Report from Dagstuhl Seminar 21431

Rigorous Methods for Smart Contracts

Edited by

Nikolaj S. Bjørner[1], Maria Christakis[2], Matteo Maffei[3], and Grigore Rosu[4]

1 **Microsoft – Redmond, US,** nbjorner@microsoft.com
2 **MPI-SWS – Kaiserslautern, DE,** maria@mpi-sws.org
3 **TU Wien, AT,** matteo.maffei@tuwien.ac.at
4 **University of Illinois – Urbana-Champaign, US,** grosu@illinois.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 21431 "Rigorous Methods for Smart Contracts". Blockchain technologies have emerged as an exciting field for both researchers and practitioners focusing on formal guarantees for software. It is arguably a "once in a lifetime" opportunity for rigorous methods to be integrated in audit processes for parties deploying smart contracts, whether for fund raising, securities trading, or supply-chain management.

Smart contracts are programs managing cryptocurrency accounts on a blockchain. Research in the area of smart contracts includes a fascinating combination of formal methods, programming-language semantics, and cryptography. First, there is vibrant development of *verification and program-analysis techniques* that check the correctness of smart-contract code. Second, there are emerging designs of *programming languages and methodologies* for writing smart contracts such that they are more robust by construction or more amenable to analysis and verification. Programming-language abstraction layers expose low-level cryptographic primitives enabling developers to design high-level *cryptographic protocols*. *Automated-reasoning mechanisms* present a common underlying enabler; and the specific needs of the smart-contract world offer new challenges.

This workshop brought together stakeholders in the aforementioned areas related to advancing reliable smart-contract technologies.

Seminar October 24–29, 2021 – http://www.dagstuhl.de/21431
2012 ACM Subject Classification Security and privacy → Logic and verification; Software and its engineering → Formal language definitions; Software and its engineering → Software verification and validation
Keywords and phrases automated reasoning, cryptographic protocols, program verification, programming languages, smart contracts
Digital Object Identifier 10.4230/DagRep.11.9.80
Edited in cooperation with Schindler, Tanja

1 Executive Summary

Nikolaj S. Bjørner (Microsoft – Redmond, US)

The seminar attracted 22 on-site and approximately as many off-site participants. The hybrid mode presented an opportunity for collaborators, particularly students, of invitees to participate remotely and contribute to the discussions. Remote participation spanned all

time zones which attested to their involvement. The on-site participants had the benefit of extended interactions and relation building so crucial for advancing scientific activities.

The technical program was organized around first day of tutorial presentations on the main topics covered by the seminar. These topics were *static analysis techniques*, *program verification methods*, *protocol design for decentralized ledgers*, and *semantic-based tools*.

The following days provided for in-depth sessions around these topics. Static analysis techniques spanned using Horn clause solvers, Datalog engines, and abstract interpretation frameworks in a mixture of academic and industrial settings. Program verification techniques, likewise, were pursued both by academic and industry participants. The seminar offered an excellent forum for the scientific and commercial community around smart contracts to exchange experiences and develop ideas.

For the social program, we hiked for two hours during a beautiful October afternoon to Landgasthof Paulus & Der Laden for a delightful dinner.

2 Table of Contents

3 Overview of Talks

3.1 The GASOL project: a GAS Optimization tooLkit

Elvira Albert (Complutense University of Madrid, ES) and Albert Rubio (Complutense University of Madrid, ES)

Joint work of Elvira Albert, Pablo Gordillo, Alejandro Hernández-Cerezo, Albert Rubio

Super-optimization is a compilation technique that searches for the optimal sequence of instructions semantically equivalent to a given (loop-free) initial sequence. This talks overviews our approach for super-optimization of smart contracts based on Max-SMT which is split into two main phases:(i) the extraction of a functional specification from the basic blocks of the smart contract, which is simplified using rules that capture the semantics of the arithmetic, bit-wise, relational operations, etc. and(ii) the synthesis of optimized blocks which, by means of an efficient Max-SMT encoding, finds the bytecode blocks with minimal cost (according to the selected optimization criteria) and whose functional specification is equal (modulo commutativity) to the extracted one. Our experiments on randomly selected real contracts achieve important gains in gas and in bytes-size over code already optimized by solc.

References

1 Elvira Albert, Pablo Gordillo, Albert Rubio, Maria Anna Schett: Synthesis of Super-Optimized Smart Contracts Using Max-SMT. CAV (1) 2020: 177-200

3.2 Formally Verifying Ethereum Smart Contracts by Overwhelming Horn Solvers

Leonardo Alt (Ethereum – Berlin, DE)

Ethereum smart contracts hold billions of USD, have (usually) immutable logic, and are (also usually) open source. Therefore, ensuring that the programs are bug free is essential for this ecosystem. Formal verification, particularly, has seen a successful application in smart contracts also due to their rather small complexity compare to other types of systems, since the contract size is limited and complex code often implies higher computation costs. In this work we present a model checker for Solidity smart contracts based on Constrained Horn Clauses [1]. The Solidity programs are encoded as systems of Horn clauses where verifying a safety clause consists of Horn satisfiability. We show how the encoding is performed following [2], and the special behaviors from smart contracts that lead to the specific Horn encoding for these problems from [1]. We also show experiments on a small scale, focusing on specific features, as well as large real-world instances, demonstrating how properties that are part of large systems can also be solved automatically. Finally, we present data comparing how the two backend Horn solvers used by the tool, Spacer and Eldarica, compare in the different instances that are part of the experiments.

