Experiments in Formal Verification of Scala Code

Degree programme: BSc in Computer Science | Specialisation: IT Security
Thesis advisor: Prof. Dr. Kai Brünnler
Expert: Urs Keller (Revault Sàrl)

Correctness of Bitcoin Software is important, because bugs can lead to loss of funds. Contrary to testing, formal verification can ensure correctness, at least in principle. In this thesis we set out to formally verify a part of Bitcoin-S, a Scala implementation of the Bitcoin protocol. We use the Stainless verifier, developed by the LARA group of EPF Lausanne.

Formal Verification

Formal verification is a method to check the correctness of a program based on a formal specification. Using a verification tool, all possible inputs can be covered, in contrast to unit testing, where the inputs must be specified separately.

The Stainless Verification Tool

We use Stainless as our verification framework, which

- takes Scala code with specification as input, but does not support all Scala language features,
- reports inputs for which a program violates the specification or
- confirms the correctness of a program.

The First Property

The first property of Bitcoin-S we set out to verify we call the no-inflation-property:

- a regular transaction (non-coinbase) cannot generate new coins

The code uses Scala language features that Stainless does not support, so we have to transform it into the supported Scala subset.
We found that the code is too large to make this feasible in the available time.

However, during this work we found a bug in the function checking the correctness of a transaction. Its implementation did not allow transactions that referenced two or more outputs of the same previous transaction. We fixed it and made a pull request which has been merged by the developers of the Bitcoin-S project.

```
def +(c: CurrencyUnit): CurrencyUnit = {
  require(c.satoshis == Satoshis.zero)
  Satoshis(satoshis.underlying + c.satoshis.underlying)
} ensuring(res => res.satoshis == this.satoshis)
```

Stainless specification of the Second Property

The Second Property

So, we turn our analysis and verification to another property:

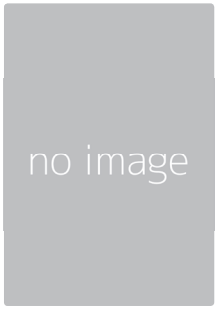
- adding zero coins to a number of coins results in the same number of coins

Here we were successful in transforming the relevant code into the supported subset of Scala and verifying it.

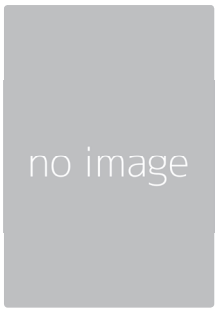
Conclusion

Because of the limitations of the verification tool, we could only verify a rewritten version of the original code. So code should be written specifically with formal verification in mind, in order to successfully verify it. Also, we found that trying to verify code reveals bugs. Finally, our work led to some feedback to the Stainless developers to improve the tool.

Our Bugfix



Ramon Boss



Anna Doukmak