temptations: ignoring the code altogether as a mere instrument of performance, or enforcing it as an exculpatory clause written in ciphered text. We've already seen advocates within the blockchain space make both arguments.

We argue for an alternative regime, which first claims transactional code, commentary and logs as a part of the contractual stack, capable of expressing meaning about the parties' transactional intent. We've also defined a hierarchy of meaning that situates natural language disclosures above code. The canons we've proposed, built on an informed understanding of the coding ecosystem, predictably enforce the parties'

---

[256] Other partial solutions to the complexity tax involve more exotic forms of cryptographic proofs (such as Proof-of-Retrievability for storage) that can reduce redundancy and thus costs. The ultimate efficacy and commercial adoption of such schemes remains an open question. *See* Andrew Miller, Ari Juel, Elaine Shi, Bryan Parno, Jonathan Katz, *Permacoin: Repurposing Bitcoin Work for Data Preservation*, IEEE SECURITY & PRIVACY, 2014 for one such example.

[257] *Cf.* ISDA, *supra* note 167.

reasonable, publicly communicated, intent, and will forbid opportunistic exploitation of subtle errors in coding. These ought to be the law's goals in compiling contracts executed on blockchains, as well as whatever forms of coded exchanges the future delivers to us.