Trust in Smart Contracts is a Process, As Well

Firas Al Khalil, Tom Butler, Leona O'Brien, and Marcello Ceci

Governance, Risk, and Compliance Technology Center
University College Cork
{firas.alkhalil,tbutler,leona.obrien,marcello.ceci}@ucc.ie

Abstract. Distributed ledger technologies are rising in popularity, mainly for the host of financial applications they potentially enable, through smart contracts. Several implementations of distributed ledgers have been proposed, and different languages for the development of smart contracts have been suggested. A great deal of attention is given to the practice of development, i.e. programming, of smart contracts. In this position paper, we argue that more attention should be given to the *"traditional developers"* of contracts, namely the lawyers, and we propose a list of requirements for a human and machine-readable contract authoring language, friendly to lawyers, serving as a common (and a specification) language, for programmers, and the parties to a contract.

1 Introduction

The emergence of distributed ledger technology, due to the development of Bitcoin [22], sparked a lot of interest in different communities: from academia to industry, and from technological and financial circles to philosophical ones [24,27].

The amount of enthusiasm generated around distributed ledgers is indicative of the potentialities that are waiting to be tapped into. What is undeniable, today, is that the financial industry is paying very close attention to cryptocurrencies, especially Bitcoin, but also to other financial applications enabled by distributed ledgers.

Which brings us to *smart contracts*, a concept first envisioned by Nick Szabo [29], as far as 1995 so is claimed, and now believed to be enabled by the advent of distributed ledgers. Several definitions for smart contracts exist, varying in their faithfulness to the original concept, and some of them only adding to the existing confusion surrounding them. We will stand by the original definition of Szabo: *"[s]mart contracts [. . .] facilitate all steps of the contracting process"*; search, negotiation, commitment, performance, and adjudication are all parts of the contracting process he mentioned [28].

Bitcoin, as a platform, is able to model and execute smart contracts, but with a lot of restrictions due to its limited scripting language. This limitation, along with the observation that cryptocurrencies can be viewed as "just another kind of smart contracts", led eventually to the development of Ethereum [31]: a decentralised platform where smart contracts are first-class citizens; the distributed ledger is equipped with a Turing complete programming language that enables

developers to write *"arbitrary"* contracts/code. More recently, platforms built on top of Bitcoin and supporting a Turing complete smart contracts language were developed (e.g. Rootstock [11]), and maybe more interestingly, platforms for smart contracts with non Turing complete languages were also developed, i.e. τ-Chain [6].

It is not a surprise that traditional programmers, if one may call them so, are unable to carry out *"economical thinking"* [10]; indeed, they are also, in our experience, ill-equipped to capture legal or regulatory thinking. The inverse can be said of *subject-matter experts*, i.e. business analysts and lawyers; they are most certainly unable to carry out *"computational thinking"*.

How to carry out the development of smart contracts in large financial institutions, where, traditionally, contracts are drafted by subject-matter experts? More importantly, how can we reason on the legality of developed contracts? Either manually by a lawyer, or automatically using a tool for compliance checking? A failure to answer these questions inevitably contributes to the scepticism of the financial industry –which has been put under the microscope by regulators since 2008– about the future of smart contracts, and the industry's reluctance in adopting it.

In this position paper, we argue that trust in smart contracts, is *also* a process; a bridge is needed to connect both sides of the abyss.

The rest of this paper is organised as follows: Section 2 shows how diverse is the scene of distributed ledger technologies; Section 3 shows how irreconcilable are the languages of programmers and subject-matter experts; Section 4 develops our views on how can we build a bridge that enables trust, from an institutional perspective, in smart contracts; we finally conclude in Section 5.

2 On Distributed Ledger Technologies

The introduction of Bitcoin by Satoshi Nakamoto [22] polarised the actors in the financial industry since the beginning: some were extremely enthusiastic about it, to the point where they claimed that Bitcoin is the *"next big thing"*, and others were extremely sceptical about it.

