Applying Formal Methods to Smart Contract Languages and Blockchain

Maxim Sokhatsky[1] and Pavlo Maslianko[1]

National Technical University of Ukraine,
"Igor Sikorsky Kyiv Polytechnical Institute",
Ukraine, maxim@synrc.com, WWW: http://groupoid.space
Address: 37, Prosp. Peremohy, Kyiv, Ukraine, 03056, Tel.: +380.44.2367989

Abstract. This paper shows our vision on the way to better contract languages and trusted foundations of Blockchain protocols. The main problem in current contract languages and its virtual machine implementations used in Blockchain and Ethereum cryptocurrencies is that they are built upon stack-based virtual machines while we propose pure lambda-calculus based computing environments with proven theorems if its evaluations strategies and also a corespondent high-level language with dependent types that can be used to embed smart contract DSL into dependently typed language. Also we will draw how formal verification could be applied to other Blockchain subsystems including proof-of-work protocols and distributed ledger.

Keywords: formal methods, software verification, program languages

1 Formal Verification

Formal verification is an ablity to prove properties in a form of theorems of a given system which are formulated in a axiomatic manner. While the origins of this approach led us to logic and automated provers (AUTOMATH, ACL2 and later Coq, F*) now it seems applicable to much wider spectrum of mathematical systems, including not only logic, but also geometry and topology.

2 Model Checkers

There are many approaches to formal verification, depending of origins and areas. E.g. one can prove properties (or we say, verify) for some model, then prove the isomorphism between that model and the actual program. Such systems are called model checkers and usually are just a tools and small steps in a process of formal verification. Well known model checkers are TLA+ for formal verification of distributed algorithms. Twelf for verification of computer languages, etc. This apporach involves several (more that one) languages into verification process.

3 Embeddable Languages

The another approach is to use single unified language for proving properties of models and language of actual computations. In that area dependently typed languages dominates in all applications. MLTT with Sigma and Pi types was introduced in 1972 and from that times developed enough to be used for extracting certified programs. Such process of erasing type information (that are used by type-checkers) is called extraction.

4 Comparing ULC and Stack-based machines

The target language of extracted programs are exacly the same as the operating language of smart contracts usually used in cryptocurrencies. So we can say that it is untyped lambda interpreter with proven properies. However those

RIPEMD160 SHA1 SHA256 HASH160 HASH256

References

1. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147, 195?197 (1981). doi:10.1016/0022-2836(81)90087-5
2. May, P., Ehrlich, H.-C., Steinke, T.: ZIB structure prediction pipeline: composing a complex biological workflow through web services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148?1158. Springer, Heidelberg (2006). doi:10.1007/11823285_121
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181?184. IEEE Press, New York (2001). doi:10.1109/HPDC.2001.945188
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: an open grid services architecture for distributed systems integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov