Formal Verification Model in Folidity (*draft*)

Introduction

Folidity features model-first approach in coding and design of smart contracts. This can be faciliated by numerous techniques that have been actively researched in the fields of symbolic execution, model-bounded verification, static analysis, etc.

As a good starting point, it was decided to opt-in for symbolic execution, static analysis and bounded model checking.

Verification techniques

A typical program in folidity consists of data structures that describe models, states, and function that can interact with each other. Models are one of core structures that provide the model consistency guarantee in folidity. States can encapsulte different of the same models and describe explicit state transition or state mutations as part of program execution, and functions are the driving points in program execution. Functions declares and describe the state transitions.

One of the core pieces in the workflow aforementioned is the model bounds that consist of individual boolean constraints. As shown below:

```
# Some model and its constraints
model ModelA {
  x: int,
  y: int
} st [
  x > 10,
  y < 5
]
# A state that encapsulates a model and provides its own constraints.
state StateA(ModelA) st [
  x < y
]
# A function that describes mutation.
fn () mutate(value: int) when (StateA s) -> StateA
st [
    value > 100,
    value < 100,
] { ... }
```

Let's break down how each of the selected techniques can be applied to the program written in Folidity.

As a good starting point, we can perform a static analysis and verify that the program statements, declarations and constraints are valid and consistent.

A a simple approach is to perform semantic analysis that carries out type checking and verfication of correct state transition in the function body. Specifically, if `mutate()` expect to return `StateA`, but instead it perform a state transition to `StateB` we can already detect that at a compile time.

The next stage of the analysis involves verification of the consistency of models described.

We can generalise the approach using the following mathematical model.

We can describe some verification system $\boldsymbol{VS}$ as $\boldsymbol{VS} = \langle \mathbf{M}, \mathbf{E}, \boldsymbol{\Upsilon}, \Theta, \mathrm{T}_{\mathrm{M}}, \mathrm{T}_{\mathrm{E},\{\mathrm{E},\mathrm{M}\}}, \mathrm{T}_{\Upsilon,\mathrm{E}} \rangle$ where

- $\mathbf{M}$ - set of models in the system.
- $\mathbf{E}$ - set of states in the system
- $\boldsymbol{\Upsilon}$ - set of functions in the system.
- Θ - set of of constraint blocks in the system, where $\Theta[\mathbf{M}]$ corresponds to the set of constraints for models, $\Theta[\mathbf{E}]$ - state constraints and $\Theta[\boldsymbol{\Upsilon}]$ function constraints.
- T_{M} - a relation $\mathrm{T} : \mathbf{M} \rightharpoonup \mathbf{M}$ describing a model inheritance.
- $\mathrm{T}_{\mathrm{E},\{\mathrm{E},\mathrm{M}\}}$ - a relation $\mathrm{T} : \mathbf{E} \rightharpoonup \{\mathbf{E}, \mathbf{M}\}$ describing any state transition bounds and encapsulated models in states, that is some state `S'` can only be transitioned to from the specified state `S`, and state some state `S` can encapsulate some model `M`
- $\mathrm{T}_{\Upsilon,\mathrm{E}}$ - a relation $\mathrm{T} : \boldsymbol{\Upsilon} \rightharpoonup \mathbf{E}$ describing any state transition bounds for states $\mathbf{E}$ in functions $\boldsymbol{\Upsilon}$

In particular, $\forall \mu \in \mathbf{M}\ \exists \theta \in \Theta[\mu]$ where θ is a set of constraints for μ, and corresponding logic can be applied for elements of E and Υ.

Then, to verify consistency of the system, we first need to verify the following satisfability *Sat*:

$$
\begin{gathered}
\forall \mu \in \mathbf{M} \\
\exists \theta \in \Theta[\mu] \\
\text{s.t. } \theta = \{c_0, c_1, ..., c_k\} \\
\left(\bigwedge_i c_i \right) \Rightarrow \mathit{Sat}
\end{gathered}
$$

We can define the following check by some functions $\rho(\theta) : \Theta \to \{Sat, Unsat\}$

which yields the following proof:

$$\exists \theta \in \Theta[e]$$
$$\text{s.t. } \theta = \{c_0, c_1, ..., c_k\}$$
$$\left(\bigwedge_i c_i\right) \Rightarrow \textit{Sat or Unsat}$$

This allows to validate the next property of ***VS***

$$A = \{\mathbf{M} \cup \mathbf{E} \cup \Upsilon\}$$
$$A = \{e_0, e_1, ..., e_k\}$$
$$\left(\bigwedge_i \rho(\Theta[e_i])\right) \Rightarrow \textit{Sat or Unsat}$$

The next stage is to verify co-dependent symbols in the system for satisfability of their respective constraints.

Let's look at the models **M**, we want to ensure that

$$\text{if for some } m \in \mathrm{M}, m' \in \mathrm{M}$$
$$\exists (m, m') \in \mathrm{T_M}$$
$$\text{s.t. } \rho(m) \times \rho(m') = (\mathit{Sat}, \mathit{Sat})$$
$$\text{and } \theta = \Theta[m] \cup \Theta[m']$$
$$\rho(\theta) \Rightarrow \mathit{Sat}$$

Very similar verification can applied to $\mathrm{T}_{\Upsilon,\mathrm{E}}$.

For $\mathrm{T}_{\mathrm{E},\{\mathrm{E},\mathrm{M}\}}$, the constraints can be extracted in the following way:

$$\text{if for some } \varepsilon \in \mathrm{E}, \varepsilon' \in \mathrm{E}$$
$$\exists (\varepsilon, \varepsilon') \in \mathrm{T}_{\mathrm{E},\{\mathrm{E},\mathrm{M}\}}$$
$$\text{s.t. } \rho(\varepsilon) \times \rho(\varepsilon') \times \rho(\mu) = (\mathit{Sat}, \mathit{Sat})$$
$$\text{and } \theta = \Theta[\varepsilon] \cup \Theta[\varepsilon']$$
$$\rho(\theta) \Rightarrow \mathit{Sat}$$

Similarly

$$\begin{gathered}\text{if for some } \varepsilon \in \mathrm{E}, \mu \in \mathrm{M} \\ \exists(\varepsilon, \mu) \in \mathrm{T}_{\mathrm{E},\{\mathrm{E},\mathrm{M}\}} \\ \text{s.t. } \rho(\varepsilon) \times \rho(\mu) = (\mathit{Sat}, \mathit{Sat}) \\ \text{and } \theta = \Theta[\varepsilon] \cup \Theta[\mu] \\ \rho(\theta) \Rightarrow \mathit{Sat}\end{gathered}$$

After the completing verification of `Tau` relations for consistency, we can provide a mathematical guarantee that ***VS*** has been modelled consistently.

Having verified the constraints, we can leverage them as the guards during state transions and can apply proofs from *temporal logic* to verify that the described state transitions will take place under the described constraints.

As the final stage, we can perform the symbolic execution of instructions in the function bodies with the constraints loaded in the global context of the system. Having tracked the states of different symbols, we can verify each function for reachability for described state transitions and provide strong guanratees of functional correctness of the system described in the smart contract.