The innovation of Bitcoin is not limited to the currency; the idea of a shared ledger itself proved to be very powerful and sprung many platforms rivalling or even complementing Bitcoin. The interested reader can refer to Tschorsch and Scheuermann [30] for an excellent technical survey on distributed ledger technologies. Moreover, a quick look at the currently available platforms inspired by Bitcoin, gives a good idea on the rising popularity of the technology: for instance, `coinmarketcap.com`, a site that tracks market capitalisation of different cryptocurrencies, lists 719 platforms.

Since its inception, Bitcoin provided a stack-based scripting language that allowed developers to define the conditions to spend Bitcoins (e.g. requiring multiple signatures), which revived the vision of smart contracts. However, this scripting language is purposefully not Turing complete, which ultimately means that it is limited in expressivity. In the following, we will take a look at four

different platforms that are meant to overcome Bitcoin's scripting limitations, illustrating the different technical choices one can make, regarding the development of smart contracts.

The first platform we are going to look at, which is currently almost synonymous to "smart contracts", is Ethereum [31]. Ethereum was proposed as a distributed platform independent of –yet very similar to– Bitcoin. To create distributed trust-less consensus and solve the double-spending problem, Ethereum uses proof-of-work, just like Bitcoin, however, it provides the Ethereum Virtual Machine (`EVM`) that runs a Turing complete stack based language, which opens the doors to a hypothetically unlimited number of applications. Developers are not forced to use the `EVM`'s opcode to write smart contracts. Indeed, they can use `Solidity` or `Serpent`, which are high-level programming languages, similar to `javascript` or `python`, respectively, that can compile to `EVM` byte code.

The second platform we are going to look at is Nxt [1], one of the earliest smart contract platforms. Unlike Bitcoin and Ethereum, Nxt uses proof-of-stake to achieve consensus and solve the double-spending problem. Moreover, Nxt does not provide a scripting language to smart contract developers; instead, it provides a RESTful API exposing a set of primitive operations (like spending, storing strings, sending messages, etc.) that developers can invoke.

The third platform we will consider is Rootstock [11]. Unlike Ethereum, and Nxt, Rootstock was developed to complement Bitcoin (as a *sidechain* [12]) and provides its own Turing complete virtual machine (the `RVM`) to enable smart contracts.

The fourth and final platform we will examine is $\tau-$Chain [6]. The authors of this platform argue that Turing completeness is not necessary for distributed ledgers, because with Turing completeness comes undecidability, i.e. smart contracts can go in an infinite loop and the network will never be able to predict this behaviour. Indeed, Ethereum overcomes the problem of undecidability by forcing the caller of the smart contract to provide *gas* with the transaction (bought with *ether*, Ethereum's own cryptocurrency); every instruction on the `EVM` consumes a predefined amount of gas, and they are non-refundable, i.e. if the gas is totally consumed and the smart contract didn't finish execution, the gas is never returned to the caller.

However, Asor [6] propose the use of an ontology [2] of rules, along with a reasoner, to enable computations on the network. Authors of smart contracts would write them in a totally functional programming language, like `Idris` [7], that will be ultimately translated into the ontology. This approach will not only make computations decidable, but it also allows the assertion of properties of smart contracts that were impossible with Turing complete languages, e.g. if the contract connects to the Internet or not, or if the contract fulfills some interfaces/requirements/etc.

The interested reader can refer to the survey written by Seijas et al. [25] for more information on scripting languages for distributed ledgers. The aforementioned platforms illustrate some of the variations that exist in distributed

[1] `https://nxt.org/`

ledger technology's ecosystem. These platforms can differ not only in the tooling and the language they expose for smart contract development, but also in the paradigms that govern them. The development of smart contracts thus requires a deep and serious understanding of the target platform. In the following section, we will examine what hinders a fast adoption of such an enabling technology by the financial industry.

3 Staring Into the Abyss

A close inspection of the literature shows that effort is being put in helping developers author smart contracts, by either developing tools, or creating abstractions.

