VLSM: A General Coq Framework for Reasoning About Faulty Distributed Systems

Vlad Zamfir, Denisa Diaconescu, Wojciech Kołowski, Brandon Moore, Karl Palmskog, Traian Florin Șerbănuță, and Ioan Teodorescu

Formally modeling and reasoning about distributed systems with faults is a challenging task, not least in Coq. Depending on the system model, an execution of a distributed protocol may be subject to many kinds of faults, from simple recoverable component crashes to Byzantine adversarial actions [2]. A failure may then require recovery actions by the affected components.

To address this problem, Zamfir et al. [3] recently proposed the theory of Validating Labeled State transition and Message production systems (VLSMs) as a general approach to modeling and verifying distributed protocols executing in the presence of faults. In particular, VLSM executions can be subject to *equivocation behavior*. Equivocation refers to claiming different beliefs about the state of the protocol to different parts of the system in order to steer the protocol-following components into making inconsistent decisions; messages received from equivocating components seem to be valid messages.

We present a general VLSM-based framework in Coq for modeling and verification of distributed protocols executing in the presence of faults. The formalized framework was mostly developed in lock-step with the (pen-and-paper) VLSM theory, enabling many constructive feedback loops. Besides an extensive abstract theory of VLSMs, our framework contains many definitions and results relevant to the development of consensus protocols in blockchains.

VLSM Theory and Framework Coq Encoding

We sketch a fragment of the theory of VLSMs and its encoding in our Coq framework.

Definition. *A Validating Labeled State transition and Message production system (*VLSM*, for short) is a structure of the form* $\mathcal{V} = (L, S, S_0, M, M_0, \tau, \beta)$*, where* L *is a set of labels,* $(S_0 \subseteq)$ S *is a non-empty set of (initial) states,* $(M_0 \subseteq)$ M *is a set of (initial) messages,* $\tau : L \times S \times M? \to S \times M?$ *is a transition function which takes as arguments a label, a state, and possibly a message, and outputs a state and possibly a message, while* β *is a validity constraint on the inputs of the transition function.*

A single VLSM can represent the local point of view of a component in a distributed system. We can obtain the global point of view by composing multiple VLSMs and lifting the local validity constraint of each component. Designers of systems can impose more global restrictions, stronger than the ones that can be specified locally on individual components.

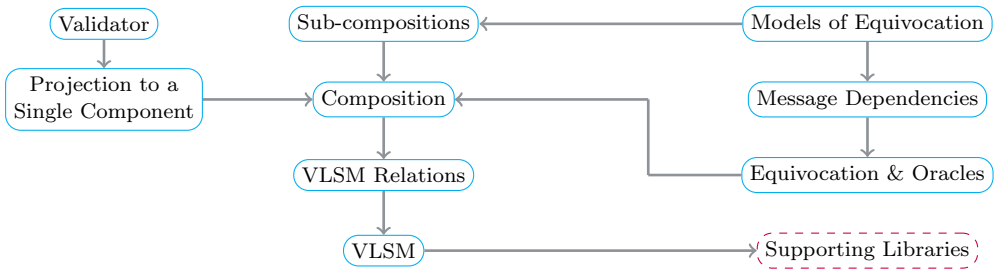
To allow conveniently reasoning about relations between VLSMs sharing messages, states, and labels, the Coq VLSM definition is split into three parts. First, we define a VLSM *type* parameterized by a message type, and then a VLSM *machine* similar to the definition above.

```
Class VLSMType (message : Type) := { state : Type; label : Type }.
Class VLSMMachine {message : Type} (vtype : VLSMType message) : Type :=
{ initial_state_prop : state → Prop;
  initial_state : Type := {s : state | initial_state_prop s};
  s0 : Inhabited initial_state;
  initial_message_prop : message → Prop;
  transition : label → state * option message → state * option message;
  valid : label → state * option message → Prop }.
```

Finally, a VLSM bundles a VLSM machine together with its VLSM type.

```
Record VLSM message := mk_vlsm { vtype : VLSMType message; vmachine : VLSMMachine vtype }.
```

Our Coq framework consists of over 80 Coq modules. To give a flavor of framework contents, we thematically organize modules into groups as illustrated below.



Framework Applications and Availability

We motivate our Coq framework by demonstrating several applications.

Parity example. As a simple running example to introduce the theory, we will use a VLSM that stores a tuple and continually decrements one of its elements while a constraint is checked at each step of the way.

ELMO protocol family. We present the ELMO family of protocols, an ideal message validating distributed system emerging from the Ethereum blockchain community, as VLSMs.

Byzantine fault reduction. We prove that, in the context of a distributed system without synchronization assumptions, the effect that components with Byzantine failures can have on honest validators is no different than the effect equivocating components can have on non-equivocating validators. This shows that equivocation fault tolerance analysis is a viable alternative to Byzantine fault tolerance analysis.

A release of our framework compatible with Coq 8.15 is available on GitHub under a permissive open-source license [1]. The core VLSM definitions and results are around 33,000 lines of code, while its support library consists of around 8,000 lines of code. We rely heavily on the Std++ library and mainly use Coq typeclasses to describe VLSM concepts.

References

[1] M. Calancea, D. Diaconescu, W. Kołowski, E. Li, B. Moore, K. Palmskog, L. Peña, G. Roșu, T. F. Șerbănuță, D. Trufas, and V. Zamfir. VLSM release 1.2, 2022. https://github.com/runtimeverification/vlsm/releases/tag/v1.2.

[2] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[3] V. Zamfir, M. Calancea, D. Diaconescu, W. Kołowski, B. Moore, K. Palmskog, T.F. Şerbănuţă, M. Stay, D. Trufaş, and J. Tušil. Validating labelled state transition and message production systems: A theory for modelling faulty distributed systems. *arXiv*, 2022. https://doi.org/10.48550/ARXIV.2202.12662.