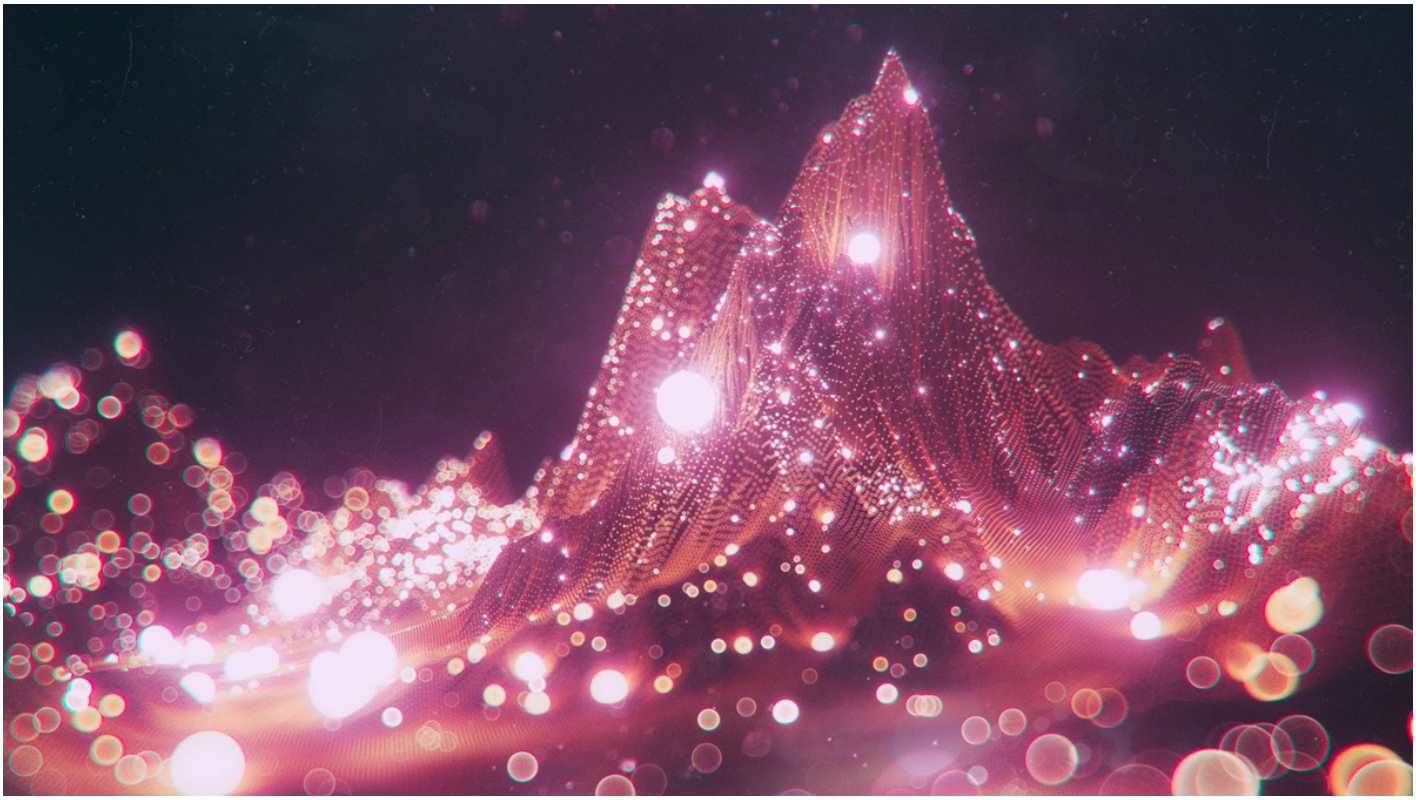
IOHK | BLOG

IELE: A New Virtual Machine for the Blockchain

Specialised smart contract execution on the blockchain

DECEMBER 15, 2017 GRIGORE ROSU 8 MIN READ



Runtime Verification (RV) is proud to release their first version of IELE, a new virtual machine for the blockchain.

What is IELE?

IELE is a variant of LLVM specialized to execute smart contracts on the blockchain. Its design, definition and implementation have been done at the highest mathematical standards, following a semantics-first approach with verification of smart contracts as a major objective. Specifically, we have defined the formal syntax and semantics of IELE using the K framework, which in return gives us an executable reference model in addition to a series of program analysis tools, including a program verifier. K was created by our team during the last 15 years and incorporates the state of the art in language design, semantics and formal methods. The design of IELE was based on our experience with formally defining dozens of languages in K, but especially on recent experience and lessons learned while formally defining two other virtual machines in K, namely:

- KEVM, our semantics of the Ethereum Virtual Machine (EVM); and
- KLLVM, our semantics of LLVM; the latest version of the LLVM semantics will be made public when complete and published, but an earlier version is available.