References

1 Matteo Marescotti, Rodrigo Otoni, Leonardo Alt, Patrick Eugster, Antti E. J. Hyvärinen, Natasha Sharygina: Accurate Smart Contract Verification Through Direct Modelling. ISoLA (3) 2020: 178-194

2 Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, Andrey Rybalchenko: Horn Clause Solvers for Program Verification. Fields of Logic and Computation II 2015: 24-51

3.3 Smart contracts in Bitcoin and BitML

Massimo Bartoletti (University of Cagliari, IT)

Joint work of Massimo Bartoletti, Roberto Zunino
Main reference Massimo Bartoletti, Roberto Zunino: "BitML: A Calculus for Bitcoin Smart Contracts", in Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pp. 83–100, ACM, 2018.
URL https://doi.org/10.1145/3243734.3243795

Although Bitcoin is mainly used to exchange crypto-currency, its blockchain and consensus mechanism can also be exploited to execute smart contracts, allowing mutually untrusted parties to exchange crypto-assets according to pre-agreed rules. To this purpose, Bitcoin features a non Turing-complete script language, which is used to specify the redeem conditions of transactions. This is a simple language of expressions, without loops or recursion. To write complex smart contracts, one needs to suitably combine transactions: in this approach, executing a contract amounts to appending sequences of transactions in a given order.

A drawback of this approach is that the complexity of writing smart contracts grows quickly in the number of transactions needed to implement it. Reasoning about the correctness of these contracts is even harder: one would have to consider computational adversaries who interact with the blockchain, only being constrained to use PPTIME algorithms. To overcome these issues we have proposed BITML [1], a high-level DSL for smart contracts with a computationally sound compiler to Bitcoin transactions.

The computational soundness property allows us to reason about contracts at the symbolic level of the BitML semantics. We exploit this possibility to investigate a landmark property of contracts, called liquidity, which ensures that funds never remain frozen within a contract. Liquidity is a relevant issue, as witnessed by a recent attack to the Ethereum Parity Wallet, which has frozen 160M USD within the contract, making this sum unredeemable by any user.

We develop a static analysis for liquidity of BitML contracts. This is achieved by first devising a finite-state, safe abstraction of infinite-state semantics of BitML, and then model-checking this abstraction.

We conclude by discussing a few open issues: in particular, how to enhance the expressiveness of Bitcoin contracts via minor extensions of the Bitcoin script language, and how to reduce the cost of executing contracts.

References

1 Massimo Bartoletti, Roberto Zunino: BitML: A Calculus for Bitcoin Smart Contracts. CCS 2018: 83-100

3.4 On Supporting Smart Contract Verification in Z3

Nikolaj S. Bjørner (Microsoft – Redmond, US)

We give an overview of current activities and features in Z3 that are aimed to make reasoning about smart contracts more efficient. The use of SMT solvers for Smart Contract analysis spans a range of scenarios, noteworthy symbolic execution, extended static analysis, symbolic model checking through solving satisfiability of Horn clauses, to program verification style reasoning. Thus, there is a fair range of reasoning capabilities that are relevant for Smart Contracts. Based on current experiences from users of z3 the talk presents ongoing work and extensions that may be of use for advancing reasoning about smart contracts.

Native Large Bit-width reasoning Verification conditions seem from Certora include code paths from EVM that involve bit-vectors with 256 bits each. When translating bit-vector reasoning to integers, the large bit-widths result in formulas with numerals that require expensive representations of large numerals. Integer reasoning does not assume fixed width numerals and has to take into account that integers can be unbounded. Integer reasoning is furthermore limited when it comes to non-linear arithmetic, as generally even quantifier-free non-linear integer satisfiability is undecidable (Hilbert's 10'th problem). Z3's native bit-vector reasoning engine converts bit-vector reasoning to propositional SAT. The overhead of representing bit-vector multiplication and division for large bit-widths makes propositional bit-vector reasoning impractical. With Jakob Rath at TU Vienna we are developing an new word level bit-vector reasoning engine in Z3 called *PolySAT*. PolySAT builds on and extends ideas developed by [1] to handle constraints that involve polynomial inequalities over bit-vectors. The main innovations in PolySAT include a generalization of conflict detection to linear inequalities, not only linear inequalities with unit coefficients. Conflict detection is complemented by an on-demand *saturation* phase to generalize infeasible cores.

Refinement Sorts Uses of Z3 at Meta (Facebook) suggest the relevance of integrating refinement sorts to the input formalism of SMT solvers. For example, the sort of natural numbers is a refinement sort of integers that are non-negative. Each natural number is an integer that is also non-negative. We illustrate how refinement sorts can be supported as theory that lazily instantiates axioms required to enforce refinement constraints.

Code as Constraints A new capability in Z3 is (re)exposing a capability to encode on-demand propagators outside of the solver. This enables users to encode properties that may require a bloated axiomatization.

References

1 Stéphane Graham-Lengrand and Dejan Jovanovic and Bruno Dutertre, *Solving Bitvectors with MCSAT: Explanations from Bits and Pieces*, in Automated Reasoning – 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I

3.5 Resource-Aware Session Types for Digital Contracts

Ankush Das (Amazon – Cupertino, US)

Joint work of Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, Ishani Santurkar
Main reference Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, Ishani Santurkar: "Resource-Aware Session Types for Digital Contracts", in Proc. of the 34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021, pp. 1–16, IEEE, 2021.
URL https://doi.org/10.1109/CSF51468.2021.00004

Programming digital contracts comes with unique challenges, which include (i) expressing and enforcing protocols of interaction, (ii) controlling resource usage, and (iii) preventing the duplication or deletion of a contract's assets. This talk presents the type-theoretic foundation and implementation of Nomos, a programming language for digital contracts that addresses these challenges. To express and enforce protocols, Nomos is based on shared binary session types. To control resource usage, Nomos employs automatic amortized resource analysis. To prevent the duplication or deletion of assets, Nomos uses a linear type system. A monad integrates the effectful session-typed language with a general-purpose functional language. Nomos' prototype implementation features linear-time type checking and efficient type reconstruction that includes automatic inference of resource bounds via off-the-shelf linear optimization. The effectiveness of the language is evaluated with case studies about implementing common smart contracts such as auctions, elections, and currencies. Nomos is completely formalized, including the type system, a cost semantics, and a transactional semantics to instantiate Nomos contracts on a blockchain. The type soundness proof ensures that protocols are followed at run-time and that types establish sound upper bounds on the resource consumption, ruling out re-entrancy and out-of-gas vulnerabilities.

