Reliable Smart Contracts

Gordon J. Pace[1], César Sánchez[2], and Gerardo Schneider[3]

[1] University of Malta, Malta
gordon.pace@um.edu.mt
[2] IMDEA Software Institute, Madrid, Spain
cesar.sanchez@imdea.org
[3] University of Gothenburg, Sweden
gerardo@cse.gu.se

Abstract. The rise of smart contracts executed on blockchain and other distributed ledger technologies enabled trustless yet decentralised computation. Various applications take advantage of this computational model, including enforced financial contracts, self-sovereign identity and voting. But smart contracts are nothing but software running on a blockchain, with risks of malfunction due to bugs in the code. Compared to traditional systems, there is an additional risk in that erroneous computation or transactions triggered by a smart contract cannot be easily rolled back due to the immutability of the underlying execution model. This ISoLA track brings together a number of experts in the field of smart contract reliability and verification to discuss the state-of-the-art in smart contract dependability and discuss research challenges and future directions.

1 Blockchains and Smart-Contracts

Blockchains and Distributed Ledger Technologies (DLTs) are essentially a distributed ledger or database, running on multiple devices. The key elements which have generated the hype of this technology are (1) that the consistency of the data stored and the guarantees on updates are performed in a fully decentralised manner, and (2) to ensure immutability of information written. This provided a way of implementing the secure management of digital asset storage in a decentralised manner, enabling the implementation of trustless cryptocurrencies. In the paper which set all this in motion [4], Satoshi Nakamoto outlined how blockchain can be implemented and used to record ownership of cryptocurrency, and acted also as the launch of the Bitcoin blockchain which keeps track of ownership of Bitcoins.[4]

It was soon realised that such a ledger could easily be used to keep track of ownership of any digital asset, recorded on a particular blockchain or as a representation (or evidence) of ownership of any object, physical or otherwise, stored off the blockchain (e.g. music, art, intellectual property, votes). Also, such a trustless network can be used to script performance of agreements between parties on the blockchain. The immutable nature of the blockchain ensures that

[4] Confusingly, the term is used both for the network and the cryptocurrency.

such agreements are themselves permanent and immutable, thus safeguarding the rights of the parties involved. In this manner, blockchain has the potential to change in a fundamental way financial services and more generally other applications, improving transparency and regulation.

The Ethereum [1] blockchain was the first widespread blockchain system to implement such smart contracts, allowing not only for transactions consisting of transfer of Ether (the native cryptocurrency) between parties, but also for transactions to deploy and interact with smart contracts. The underlying implementation ensured that the smart contracts are irrevocably written on the blockchain, and their invocations are executed autonomously and securely on the blockchain. On such a public blockchain, smart contracts are openly stored on the blockchain (i.e. they can be read and used by anyone).

Since smart contracts would thus have to be executed and verified automatically by the nodes of the underlying blockchain network, they have to be expressed in a formalism which has a deterministic operationalisation. Many DLTs enabling smart contract execution, including Ethereum, opted for supporting a full Turing-complete instruction set. The execution of smart contract invocations is performed, and the results recorded, on the blockchain network by nodes on the network acting as "miners", who are rewarded (in cryptocurrency) in return. The corresponding instructions in the smart contract may perform normal computation logic but also manipulate the local book-keeping of data (including cryptocurrency). In addition, smart contracts may act as other parties in that they can own and can transfer cryptovalues.

A smart contract can be seen as an embodiment of an agreement between different parties in order to automate the regulated exchange of value and information over the internet. Their promise is that of reducing costs of contracting, and of enforcing contractual agreements (*"robust* [...] *against sophisticated, incentive compatible breach"* [6]), and making payments, while at the same time ensuring trust and compliance, all in the absence of a central authority.