Recently, Delmolino et al. [10] reported on their experience in teaching smart contract programming, using Ethereum, to undergraduate students at the University of Maryland. The authors concluded that smart contract programming *requires an "economic thinking" perspective that traditional programmers may not have acquired*. Indeed, students repeatedly made logical errors that ultimately lead to money leaks, failed to use cryptographic primitives to secure the contracts from attackers, failed to account for the incentives of contract callers, and even made mistakes directly related to Ethereum.

This observation lead to the development of a Masters thesis by Pettersson and Edström [23], and their objective was to help said programmers to develop safer smart contracts. Their aim is to prevent 3 kinds of mistakes contract developers fall in: unexpected states, failure to use cryptography, and overflowing the `EVM`'s stack. They propose to use of a functional programming language, namely `Idris`. They developed a code generator that transforms code produced by an `Idris` compiler to `Serpent` code, which can be subsequently compiled into `EVM` bytecode.

In a different, yet related work, Luu et al. [21] noted that a class of security-related bugs in smart contracts are due to the gaps in the understanding of the distributed semantics of the underlying platform.

Another interesting work is that of Florian et al. [20], who propose the use of logic-based smart contracts. They showed that this approach can complement smart contracts written in procedural code, in terms of contract negotiation, formation, storage/notarizing, enforcement, monitoring and activities related to dispute resolution.

In a different take, García-Bañuelos et al. [16] showed how the business process language `BPMN` can be mapped into executable smart contracts on the Ethereum. This development lead Hull et al. [19] to propose a *Business Collaboration Language* (`BCL`) for shared ledgers. Indeed, this `BCL` can be thought of as the equivalent of `SQL` for relational databases, targeting shared ledgers, regardless of implementation-specific details.

As far as we know, the only works that consider the issue of authoring smart contracts from the subject-matter expert's perspective are those proposed by Frantz and Nowostawski [14] and Clack et al. [9].

Frantz and Nowostawski [14] propose a semi-automated method for the translation of human readable contracts to smart contracts on Ethereum. The authors develop a domain specific language for contract modelling, where statements are rules expressed in English, and that translates into `Solidity`. However, this solution is very tied to Ethereum, and it is not clear how extensible or adaptable it is. Additionally, it doesn't cover the legal language that a lawyer would be accustomed to.

Clack et al. [9] rightly identify two semantics of contracts:

Operational semantics: concerned with the execution of the contract on a specific platform

Denotational semantics: that captures the *"legal meaning"* of the contract, as understood by a lawyer.

The authors envision the use of smart contract templates, based on the idea of Ricardian Contracts [17,18]. A Ricardian Contract is a digitlly signed triple $\langle P, C, M \rangle$, where P is the legal prose, capturing denotational semantics, C is the platform specific code expressing operational semantics, and M is a map (key-value pairs) of parameters used in P and C.

While the use of smart contract templates, based on Ricardian Contracts, looks like a move towards the right direction, we argue that prose should not be tied to code:

- While the semantics of legal language can be expressed as a set of deontic defeasible rules, the code is rather procedural. The order of the instructions in the procedure does not reflect the natural order of the contract clauses expressed in natural language [20].
- The life-cycle of legal prose is independent from the life-cycle of the code. For example, a lawyer might describe the terms of a contract in prose and never come back to it, while a developer will –most likely– iterate through different implementations (e.g. bug fixes).
- There is not a single smart contract platform, which ultimately means that different parameters (key-value pairs of M) will be needed for different platforms. For example, several works (e.g. [32,3,1]) describe data feed systems that enable smart contracts to consume data feeds from outside the distributed ledger (e.g. a stock market index); while the notion of an external feed might be familiar to a lawyer, its technical details, thus the choices related to the adoption of one method over another, and eventually the parametrisation is definitely out of her/his reach and/or interest.

In the following section, we will identify the key issues, as we see them, regarding the adoption of smart contracts, and how we envision to solve them.