3.6 Rich Specifications for Ethereum Smart Contract Verification

Marco Eilers (ETH Zürich, CH)

Joint work of Christian Bräm, Marco Eilers, Peter Müller, Robin Sierra, Alexander J. Summers
Main reference Christian Bräm, Marco Eilers, Peter Müller, Robin Sierra, Alexander J. Summers: "Rich specifications for Ethereum smart contract verification", Proc. ACM Program. Lang., Vol. 5(OOPSLA), pp. 1–30, 2021.
URL https://doi.org/10.1145/3485523

The verification of smart contracts poses challenges that rarely arise in other domains due to their typical use case (manipulating and transferring resources) and the necessity to interact with adversarial outside code. In this talk, I present a novel specification methodology, tailored to the domain of smart contracts, which enables (1) sound and precise reasoning in the presence of unverified code and arbitrary re-entrancy, (2) modular reasoning about collaborating smart contracts, and (3) domain-specific specification of resources and resource transfers, expressing a contract's behavior in intuitive and concise ways and excluding typical errors by default. I also briefly show the implementation of our technique in 2vyper, an SMT-based automated verification tool for Ethereum smart contracts written in Vyper, demonstrating its effectiveness for verifying real-world contracts.

3.7 Verifying Lighting in Why3

Grzegorz Fabianski (University of Warsaw, PL)

Joint work of Grzegorz Fabianski, Rafał Stefański

Lighting Network is an off-chain payment protocol working over bitcoin (and arguably the biggest application of scripting capabalities of bitcoin). As such, it uses complicated logic on the client-side to circumvent limited capabalities of bitcoin scripting language. In this talk, I present the status of ongoing work about verifying the Lighting network in the Why3 system. I will describe techniques that will enable us to verify randomized protocol (like Lighting) using deterministic Hoare Logic. Then I'll explain an overview of the project architecture and used abstractions.

3.8 Consensus for Decentralized Ledgers

Bryan Ford (EPFL Lausanne, CH)

URL https://drive.google.com/file/d/1M0xEtOT7lNBQFE7DCzWhBDMFrSKXnOey/view?usp=sharing

Blockchain and distributed ledger technology, as popularized by Bitcoin, has reinvigorated the classic computer science topic of consensus algorithms and protocols, and sent research in this space in many new directions. This talk summarizes a few of these developments in consensus for decentralized ledger systems. While classic "permission" consensus mechanisms such as Paxos assume a fixed set of a few consensus nodes, cryptocurrencies like Bitcoin established new expectations: to be open to "permissionless" participation; to scale to thousands or millions of participants; and to ensure that Byzantine security increases as participation increases and diversifies. Bitcoin's "Nakamoto consensus" is slow and has many other costs and limitations, however. Research on improving blockchain consensus has introduced "hybrid" schemes such as Byzcoin that achieve the best properties of Bitcoin-style permissionless consensus and PBFT-style Byzantine consensus. Sharding schemes such as Omniledger allow a blockchain's processing capacity to increase via horizontal scalability as the number of participants grows, potentially without bound, without sacrificing Byzantine security or the ability to execute transactions atomically across shards. Random beacon protocols such as RandHound/RandHerd and drand are instrumental to enabling secure sharding and other advanced blockchain consensus schemes. New asynchronous consensus algorithms inspired by blockchain systems, while still not yet deployed in practice, promise greater resilience to potentially-adversarial network conditions such as denial-of-service attacks once they become truly practical. New structuring concepts such as threshold logical clocks (TLC) may help make asynchronous consensus both more practical and more understandable. Proof of Stake offers an alternative permissionless participation foundation to proof of work, offering much lower energy waste, but still suffers from potential (re-)centralization or "rich get richer" effects. Proof of Personhood schemes, such as pseudonym parties, offer a more egalitarian path towards inclusive permissionless participation, attempting to ensure "one person, one vote" or "one person, one unit of stake" in permissionless blockchain consensus.

References

1 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, Bryan Ford: OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. IEEE Symposium on Security and Privacy 2018: 583-598.

2 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, Bryan Ford: Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. USENIX Security Symposium 2016: 279-296

3 Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, Bryan Ford: Scalable Bias-Resistant Distributed Randomness. IEEE Symposium on Security and Privacy 2017: 444-460

4 Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, Bryan Ford: CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. USENIX Security Symposium 2017: 1271-1287

5 Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Bryan Ford: Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies. EuroS&P Workshops 2017: 23-26

6 Bryan Ford: Threshold Logical Clocks for Asynchronous Distributed Coordination and Consensus. CoRR abs/1907.07010 (2019)

7 Bryan Ford, Philipp Jovanovic, Ewa Syta: Que Sera Consensus: Simple Asynchronous Agreement with Private Coins and Threshold Logical Clocks. CoRR abs/2003.02291 (2020)

8 Bryan Ford: Identity and Personhood in Digital Democracy: Evaluating Inclusion, Equality, Security, and Privacy in Pseudonym Parties and Other Proofs of Personhood. CoRR abs/2011.02412 (2020)

3.9 Program analysis tools for software auditors

Diego Garbervetsky (University of Buenos Aires, AR)

Joint work of Diego Garbervetsky, Javier Godoy, Juan Pablo Galeotti, Sebastian Uchitel

In this talk I will summary part of the collaboration work with OpenZeppelin. In particular our quest for tools that can help auditors to be more productive. In this collaboration we first explore how expert perform software auditors, we then participated in audits trying to see how to formalize some of the properties found by auditors. We also performed a survey of existing tools and analyze which tools could fit OpenZeppelin's audit process. Finally we discuss a new approach to understand and validate smart contracts based on abstractions of behavioral models.