The challenge, however, lies in the fact that smart contracts, just like any other program, can have bugs and vulnerabilities. The agreement enforced can thus inadvertently not match the intended one, and may result in losses for the parties involved. In the literature, one can find reports on various bugs which resulted in the equivalent of millions of euros [2, 3, 5]. Reliability and correctness of smart contracts is, effectively, a question of reliability and correctness of programs, one of the Holy Grails of computer science. However, one may argue that the need for reliability is even more acute in smart contracts, where code is immutable[5] and critical in that it typically manipulates cryptocurrency. Writing a smart contract carries risk akin to a payment service provider implementing their software in hardware circuits which cannot (or are too expensive to) be updated. On the other hand, since in this setting the computation is performed in small, bite-sized chunks of code, smart contracts are more amenable to ver-

[5] It shares, in a sense, the permanent nature of algorithms implemented as hardware circuits. Hardware led the efforts for automated verification in the 90s due to the high cost of potential bugs.

ification than traditional systems which may run into millions of lines of code. This calls for better programming languages for smart contracts with stronger security and privacy guarantees, achieved either through improved underlying DLT platforms, the design of programming languages better suited for smart contract programming or through verification techniques for smart contracts.

In the track we have collected new results and discussions related to:

- Research on different languages for smart contracts including their expressivity and reasoning methods.
- Research on the use of formal methods for specifying, validating and verifying smart contracts.
- Surveys and state-of-knowledge about security and privacy issues related to smart contract technologies.
- New applications based on smart contracts.
- Description of challenges and research directions to future development for better smart contracts.

2 Summary of Selected Articles

In this section we briefly summarise the articles invited to the ISoLA'20[6] track on "Reliable Smart Contracts: State-of-the-art, Applications, Challenges and Future Directions"[7], appearing in this volume.

- **Functional Verification of Smart Contracts via Strong Data Integrity**, by Wolfgang Ahrendt and Richard Bubel, presents an invariant-based static analysis approach and tool implementation for Solidity smart contracts. The approach is based on theorem proving and symbolic execution and was built upon the tool KeY. Unlike much of the other work being carried out in static analysis for smart contracts, the approach presented by the authors focuses on the business logic, the expected functionality of the smart contract in question, although it also addresses standard problems, in particular reentrancy.
- **Bitcoin Covenants Unchained**, by Massimo Bartoletti, Stefano Lande and Roberto Zunino, proposes an extension of the Bitcoin script language with "covenants", a set of script operators used to constrain how funds can be used by the redeeming transactions. Covenants can be recursive, extending the expressiveness of Bitcoin contracts. A formal model for covenants is given to show the expressiveness of the extension. Finally, the paper discuss how covenants could be integrated in a high-level language, in particular it is hinted how this could be done in the BitML language.
- **Specifying Framing Conditions for Smart Contracts**, by Bernhard Beckert and Jonas Schiffl, proposes a formalism to enrich smart contract specifications with frame conditions in order to specify what a smart contract

[6] `http://isola-conference.org/isola2020`
[7] `http://www.cs.um.edu.mt/gordon.pace/Workshops/RSC2020`

function cannot (and will not) do. The approach is based on the concept of dynamic frames. It proposes languages for specifying frame conditions for both Ethereum and Hyperledger Fabric.

- **Making Tezos Smart Contracts More Reliable with Coq**, by Bruno Bernardo, Raphaël Cauderlier, Guillaume Claret, Arvid Jakobsson, Basile Pesin and Julien Tesson, presents the Mi-Cho-Coq framework, a Coq library defining formal semantics of Michelson (the Tezos language to write smart contracts) as well as an interpreter, a simple optimiser and a weakest-precondition calculus to reason about Michelson smart contracts. The paper also introduces Albert, an intermediate language with a compiler written in Coq that targets Mi-Cho-Coq, illustrating how Mi-Cho-Coq can be used as a compilation target to generate correct code.
- **UTxO- vs Account-Based Smart Contract Blockchain programming paradigms**, by Lars Brünjes and Murdoch J. Gabbay, formalises and proves desirable properties of Cardano, most importantly that the effects of a transaction do not depend on the other transactions in the same block. A formalisation of Ethereum is given and contrasted in terms of smart contract language paradigm — Solidity and Plutus, vulnerability to DDoS attacks and in terms of propensity to certain errors (e.g. due to the effect of Ethereum transactions can change depending on the behaviour of other transactions in the same block, but not in Cardano).
- **Native Custom Tokens in the Extended UTXO Model**, by Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova and Philip Wadler, presents an extension of user-defined tokens UTXO (present in blockchains like Bitcoin) and how this exstension can be used to define some smart-contract behavior. Extended UTXO allow to encode, pass and validate arbitrary data across multiple transactions which is sophisticated enough to validate runs with so-called Constraint Emitting machines (Mealy machine with data).
- **UTXO$_{\text{ma}}$: UTXO with Multi-Asset Support**, by Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova, Philip Wadler and Joachim Zahnentferner, explores a design for the creation of user-defined tokens based on UTXO ledgers. Unlike platforms such as Ethereum, which allow for the implementation of such tokens using general scripting capabilities which comes with well-known risks, the authors propose an extension to the UTXO model to manage an unbounded number of user-defined, native tokens using a simple domain-specific language with bounded computational expressiveness. They formalise the model and its semantics using the Agda proof assistant.
- **Towards Configurable and Efficient Runtime Verification of Blockchain based Smart Contracts at the Virtual Machine Level**, by Joshua Ellul, studies the problem of alleviating the overhead imposed by monitoring the execution of smart-contracts, using low-level (virtual machine) infrastructure. The paper presents modifications of the VM level that permit enabling and disabling state variable monitoring and syscall mon-