4 Trusting Smart Contracts

In Section 2 we tried to show, through a non-exhaustive list of examples, how distributed ledgers can differ on a deep technical level, which requires a very

intimate technological knowledge by the *implementer* of the smart contract. Afterwards, we showed, in Section 3, how current effort is mostly focused on developing technical tools and infrastructure aimed at facilitating the technical implementation of smart contracts. However, there is a major lacuna in all this: that is the translation, or mapping, of the contract's denotational semantics to its operational semantics.

We share the view of Clack et al. [9] on the separation between operational and denotational semantics of contracts. In fact, we argue that trust in smart contracts can only stem from the ability of lawyers in financial institutions to understand, express, and ultimately validate the denotational semantics of a contract. However, we disagree on the assumption they make on the languages expressing these semantics, i.e. any assumption on the correspondence between a *"legal language"* and the *"technical language"* cannot be achieved, as the lawyer cannot predict the behaviour of the code.

What is missing from all of the described work, is the realisation that the involvement of a lawyer, especially in the heavily regulated financial industry, in the authoring of contracts, not only smart contracts, is paramount, for her/his knowledge on the regulation governing said contracts dictates the denotational semantics. A lawyers' knowledge of the explicit and implicit rights and obligations, counterparties, stakeholders, schedules and penalties, and regulations governing a financial contract needs to be represented.

Indeed, the financial crisis of 2008 was in part caused by the sub-prime lending practice that encouraged high credit risk borrowers to take on mortgages at high interest rates that they had little ability to repay. These debts were pooled together and engineered to be offered as low risk asset-backed securities. These were heavily traded because of the *perceived* low risk while providing high returns. The housing market in the US slumped setting off a chain reaction that ultimately meant the mortgage-backed securities became worthless having direct effect on the capital of the major global banks. Funding dried up and more importantly, the trust that keeps the financial system performing dissolved. As a result, regulation in the financial industry has grown exponentially.

There are two scenarios where the lawyer's involvement is unavoidable:

- When the contract is partly fulfilled through code, because the lawyer can only validate its textual version [20], i.e. the prose.
- When assessing the compliance of the contract with regulations, from the point of view of both the legal requirements introduced by the regulation (e.g. on financial activities, anti-money laundering, or consumer protection), and of the effects that these regulations automatically bind to the contract (*naturalia negotii* [15]).

Therefore, we think that proper authoring of smart contracts should involve two main agents: the lawyer and the developer. The interaction between both agents should be governed by a common language. The lawyer authors and consumes contracts written in that language, while the developer uses it as a specification guiding her/his implementation. This common language should have the following properties:

- It should not alienate the lawyer, i.e. it should be as close as possible to the language of contracts s/he is used to.
- It should be expressive enough to allow the authoring of smart and *"not-so-smart"* contracts.
- It should be a Controlled Natural Language (CNL) with an unambiguous grammar. The CNL should be mappable to a logical formalism which will facilitate compliance checking with existing regulations.
- The concepts and actions described in the contract (i.e. the vocabulary) along the clauses of the contract (i.e. the rules) should be shareable across the network, which is important for both *discoverability* and *negotiation* –two defining aspects of smart contracts– by human and autonomous agents.
- It should be able to represent the actions coded in the smart contract [9], the duties and powers arising from the contract [14], and the meta-rules governing it (e.g. regulation on financial activities, Anti-Money Laundering or consumer protection).

In our previous work [8] we describe Mercury, a language to capture regulations for the purpose of compliance checking, alongside a methodology [4] to capture legal knowledge and translate it to `OWL` [5]. Mercury is based on the Semantics for Business Vocabulary and Business Rules [26] (`SBVR`), but the language of smart contracts should not forcibly be based on `SBVR`, as long as it can be mapped to a logical formalism, e.g. `OWL`, where reasoning on compliance is feasible.

In a recently published technical report, English et al. [13] investigated how distributed ledger technologies and the Semantic Web can affect one another. Indeed, the blockchain can provide secure resource identifiers (by ensuring authenticity, human-readability, and decentralisation), and ontologies can provide a unified way to understand blockchain concepts between humans, and exposing blockchain data according to an ontology enables the interlinking with other linked data and to perform reasoning.