3.10 Modular verification of memory-manipulating programs

Isabel Garcia-Contreras (IMDEA Software – Madrid, ES)

Joint work of Isabel Garcia-Contreras, Arie Gurfinkel, Jorge A. Navas

In SMT-based model-checking (SMT-MC) the correctness of a program is determined through the satisfiability of logical verification conditions (VCs) expressing the program semantics. A popular approach to model memory is to encode memory accesses using array store or select terms. When generating modular (i.e., per function) VCs, functions modifying memory take arrays as parameters, and their summaries consist then of two sub-formulae: one expressing memory changes and the other, called the frame, expressing the unmodified parts. Due to the unbounded nature of arrays, the frame is often expressed by quantified formulae.

In this talk, we focus on the problem of discovering automatically inductive invariants and function summaries. We contribute to the generation of modular VCs, using Constrained Horn Clauses (CHCs), which are more amenable for SMT-MC. We first propose a new static analysis that infers the finite memory footprint of a function. That is, the memory regions that may be only accessed in a bounded number of locations. Second, we encode finite memory using finite maps, eliminating the need of quantifiers to express frame axioms. We propose a theory of finite maps adapted to CHCs and an algorithm to check satisfiability of CHCs over integers, arrays and finite maps.

3.11 On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols

Arthur Gervais (Imperial College London, GB)

Joint work of Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, Arthur Gervais
Main reference Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, Arthur Gervais: "On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols", CoRR, Vol. abs/2103.02228, 2021.
URL https://arxiv.org/abs/2103.02228

In this paper, we investigate two methods that allow us to automatically create profitable DeFi trades, one well-suited to arbitrage and the other applicable to more complicated settings. We first adopt the Bellman-Ford-Moore algorithm with DEFIPOSER-ARB and then create logical DeFi protocol models for a theorem prover in DEFIPOSER-SMT. While DEFIPOSER-ARB focuses on DeFi transactions that form a cycle and performs very well for arbitrage, DEFIPOSER-SMT can detect more complicated profitable transactions. We estimate that DEFIPOSER-ARB and DEFIPOSER-SMT can generate an average weekly revenue of 191.48ETH (76,592USD) and 72.44ETH (28,976USD) respectively, with the highest transaction revenue being 81.31ETH(32,524USD) and22.40ETH (8,960USD) respectively. We further show that DEFIPOSER-SMT finds the known economic bZx attack from February 2020, which yields 0.48M USD. Our forensic investigations show that this opportunity existed for 69 days and could have yielded more revenue if exploited one day earlier. Our evaluation spans 150 days, given 96 DeFi protocol actions, and 25 assets.

Looking beyond the financial gains mentioned above, forks deteriorate the blockchain consensus security, as they increase the risks of double-spending and selfish mining. We explore the implications of DEFIPOSER-ARB and DEFIPOSER-SMT on blockchain consensus.

Specifically, we show that the trades identified by our tools exceed the Ethereum block reward by up to 874x. Given optimal adversarial strategies provided by a Markov Decision Process (MDP), we quantify the value threshold at which a profitable transaction qualifies as Miner ExtractableValue (MEV) and would incentivize MEV-aware miners to fork the blockchain. For instance, we find that on Ethereum, a miner with a hash rate of 10% would fork the blockchain if an MEV opportunity exceeds 4x the block reward.

3.12 Gigahorse: A Declaratively-Specified EVM Binary Lifter

Neville Grech (University of Malta – Msida, MT)

Joint work of Neville Grech, Lexi Brent, Bernhard Scholz, Yannis Smaragdakis
Main reference Neville Grech, Lexi Brent, Bernhard Scholz, Yannis Smaragdakis: "Gigahorse: thorough, declarative decompilation of smart contracts", in Proc. of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, pp. 1176–1186, IEEE / ACM, 2019.
URL https://doi.org/10.1109/ICSE.2019.00120

Smart contracts on blockchain platforms (e.g. Ethereum) represent a software domain with critical correctness needs. Smart contract users and security auditors can greatly benefit from a mechanism to recover the original structure of contracts, as evident from past work: many security analyses of smart contracts begin with a decompilation step.

In this talk, we present the Gigahorse framework, which is at the core of the contract-library.com service. Contract-library.com contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analyses applied to these in realtime. The Gigahorse framework is a decompilation and security analysis framework that natively supports Ethereum Virtual Machine (EVM) bytecode. Its internal intermediate representation of smart contracts makes implicit data- and control-flow dependencies of the EVM bytecode explicit. Using this framework we have developed and adapted several advanced high-level client analyses, including MadMax and Ethainter. All our client analyses benefit from high-level domain-specific concepts (such as "dynamic data structure storage" and "safely resumable loops") and achieve high precision and scalability.

One such client analysis, MadMax, flags contracts with a current monetary value in the $B range. (Manual inspection of a sample of flagged contracts shows that 81% of the sampled warnings do indeed lead to vulnerabilities.)

3.13 solc-verify: A Modular Verifier for Solidity Smart Contracts

Ákos Hajdu (Budapest Univ. of Technology & Economics, HU)

Joint work of Hajdu, Ákos; Jovanović, Dejan; Ciocarlie, Gabriela
Main reference Ákos Hajdu, Dejan Jovanovic: "solc-verify: A Modular Verifier for Solidity Smart Contracts", in Proc. of the Verified Software. Theories, Tools, and Experiments – 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 12031, pp. 161–179, Springer, 2019.
URL https://doi.org/10.1007/978-3-030-41600-3_11

Solc-verify [1] is a source-level verification tool for Ethereum smart contracts. It takes smart contracts written in Solidity and discharges verification conditions using modular program analysis and SMT solvers. Built on top of the Solidity compiler, solc-verify reasons at the

level of the contract source code. This enables solc-verify to effectively reason about high-level contract properties while modeling low-level language semantics precisely. The contract properties, such as contract invariants, loop invariants, function pre- and post-conditions, and event specifications [1, 3] can be provided as annotations in the code by the developer. This enables automated, yet user-friendly formal verification for smart contracts.

