Ensuring Correctness of Smart Contracts with Constrained Horn Clauses

Fabio Fioravanti[1] and Giulia Matricardi[1,2]

[1] DEc, University 'G. d'Annunzio', Chieti-Pescara, Italy
fabio.fioravanti@unich.it, giulia.matricardi001@studenti.unich.it
[2] Dottorato di Interesse Nazionale in Blockchain e DLT, University of Camerino, Italy
giulia.matricardi@studenti.unicam.it

Smart contacts are computer programs that specify and impose the execution of contracts and agreements by automatically performing predetermined actions, including payments, upon the occurrence of certain events or conditions. Any bugs or errors in the smart contracts code will become permanent once published and could lead to huge economic losses (e.g. DAO attack). Ensuring correctness of smart contracts is therefore of fundamental importance.

Many recent program analysis and verification methods [1,2] use *Constrained Horn Clauses* (CHCs) as a language for specifying (i) the semantics of programs, that can be written in a variety of programming languages, including imperative, functional, object-oriented, and concurrent ones, and (ii) program properties, including safety, termination, and program equivalence. CHCs are a fragment of First-Order Logic with the same syntax and semantics of *Constraint Logic Programs* (CLP) [3] but they are not intended to be directly executed as programs. Once the verification problem has been translated to CHCs the main interest is in checking their satisfiability (respectively, unsatisfiability) that guarantees that the property of interest is valid (resp. invalid) for the considered program.

Different CHC solvers have been developed for checking the satisfiability of CHCs such as, Eldarica[3], Golem[4], Spacer/Z3[5], and VeriMAP[6] that apply different decision procedures, possibly combining predicate abstraction, Counterexample Guided Abstraction Refinement (CEGAR), Property Directed Reachability (PDR), program transformation and abstract interpretation. Well-known program verification tools that are based on CHCs include SeaHorn[7] for LLVM languages, JayHorn[8] for Java, and RustHorn[9] for Rust. A workshop on Horn Clauses for Verification and Synthesis (HCVS)[10] and a competition of CHC solvers (CHC-COMP)[11] are held every year, since 2014 and 2018, respectively.

[3] https://github.com/uuverifiers/eldarica
[4] https://github.com/usi-verification-and-security/golem
[5] https://github.com/Z3Prover/z3/
[6] https://fmlab.unich.it/verimapwebgui/
[7] https://seahorn.github.io
[8] https://jayhorn.github.io/jayhorn/
[9] https://github.com/hopv/rust-horn
[10] https://www.sci.unich.it/hcvs23/
[11] https://chc-comp.github.io/

Ethereum is the most popular platform for storing and executing smart contracts, overall managing billions of dollars. Several tools for analysis and formal verification of Ethereum smart contracts are based on Horn clauses and combine abstraction, symbolic execution and partial evaluation, including Securify[12], eThor[13], HoRStify[14] and SmartACE[15]. Recently, the official Solidity compiler developed by the Ethereum Foundation has been equipped with a symbolic model checker that uses SMT and CHC solvers to check properties of smart contracts annotated using require and assert statements[16].

In order to ease the proof of (un)satisfiability, some semantics-preserving CHC transformation rules can be applied to perform a sequence of small modifications at clause level, which may propagate constraints by symbolic evaluation, uncover relations among variables, eliminate redundant variables or clauses, conjecture and confirm the existence of inductive invariants. The result of the transformation may be a radical restructuring of the whole set of clauses by changing their pattern of recursion. These CHC transformations, guided by suitable strategies, have been used, for instance,
(i) to generate the CHCs corresponding to a given verification problem by specializing different interpreters of a programming language,
(ii) to verify safety properties (unreachability of bad states) and relational properties (such as equivalence, injectivity, monotonicy, functional dependence, non-interference) of imperative programs over integers and arrays of integers,
(iii) to efficiently generate complex test data structures, e.g. AVL trees, in the property-based testing setting,
(iv) to verify properties of programs over Algebraic Data Types (ADT) (such as lists, trees, heaps, queues) improving the effectiveness of state-of-the-art solvers.

We propose to investigate the development of techniques based on CHC transformation for ensuring correctness of smart contracts including, but not limited to, those written for the Ethereum blockchain.

References

1. E. De Angelis, F. Fioravanti, J. P. Gallagher, M. V. Hermenegildo, A. Pettorossi, and M. Proietti. Analysis and transformation of constrained Horn clauses for program verification. *Theory and Practice of Logic Programming*, 22(6):974–1042, 2022.
2. A. Gurfinkel. Program verification with constrained horn clauses (invited paper). In S. Shoham and Y. Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2022.
3. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[12] https://github.com/eth-sri/securify2
[13] https://secpriv.wien/ethor/
[14] https://www.horstify.org/
[15] https://github.com/contract-ace/smartace
[16] https://docs.soliditylang.org/en/latest/smtchecker.html