Our proposal improves transparency, which is one of the major luring qualities of distributed ledgers, and a determining factor of the trust-less trust in the network. But doubt rises when it comes to the trust in the fact that the contract, as written by the lawyer, was correctly translated into code, i.e. the trust in the fact operational semantics faithfully represent denotational semantics. One may argue that this trust can be guaranteed if there is a mechanism $\mathcal{G}$ that automatically generates code from prose and/or a mechanism $\mathcal{C}$, potentially the inverse of $\mathcal{G}$, that proves the correspondence of the code to the prose, but a closer inspections shows that:

1. There is evidence from the literature that $\mathcal{G}$ and $\mathcal{C}$ can exist, especially from [20] and τ-Chain [6]. Indeed, if the vision of τ-Chain is possible, then there is an opportunity to go directly from denotational to operational semantics using our approach, but this *may* imply the restriction of said trust to one specific distributed ledger technology.

2. It is not really clear, at least for us, if $\mathcal{G}$ and $\mathcal{C}$ exist for shared ledgers that use stack-based languages. This is an open question that deserves closer attention, and can have one of two clear answers:
   (a) It is possible, or practically feasible, which is great news for everyone, or
   (b) It is impossible, or practically infeasible. Then it is only reasonable to ask: *is the existence of $\mathcal{G}$ and $\mathcal{C}$ a prerequisite for the establishment of said trust?* We conjecture that it is not, for two reasons:
      i. The implementation processes of existing financial contracts in the form of software is already opaque, especially to the consumer, and our proposed approach would only facilitate transparency.
      ii. Trust can be gained through the establishment of reputation: the better you are in effectively transforming your specification to code, the more reputable you are; the more reputable you are, the more trustworthy you are perceived to be.

5 Conclusion

In this position paper, we expressed our point of view on how trust in smart contrast, from a financial institution's point of view, can be enabled. It is true that cryptographic guarantees are enablers of, and integral to, trust in distributed ledger technology, but we argue that another kind of trust is needed; one that is established by a process involving lawyers.

We showed how distributed ledger technologies can vary on a deep technical level, which led to the development of tools and abstractions to help developers in programming smart contracts. These developments are essential for the technological ecosystem, but we show how most of the existing work do not take into account compliance with existing (and ever growing) regulations.

To that end, we set a list of criteria for a language necessary for the development of contracts, executed on the ledger, or not, that is close to the legal prose, transparent, and rooted in formal logic. We also identify a key research challenge, which is the ability to translate the aforementioned language to executable code.

References

1. Oraclize: "The provably honest oracle service". `http://www.oraclize.it/`, accessed: 2017-01-30
2. OWL 2 Web Ontology Language Document Overview (Second Edition). `https://www.w3.org/TR/2012/REC-owl2-overview-20121211/`, accessed: 2017-01-30
3. PriceFeed smart contract. `http://feed.ether.camp/`, accessed: 2017-01-30
4. Abi-Lahoud, E., O'Brien, L., Butler, T.: On the Road to Regulatory Ontologies, pp. 188–201. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), `http://dx.doi.org/10.1007/978-3-662-45960-7_14`
5. Al Khalil, F., Ceci, M., Yapa, K., O'Brien, L.: SBVR to OWL 2 Mapping in the Domain of Legal Rules, pp. 258–266. Springer International Publishing (2016), `http://dx.doi.org/10.1007/978-3-319-42019-6_17`
6. Asor, O.: About Tau-Chain. ArXiv e-prints (Feb 2015)