A distinguishing feature of solc-verify is its memory model [2], which is based on a formalization that covers all features of the language related to managing state and memory. In addition, the formalization is effective: all but few features can be encoded in the quantifier-free fragment of standard SMT theories. This enables precise and efficient reasoning about the state of smart contracts. The formalization is implemented in solc-verify and we provide an extensive set of tests that covers the breadth of the required semantics. We also provide an evaluation on the test set that validates the semantics and shows the novelty of the approach compared to other Solidity-level contract analysis tools.

References

1 Ákos Hajdu and Dejan Jovanović. *solc-verify: A Modular Verifier for Solidity Smart Contracts*. VSTTE 2019
2 Ákos Hajdu and Dejan Jovanović. *SMT-Friendly Formalization of the Solidity Memory Model*. ESOP 2020
3 Ákos Hajdu, Dejan Jovanović and Gabriela Ciocarlie. *Formal Specification and Verification of Solidity Contracts with Events (short paper)*. FMBC 2020

3.14 Smart contract = contract + control + settlement

Fritz Henglein (University of Copenhagen, DK)

Joint work of Fritz Henglein, Christian Kjær Larsen, Agata Murawska
Main reference Fritz Henglein, Christian Kjær Larsen, Agata Murawska: "A Formally Verified Static Analysis Framework for Compositional Contracts", in Proc. of the Financial Cryptography and Data Security – FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 12063, pp. 599–619, Springer, 2020.
URL https://doi.org/10.1007/978-3-030-54455-3_42

We present a smart contract architecture where a smart contract is decomposed into a declarative contract (composed from subcontracts specifying obligations and permission), contract manager (a generic program that, for any given contract, monitors or controls contract events to be consistent with the contract's semantics), and ra esource manager (a system that maintains ownership state of user-definable resource types, which accepts only resource-preserving transfers and thus guarantees nonduplication of any resource). This separation of concerns facilitates expressing contracts in a declarative domain-specific language for expressing commercial contracts and financial instruments, with formally specified denotational semantics and support for mechanized static analysis [1].

References

1 Fritz Henglein, Christian Kjær Larsen, Agata Murawska. *A Formally Verified Static Analysis Framework for Compositional Contracts*. Proc. 4th Workshop on Trusted Smart Contracts, February 2020

3.15 Speculative Smart Contracts

Jing Chen (Algorand Inc, US)

Joint work of Jing Chen, Maurice Herlihy, John Jannotti, Victor Luchangco, Liuba Shrira

Existing smart contract architectures suffer from a bottleneck problem: smart contract calls result in user code being executed in the ledger's critical path, potentially delaying simple payments and transfers. We describe an alternative smart contract structure that executes user code speculatively away from the blockchain's critical path. A secure committee validates and votes on the results of each such execution, certifying the execution's preconditions and its effects, and forwarding the certified results to a distinct consensus committee that manages access to the ledger itself.

3.16 Testing Cosmos applications with TLA+ and Apalache

Igor Konnov (Informal Systems – Wien, AT)

TLA+ is a language for formal specification of all kinds of computer systems. System designers use this language to specify concurrent, distributed, and fault-tolerant protocols, which are traditionally presented in pseudo-code. At Informal Systems, we are using TLA+ to specify and reason about the protocols that are implemented in the Tendermint blockchains and Cosmos ecosystem. To this end, we run Apalache, our symbolic model checker for TLA+.

In this talk, we show how to leverage TLA+ and Apalache to produce tests for blockchain applications. In our approach, verification engineers are incrementally writing TLA+ specifications and their expected properties. By running the model checker, they produce sequences of transactions, to be tried against the test environment. While this approach can be used for testing Cosmos applications as a black box, we find it to be the most effective when verification engineers have access to the source code.

3.17 Towards Automated Verification of Smart Contract Fairness

Yi Li (Nanyang TU – Singapore, SG)

Joint work of Ye Liu, Yi Li, Shang-Wei Lin, Rong Zhao
Main reference Ye Liu, Yi Li, Shang-Wei Lin, Rong Zhao: "Towards Automated Verification of Smart Contract Fairness", in Proc. of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, p. 666–677, Association for Computing Machinery, 2020.
URL https://doi.org/10.1145/3368089.3409740

Smart contracts are computer programs allowing users to define and execute transactions automatically on top of the blockchain platform. Many of such smart contracts can be viewed as games. A game-like contract accepts inputs from multiple participants, and upon ending, automatically derives an outcome while distributing assets according to some predefined rules. Without clear understanding of the game rules, participants may suffer

from fraudulent advertisements and financial losses. In this paper, we present a framework to perform (semi-)automated verification of smart contract fairness, whose results can be used to refute false claims with concrete examples or certify contract implementations with respect to desired fairness properties. We implement FairCon, which is able to check fairness properties including truthfulness, efficiency, optimality, and collusion-freeness for Ethereum smart contracts. We evaluate FairCon on a set of real-world benchmarks and the experiment result indicates that FairCon is effective in detecting property violations and able to prove fairness for common types of contracts.