itoring dynamically, and compares the different approaches empirically, in terms of the reduction in the monitoring overhead obtained.
- **Compiling Quantitative Type Theory to Michelson for Compile-Time Verification & Run-time Efficiency in Juvix**, by Christopher Goes, uses quantitative type theory (QTT)—a typed lambda calculus equipped with resources using dependent types—to construct a theoretical basis of the core language within Juvix. The paper illustrate how this basis can be used to efficiently compile Juvix into efficient Michelson code, the execution language of the Tezos Blockchain ecosystem.
- **Efficient Static Analysis of Marlowe Contracts**, by Pablo Lamela Seijas, David Smith and Simon Thompson, discusses the authors' experience in the implementation and optimisation of static analysis for the smart contract language Marlowe which is designed specifically for self-enforcing financial smart-contracts. In particular, the authors look at the use of SMT solvers for the verification of properties written in this language.
- **Accurate Smart Contract Verification through Direct Modelling**, by Matteo Marescotti, Rodrigo Otoni, Leonardo Alt, Patrick Eugster, Antti E. J. Hyvärinen and Natasha Sharygina, presents a formal analysis engine used in the Solidity compiler. The verification is performed using the logic of constrained Horn clauses. The approach, evaluated on a set of smart contracts deployed in the Ethereum platform, is able to prove correctness and discover bugs in number of such contracts.
- **Smart Derivatives: On-chain Forwards for Digital Assets** by Alfonso D.D.M. Rius and Eamonn Gashier, introduces a framework to facilitate the development of on-chain forwards (and futures), based on smart contracts. The paper builds upon, and extends, previous work by the authors on on-chain options. The paper makes a connection between computer smart contracts and "real world" (financial) contracts.
- **The Good, the Bad and the Ugly: Pitfalls and Best Practices in Automated Sound Static Analysis of Ethereum Smart Contracts**, by Clara Schneidewind, Markus Scherer and Matteo Maffei, focuses on static analysis of smart contracts, particularly for Ethereum. It discusses the challenges of providing sound verification techniques, highlighting unsoundness of existing tools through concrete examples. The paper then proceeds to give details about eThor, a tool developed by the authors, and outline its use for addressing the reentrancy problem.

References

1. Ethereum. `https://www.ethereum.org`.
2. A. Hern. $300m in cryptocurrency accidentally lost forever due to bug. Appeared at The Guardian `https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether`, Nov 2017.
3. Mix. Ethereum bug causes integer overflow in numerous erc20 smart contracts (update). Appeared at HardFork `https://thenextweb.com/hardfork/2018/04/25/ethereum-smart-contract-integer-overflow/`, 2018.

4. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. White Paper `https://bitcoin.org/bitcoin.pdf`, 2009.
5. Haseeb Qureshi. A hacker stole $31M of Ethereum — how it happened, and what it means for Ethereum. Appeared at FreeCodeCamp `https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce`, 2017.
6. Nick Szabo. Smart contracts: Building blocks for digital markets. *Extropy*, (16), 1996.