7. BRADY, E.: Idris, a general-purpose dependently typed programming language: Design and implementation. Journal of Functional Programming 23(5), 552–593 (Sep 2013)
8. Ceci, M., Al Khalil, F., O'Brien, L.: Making Sense of Regulations with SBVR. In: RuleML 2016 Challenge, Doctoral Consortium and Industry Track hosted by the 10th International Web Rule Symposium (RuleML 2016) (2016)
9. Clack, C.D., Bakshi, V.A., Braine, L.: Smart Contract Templates: essential requirements and design options. ArXiv e-prints (Dec 2016)
10. Delmolino, K., Arnett, M., Kosba, A.E., Miller, A., Shi, E.: Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In: Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers. pp. 79–94 (2016), http://dx.doi.org/10.1007/978-3-662-53357-4_6
11. Demian Lerner, S.: Rootstock. bitcoin powered smart contracts. white paper. (Nov 2015), https://uploads.strikinglycdn.com/files/90847694-70f0-4668-ba7f-dd0c6b0b00a1/RootstockWhitePaperv9-Overview.pdf
12. Demian Lerner, S.: Drivechains, sidechains, and 2-way hybrid peg designs (Jan 2016), https://uploads.strikinglycdn.com/files/27311e59-0832-49b5-ab0e-2b0a73899561/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf
13. English, M., Auer, S., Domingue, J.: Block chain technologies & the semantic web: A framework for symbiotic development. Tech. rep., Technical report, University of Bonn, Germany (2016)
14. Frantz, C.K., Nowostawski, M.: From institutions to code: Towards automated generation of smart contracts. In: 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W). pp. 210–215 (Sept 2016)
15. Frignani, A.: Some Basic Differences between the Common Law and the Civil Law Approach. http://www.jus.unitn.it/CARDOZO/Review/Comparative/Frignani-1997/Washingt.htm (1996), accessed: 2017-02-02
16. García-Bañuelos, L., Ponomarev, A., Dumas, M., Weber, I.: Optimized Execution of Business Processes on Blockchain. ArXiv e-prints (Dec 2016)
17. Grigg, I.: The ricardian contract. In: Proceedings. First IEEE International Workshop on Electronic Contracting, 2004. pp. 25–31 (July 2004)
18. Grigg, I.: On the intersection of Ricardian and Smart Contracts. http://iang.org/papers/intersection_ricardian_smart.html (Feb 2017), accessed: 2017-01-30
19. Hull, R., Batra, V.S., Chen, Y.M., Deutsch, A., Heath III, F.F.T., Vianu, V.: Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes, pp. 18–36. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-46295-0_2
20. Idelberger, F., Governatori, G., Riveret, R., Sartor, G.: Evaluation of Logic-Based Smart Contracts for Blockchain Systems, pp. 167–183. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-42019-6_11
21. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 254–269. CCS '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2976749.2978309
22. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)

23. Pettersson, J., Edström, R.: Safer smart contracts through type-driven development. Ph.D. thesis, Master's thesis, Dept. of CS&E, Chalmers University of Technology & University of Gothenburg, Sweden (2015)
24. Reijers, W., O'Brolcháin, F., Haynes, P.: Governance in blockchain technologies & social contract theories. Ledger 1(0), 134–151 (2016), http://www.ledgerjournal.org/ojs/index.php/ledger/article/view/62
25. Seijas, P.L., Thompson, S., McAdams, D.: Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156 (2016), http://eprint.iacr.org/2016/1156
26. Semantics of Business Vocabulary and Business Rules (SBVR) Version 1.3. http://www.omg.org/spec/SBVR/1.3/PDF (May 2015), http://www.omg.org/spec/SBVR/1.3/PDF
27. Swan, M.: Blockchain Temporality: Smart Contract Time Specifiability with Blocktime, pp. 184–196. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-42019-6_12
28. Szabo, N.: Formalizing and Securing Relationships on Public Networks. https://web.archive.org/web/20050217172626/http://www.firstmonday.dk/ISSUES/issue2_9/szabo/index.html (1997), accessed: 2017-01-25
29. Szabo, N.: The Idea of Smart Contracts. https://web.archive.org/web/20160831070942/http://szabo.best.vwh.net/smart_contracts_idea.html (1997), accessed: 2017-01-25
30. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys and Tutorials 18(3), 2084–2123 (2016), http://dx.doi.org/10.1109/COMST.2016.2535718
31. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151 (2014)
32. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town crier: An authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 270–282. CCS '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2976749.2978326