3.18 Practical and Provably Sound Static Analysis of Ethereum Smart Contracts

Matteo Maffei (TU Wien, AT)

Joint work of Matteo Maffei, Clara Scheidewind, Markus Scherer, Ilya Grishchenko
Main reference Clara Schneidewind, Ilya Grishchenko, Markus Scherer, Matteo Maffei: "eThor: Practical and Provably Sound Static Analysis of Ethereum Smart Contracts", in Proc. of the CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020, pp. 621–640, ACM, 2020.
URL https://doi.org/10.1145/3372297.3417250

Ethereum has emerged as the most popular smart contract development platform, with hundreds of thousands of contracts stored on the blockchain and covering a variety of application scenarios, such as auctions, trading platforms, and so on. Given their financial nature, security vulnerabilities may lead to catastrophic consequences and, even worse, they can be hardly fixed as data stored on the blockchain, including the smart contract code itself, are immutable. An automated security analysis of these contracts is thus of utmost interest, but at the same time technically challenging for a variety of reasons, such as the specific transaction-oriented programming mechanisms, which feature a subtle semantics, and the fact that the blockchain data which the contract under analysis interacts with, including the code of callers and callees, are not statically known.

In this talk, I will present eThor, the first sound and automated static analyzer for EVM bytecode, which is based on an abstraction of the EVM bytecode semantics based on Horn clauses. In particular, our static analysis supports reachability properties, which we show to be sufficient for capturing interesting security properties for smart contracts (e.g., single-entrancy) as well as contract-specific functional properties. Our analysis is proven sound against a complete semantics of EVM bytecode and an experimental large-scale evaluation on real-world contracts demonstrates that eThor is practical and outperforms the state-of-the-art static analyzers.

This talk is based on a paper with the same title presented at CCS 2020.

3.19 What we do at Certora

Alexander Nutz (Certora – Berlin, DE)

Joint work of All members of Certora

Certora's mission is "ensuring smart contract security". To achieve this we are developing a specification language for EVM smart contracts with the goal of being accessible to programmers with only basic preexisting knowledge of formal verification. In addition, specifications should be as portable as possible. In this context it is crucial to strike a balance between minimally invasive specifications (which is important for understandability and portability) and specs that are amenable to automatic proving techniques. We are also developing a tool to automatically check these specifications, which works by translating correctness queries to SMT formulas. There are numerous challenges in having these formulas solved by today's SMT solvers. We apply a variety of simplifications and abstractions to make this feasible. Static analysis is an important enabler to make these transformations sound. When running SMT solvers we are using a portfolio approach; depending on the input, we can translate to various different encodings as well as running different solvers and configurations. We also work closely with SMT solver developers to solve remaining problems. In particular the fragment of large bit vectors (256 bit) combined with nonlinear arithmetic is crucial for our efforts while having gotten comparatively little attention in the past.

3.20 Off-Chain Protocols meet Game Theory

Sophie Rain (TU Wien, AT)

Joint work of Sophie Rain, Zeta Avarikioti, Laura Kovács, Matteo Maffei
Main reference Sophie Rain, Zeta Avarikioti, Laura Kovács, Matteo Maffei: "Towards a Game-Theoretic Security Analysis of Off-Chain Protocols", CoRR, Vol. abs/2109.07429, 2021.
URL https://arxiv.org/abs/2109.07429

On-chain protocols constitute one of the most promising approaches to solve the inherent scalability issue of blockchain technologies. The core idea is to let parties transact on-chain only once to establish a channel between them, leveraging later on the resulting channel paths to perform arbitrarily many peer-to-peer transactions on-chain. While significant progress has been made in terms of proof techniques for on- chain protocols, existing approaches do not capture the game-theoretic incentives at the core of their design, which led to overlooking significant attack vectors like the Wormhole attack in the past. This work introduces the first game-theoretic model that is expressive enough to reason about the security of on-chain protocols. We advocate the use of Extensive Form Games EFGs and introduce two instances of EFGs to capture security properties of the closing and the routing of the Lightning Network. Specifically, we model the closing protocol, which relies on punishment mechanisms to disincentivize the uploading on-chain of old channel states, as well as the routing protocol, thereby formally characterizing the Wormhole attack, a vulnerability that undermines the fee-based incentive mechanism underlying the Lightning Network.

3.21 Formal Methods in Zero-Knowledge Protocols: Challenges in the circom Programming Language

Albert Rubio (Complutense University of Madrid, ES)

Joint work of Elvira Albert, Jordi Baylina, Marta Belles-Muñoz, Hermenegildo García-Navarro, Miguel Isabel-Márquez, José Manuel Muñoz-Tapia, Clara Rodríguez-Núñez, Albert Rubio

The most widely studied language in the context of Zero-Knowledge (ZK) proofs is arithmetic circuit satisfiability. In this talk we present circom, a programming language and a compiler that allows the programmer to provide a low-level description of the arithmetic circuit together with an effective way to execute it. We will introduce challenging safety properties to be checked in circom programs and show the need of improving existing techniques to analyse and simplify the nonlinear arithmetic constraints generated by the compiler.

3.22 Sharding Smart Contracts

Ilya Sergey (National University of Singapore, SG)

Joint work of George Pîrlea, Amrit Kumar, Ilya Sergey
Main reference George Pîrlea, Amrit Kumar, Ilya Sergey: "Practical smart contract sharding with ownership and commutativity analysis", in Proc. of the PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, pp. 1327–1341, ACM, 2021.
URL https://doi.org/10.1145/3453483.3454112

Sharding is a popular way to achieve scalability in blockchain protocols. Existing approaches for blockchain sharding, however, do not scale well when concurrent transactions alter the same replicated state component–a common scenario in Ethereum-style smart contracts.

I will outline a novel approach for efficiently sharding such transactions. It is based on a folklore idea: state-manipulating atomic operations that commute can be processed in parallel, with their cumulative result defined deterministically, while executing non-commuting operations requires one to own the state they alter. We developed a static program analysis that soundly infers ownership and commutativity summaries for smart contracts and translates those summaries to sharding signatures that are used by the blockchain protocol to maximise parallelism. Our evaluation shows that using the analysis introduces negligible overhead to the transaction validation cost, while the inferred signatures allow the system to achieve a significant increase in transaction processing throughput for real-world smart contracts.