Unlike the EVM, which is a stack-based machine, IELE is a register-based machine, like LLVM. It has an unbounded number of registers and also supports unbounded integers. To get a feel for how IELE programs look like, here are two of them (these have not been verified yet and may change):

- erc20.iele - a IELE implementation of an ERC20 token
- forwardingWallet.iele - a wallet implementation that creates and calls into another contract

Design Rationale

Here are the forces that drove the design of IELE:

1. To serve as a uniform, lower-level platform for translating and executing smart contracts from higher-level languages. The contracts can interact with each other by means of an ABI (Application Binary Interface). The ABI is a core element of IELE, and not just a convention on top of it. The unbounded integers and unbounded number of registers should make compilation from higher-level languages more straightforward and elegant and, looking at the success of LLVM, more efficient in the long term. Indeed, many of the LLVM optimizations are expected to carry over. For that reason, IELE followed the design choices and representation of LLVM as much as possible. The team also includes LLVM experts from the University of Illinois (where LLVM was created).

2. To provide a uniform gas model, across all languages. The general design philosophy of gas calculation in IELE is "no limitations, but pay for what you consume". For example, the more registers a IELE program uses, the more gas it consumes. Or the larger the numbers computed at runtime, the more gas it consumes. The more memory it uses, in terms of both locations and size of data stored at locations, the more gas it consumes. And so on.

3. To make it easier to write secure smart contracts. This includes writing requirements specifications that smart contracts must obey as well as making it easier to develop automated techniques that mathematically verify / prove smart contracts correct with respect to such specifications. For example, pushing a possibly computed number on the stack and then jumping to it regarded as an address makes verification hard, and thus security weaker, with current smart contract paradigms. IELE has named labels, like LLVM, and jump statements can only jump to those labels. Also, it avoids the use of a bounded stack and not having to worry about stack or arithmetic overflow makes specification and verification of smart contracts significantly easier.

Like KEVM, the formal semantics of EVM that we previously defined, validated and evaluated using the K framework, the design of IELE was also done in a semantics-based style, using K. Together with a fast (LLVM-based) execution backend for K that is still under development, it is expected that the interpreter obtained automatically from the semantics of IELE will be sufficiently efficient to serve as a reference implementation of IELE.

What's next?

To achieve the full potential of IELE, we plan to next work on the following:

- Efficient backend for K. Then K semantics, including IELE, can be executed at acceptable performance. As discussed in our KEVM paper, the current version of K can execute the EVM semantics at performance that stays within an order of magnitude from the performance of the reference C++ implementation of the EVM. We believe that we can improve the execution performance of K by one order of magnitude. If this is achieved, then there is no incentive to implement IELE in an ad-hoc way: the K executable semantics of IELE will also be its implementation, so it will be correct by construction and thus implementation defects of the VM itself cannot be exploited anymore. Also, IELE itself would be easier to maintain and future versions will be easier to deploy.

- Compilers/Translators from Solidity and Plutus to IELE. Writing smart contracts directly in IELE is a bit more feasible than in the EVM, because IELE follows the LLVM IR which was designed to be human-readable, but the IELE code is still low-level and thus error-prone. To properly test IELE and gain confidence in its overall design and capabilities, we will implement a compiler/translator from Solidity to IELE, also in K. Since Plutus rises as a star among the functional programming languages for smart contracts and since we are defining a formal semantics of Plutus as well, a compiler from Plutus to IELE will be developed immediately after Solidity's.

- Semantics-based compilation. In addition to improving K's performance, we plan to implement a tool that we call semantics-based compiler on top of K. See our previous blog post for details. The idea is to take a programming language semantics L and a program P in L, and generate (using symbolic execution heavily) a new language semantics L' which is a specialization of L for P. We expect at least one order of magnitude increase in performance. More importantly, this will give us a uniform mechanism to translate any programs in any

programming languages that have a K semantics to IELE, thus making IELE and K into a universal platform for executing smart contracts in any language.

- Deploy IELE on the Cardano blockchain.

Technical Details and Download

IELE is thoroughly commented and freely available under the UIUC license (as permissive as the MIT license) on Github:

- IELE repository on Github

In addition to the two IELE programs mentioned above, erc20.iele and forwardingWallet.iele showing that the IELE code is human readable, the following links into the github repo will give a good idea what IELE is and how it differs from EVM and LLVM:

- iele-syntax.md - the complete formal syntax of the IELE language.
- iele.md - the complete formal executable semantics of the IELE language
- Design.md - the design rationale of IELE, as well as detailed comparisons with LLVM and EVM
- iele-gas.md - the current gas model of IELE (which is still to be tuned as we develop compilers to IELE)

Get involved

In the spirit of open source, community-driven development, we will be holding all IELE discussions on our channels

- #IELE:matrix.org on Riot
- runtimeverification/iele-semantics on Gitter

We encourage any interested parties to engage us, ask questions, contribute code, or build experience with our tools. We are also always looking for contributors able to work on documentation, efficient install/quickstart process for new developers, and more examples and tests. We are hiring, and will be sure to keep an eye open for helpful contributors!

We will also be posting updates on our brand new Twitter page @rv_inc, which we hope any interested developers will follow and interact with.

Let's build more secure smart contracts for everybody, together!

Acknowledgements

We warmly thank IOHK for their generous funding support of both IELE and KEVM. IELE, in particular, would have not been possible without IOHK's support, its continuous research meetings, and the stimulating technical discussions we had with their research team.

We also thank the K team who defined the KEVM semantics (see technical report, too) and verified smart contracts for ERC20 compliance. Their effort and non-trivial proofs at the EVM level led to the quest for a new VM with better support for verification of smart contracts.