3.23 Smart Contract Vulnerabilities and Analysis

Yannis Smaragdakis (University of Athens, GR) and Neville Grech (University of Malta – Msida, MT)

Joint work of Yannis Smaragdakis, Neville Grech, Sifis Lagouvardos, Konstantinos Triantafyllou, Ilias Tsatiris
Main reference Yannis Smaragdakis, Neville Grech, Sifis Lagouvardos, Konstantinos Triantafyllou, Ilias Tsatiris: "Symbolic value-flow static analysis: deep, precise, complete modeling of Ethereum smart contracts", Proc. ACM Program. Lang., Vol. 5(OOPSLA), pp. 1–30, 2021.
URL https://doi.org/10.1145/3485540

In this talk, I give a quick introduction to the kinds of vulnerabilities that often appear in Ethereum smart contract coding, and discuss a static analysis infrastructure that has led to multiple high-profile vulnerability disclosures in the past year.

The main analysis architecture is "symbolic value-flow" (symvalic) analysis: a technique that reasons about the program both symbolically and with concrete values, while abstracting away from the program's control flow. Precision is being maintained through a set of "dependencies" between inferred values. This analysis architecture represents an attempt to defeat the state-explosion problem (as in model checking or concrete execution) by sacrificing a small amount of precision.

3.24 Accounts vs UTXO

Philip Wadler (University of Edinburgh, GB)

Joint work of Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, Philip Wadler
Main reference Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, Philip Wadler: "The Extended UTXO Model", in Proc. of the Financial Cryptography and Data Security – FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers, Lecture Notes in Computer Science, Vol. 12063, pp. 525–539, Springer, 2020.
URL https://doi.org/10.1007/978-3-030-54455-3_37

The talk offered a brief description of two approaches to tracking balances, *accounts* as used by Ethereum and *UTxO* (Unspent Transaction Outputs) as used by Bitcoin and Cardano. The strengths and weaknesses of the two are contrasted. One strength of UTxO as compared with contracts is that the precise cost of running the smart contract can be calculated in advance – there are never any surprises where, due to a change on the blockchain that occurred between submitting the transaction and running the transaction, the cost of running the transaction has changed.

References

1 Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, Philip Wadler: The Extended UTXO Model. Financial Cryptography Workshops 2020: 525-539.
2 Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova, Philip Wadler: Native Custom Tokens in the Extended UTXO Model. ISoLA (3) 2020: 89-111

3.25 Formal Verification of Smart Contracts with the Move Prover

Wolfgang Grieskamp (Facebook – Bellevue, US)

Joint work of David Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Meng Xu, Emma Zhong
Main reference David L. Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Meng Xu, Jingyi Emma Zhong: "Fast and Reliable Formal Verification of Smart Contracts with the Move Prover", CoRR, Vol. abs/2110.08362, 2021.
URL https://arxiv.org/abs/2110.08362

The Move Prover (MVP) is a formal verifier for smart contracts written in the Move programming language. MVP has an expressive specification language, and is fast and reliable enough that it can be run routinely by developers and in integration testing. Besides the simplicity of smart contracts and the Move language, three implementation approaches are responsible for the practicality of MVP: (1) an alias-free memory model, (2) fine-grained invariant checking, and (3) monomorphization. The entirety of the Move code for the Diem blockchain has been extensively specified and can be completely verified by MVP in a few minutes. Changes in the Diem framework must be successfully verified before being integrated into the open source repository on GitHub.

3.26 Checking Properties of Smart Contract Systems

Valentin Wüstholz (ConsenSys – Kaiserslautern, DE) and Maria Christakis (MPI-SWS – Kaiserslautern, DE)

Joint work of Dimitar Bounov, Maria Christakis, Arie Gurfinkel, Joran J. Honig, Jorge A. Navas, Richard J. Trefler, Scott Wesley, Valentin Wüstholz
Main reference Valentin Wüstholz, Maria Christakis: "Harvey: a greybox fuzzer for smart contracts", in Proc. of the ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020, pp. 1398–1409, ACM, 2020.
URL https://doi.org/10.1145/3368089.3417064

Ensuring the correctness of smart contracts is becoming increasingly challenging. We provide an overview of how to check custom correctness properties for complex systems of smart contracts. We introduce the Harvey fuzzer which is used in two industrial analysis services for smart contracts. We also provide a brief overview of the Scribble specification language that we use to instrument contracts with runtime checks. Finally, we introduce SmartACE, a new analysis framework for Solidity contracts.

References

1 Valentin Wüstholz, Maria Christakis: *Harvey: a greybox fuzzer for smart contracts.* ESEC/SIGSOFT FSE 2020: 1398-1409
2 Valentin Wüstholz, Maria Christakis: *Targeted greybox fuzzing with static lookahead analysis.* ICSE 2020: 789-800
3 Scott Wesley, Maria Christakis, Jorge A. Navas, Richard J. Trefler, Valentin Wüstholz, Arie Gurfinkel: *Compositional Verification of Smart Contracts Through Communication Abstraction.* SAS 2021: 429-452
4 Scott Wesley, Maria Christakis, Jorge A. Navas, Richard J. Trefler, Valentin Wüstholz, Arie Gurfinkel: *Verifying Solidity Smart Contracts Via Communication Abstraction in SmartACE.* VMCAI 2022 (to appear)

3.27 Int-blasting

Yoni Zohar (Bar-Ilan University – Ramat Gan, IL)

Joint work of Ahmed Irfan, Makai Mann, Aina Niemetz, Andres Noetzli, Mathias Preiner, Andrew Reynolds, Clark Barrett, Cesare Tinelli, Yoni Zohar

The state of the art for bit-precise reasoning in the context of Satisfiability Modulo Theories (SMT) is a SAT-based technique called bit-blasting where the input formula is first simplified and then translated to an equisatisfiable propositional formula. The main limitation of this technique is scalability, especially in the presence of large bit-widths and arithmetic operators.

In this talk we introduced an alternative technique, which we call *int-blasting*, based on a translation to an extension of integer arithmetic rather than propositional logic. We present several alternative translations, discuss their differences, and evaluate them on benchmarks that arise from verification of rewrite rule candidates for bit-vector solving, as well as benchmarks from SMT-LIB. We also provide preliminary results on 35 benchmarks that arise from smart contract verification. The evaluation shows that this technique is particularly useful for benchmarks with large bit-widths and can solve benchmarks that the state of the art cannot.

4 Working groups

4.1 Specification Languages for Smart Contracts (Group Discussion)

discussion participants

The discussion was centered around specification languages for smart contracts and broadly touched on various related topics, such as expressiveness (e.g., safety properties, liveness properties, hyperproperties) and accessibility to programmers without background in formal methods.

4.2 Verifying Arithmetic Circuits from Zero Knowledge Applications

Leo Alt and Nikolaj Bjørner

Zero knowledge cryptography enables private computation in the form of zkSNARKs [1], that is, you can use an application without revealing some private input. The ZCash blockchain pioneered private cryptocurrency transactions, and nowadays several different applications use the same or similar technology in different ways. These computations that are performed on private data are represented as polynomials in the deepest layer of the proof system. However, initially, they are represented by arithmetic circuits. The small but huge difference from the machine circuits we may be used too is that the arithmetic on these circuits is performed over a very large prime field, since that is where the cryptographic primitives operate.

We studied how the ZoKrates [2] compiler translates high level program statements into arithmetic circuits. Because of the nature of the proof system and the verification process in the zkSNARK workflow (which I will not expand here), it is useful for such a compiler to not translate the statements intro precise constraints, but use rather weaker constraints to save circuit and proof size. In some cases, this may lead to nondeterminism in the circuit, which may or may not be harmful. This is under control for the statements that the compiler itself generates, but may become problematic once a developer starts writing custom constraints, as is common in Circom [3] and TurboPLONK [4].

We discussed about safety properties over these circuits, but converged on the smaller problem of detecting nondeterminism of a circuit. The research question now is, how can we modify/extend the tools we currently have in order to check the stated problem? If we just feed the circuit into an SMT solver that has Nonlinear Arithmetic support with extra modulo operations on all sums and multiplications, we know that this is quickly going to explode. So we need a specific approach that works fundamentally better on prime fields. Moreover, we also know that these primes are very large (254 bits), so this also needs to be taken into account when designing an algorithm for that. There are algorithms for finding roots of a system of polynomials, also on prime fields, but their complexity is prohibitively high in practice. However, we do not need full roots solving, we would be happy with satisfiability. So how can we design something in the middle?

References

1 https://zcash.github.io/halo2/concepts/proofs.html
2 https://zokrates.github.io/
3 https://github.com/iden3/circom
4 https://zcash.github.io/halo2/concepts/arithmetization.html

Participants

- Elvira Albert
Complutense University of Madrid, ES
- Leonardo Alt
Ethereum – Berlin, DE
- Nikolaj S. Bjørner
Microsoft – Redmond, US
- Maria Christakis
MPI-SWS – Kaiserslautern, DE
- Marco Eilers
ETH Zürich, CH
- Grzegorz Fabianski
University of Warsaw, PL
- Josselin Feist
Trail of Bits Inc. – New York, US
- Bryan Ford
EPFL Lausanne, CH
- Diego Garbervetsky
University of Buenos Aires, AR
- Isabel Garcia-Contreras
IMDEA Software – Madrid, ES
- Arthur Gervais
Imperial College London, GB
- Neville Grech
University of Malta – Msida, MT
- Wolfgang Grieskamp
Facebook – Bellevue, US
- Fritz Henglein
University of Copenhagen, DK
- Matteo Maffei
TU Wien, AT
- Alexander Nutz
Certora – Berlin, DE
- Sophie Rain
TU Wien, AT
- Albert Rubio
Complutense University of Madrid, ES
- Tanja Schindler
Universität Freiburg, DE
- Yannis Smaragdakis
University of Athens, GR
- Valentin Wüstholz
ConsenSys – Kaiserslautern, DE

Remote Participants

- Massimo Bartoletti
University of Cagliari, IT
- Andreea Buterchi
MPI-SWS – Kaiserslautern, DE
- Jing Chen
Stony Brook University, US
- Shuo Chen
Microsoft Research Asia – Beijing, CN
- Ankush Das
Amazon – Cupertino, US
- Stefan Dziembowski
University of Warsaw, PL
- Ákos Hajdu
Budapest Univ. of Technology & Economics, HU
- Aniket Kate
Purdue University – West Lafayette, US
- Markulf Kohlweiss
University of Edinburgh, GB
- Igor Konnov
Informal Systems – Wien, AT
- Yi Li
Nanyang TU – Singapore, SG
- Victor Luchangco
Algorand – Boston, US
- Anastasia Mavridou
NASA – Moffett Field, US
- Noam Rinetzky
Tel Aviv University, IL
- Grigore Rosu
University of Illinois – Urbana-Champaign, US
- Giulia Scaffino
TU Wien, AT
- Gerardo Schneider
University of Gothenburg, SE
- Ilya Sergey
National University of Singapore, SG
- Zhong Shao
Yale University – New Haven, US
- Philip Wadler
University of Edinburgh, GB
- Yoni Zohar
Bar-Ilan University – Ramat Gan, IL