CoqIOA: A Formalization of IO Automata in the Coq Proof Assistant

Anish Athalye
MIT CSAIL

Abstract

Implementing distributed systems correctly is difficult. Designing correct distributed systems protocols is challenging because designs must account for concurrent operation and handle network and machine failures. Implementing these protocols is challenging as well: it is difficult to avoid subtle bugs in implementations of complex protocols. Formal verification is a promising approach to ensuring distributed systems implementations satisfy their specifications, but verification is challenging and time-consuming. Unfortunately, current approaches to mechanically verifying distributed systems in proof assistants using deductive verification do not allow for compositional reasoning, which could greatly reduce the effort required to implement verified distributed systems by enabling reuse of code and proofs.

We present CoqIOA, a framework for reasoning about distributed systems in a compositional way. CoqIOA builds on the theory of input/output automata to support specification, proof, and composition of systems within the proof assistant. The framework's implementation of the theory of IO automata, including refinement, simulation relations, and composition, are all machine-checked in the Coq proof assistant. An evaluation of CoqIOA demonstrates that the framework enables compositional reasoning about distributed systems within the proof assistant.

Introduction

Implementing distributed systems correctly is difficult. Designing protocols for distributed systems is challenging because designs must account for concurrent operation and handle network and machine failures. Furthermore, because distributed systems protocols are complicated, it is difficult to avoid subtle bugs in implementations of these protocols.

Production systems under wide use have had subtle correctness bugs. For example, testing has revealed correctness bugs in releases of popular systems such as Cassandra, Consul, ElasticSearch, etcd, Kafka, MongoDB, and others [1].

Unfortunately, tracking down correctness bugs in distributed systems through testing is time-consuming, and furthermore, it is incomplete. Formal methods provide a much more rigorous means of building high-assurance systems. Deductive verification provides a machine-checkable proof that code satisfies the specification. Theory and tools have advanced in recent years, and researchers have succeeded in building provably correct implementations of realistic distributed systems such as Raft [2] on top of the Verdi framework [3] and a replicated state machine and sharded key-value store using the IronFleet methodology [4].

Problem and goal

Prior work in verifying realistic distributed systems represents impressive engineering effort. Unfortunately, there is no straightforward way to combine individual verified systems into larger services, because prior work is not designed for compositional reasoning.

We define compositional reasoning as follows. With an approach to verification that supports compositional reasoning, we should be able to take individual verified systems, compose them together into a larger service, and easily prove the service correct.

We need compositional reasoning to make verified distributed systems practical, because compositional reasoning enables us to structure code and proofs to manage complexity and reduce programming effort. With regular non-verified distributed systems, programmers separate larger services into individual systems. For example, programmers may use a caching system like Memcached together with a database like MySQL. In a similar manner, we need to be able to build verified services out of individual verified systems.

CoqIOA aims to enable building reusable systems that are independently verifiable in such a way that reasoning about layering and composition reuses proofs of correctness of individual components. Analogous to the previous real-world example, we would want to enable building a verified caching system and a verified database such that we can verify the use of the caching system on top of the database and prove end-to-end correctness. In the general case, we want to be able to compose components of distributed systems that have been proven correct and easily provide end-to-end correctness guarantees for the overall service.

Prior work does not support this kind of compositional reasoning. The IronFleet methodology [4] of layered refinement allows for building individual verified distributed systems. Verdi [3] supports a type of vertical composition through verified system transformers, but Verdi does not support reasoning about services consisting of multiple verified systems.

Approach

We base our approach on the theory of input/output automata [5], a formal model for reasoning about asynchronous concurrent systems, to reason about distributed systems. Systems are specified as automata, and implementations are shown to refine specifications by proving that the behavior of the implementation is a subset of the behavior of the specification. Automata in compositions can be substituted by others that refine the original automata, so that the resultant composition refines the original composition; a specification can be swapped for its implementation while preserving correctness. This is what enables compositional reasoning.

We formalize a theory based on IO automata in a proof assistant to enable machine-checked formal reasoning about distributed systems in a compositional way. Our formalization includes a specification of IO automata, theorems about simulation relations used to prove refinement, and theorems about composition.

Implementation

We implement the CoqIOA prototype entirely in the Coq proof assistant [6]. Only our definitions of IO automata and automata refinement, comprising about 100 lines of code, are trusted. All other components of the framework, about 1000 lines of code, are mechanically verified by Coq's proof checker: all theorems about IO automata, including theorems on simulation relations and composition, are proven correct in Coq.

We have used the CoqIOA framework to reason about toy systems implemented using IO automata. The CoqIOA prototype currently has one major limitation: we do not have a code extraction mechanism to produce executable code from IO automata descriptions. IO automata are specified in a relational manner, and they can be nondeterministic, and so they are not inherently executable. We plan to implement a programmer-assisted translation between IO automata descriptions and executable versions of the automata that provably refine the originals.

Evaluation

We demonstrate that CoqIOA enables compositional reasoning through a case study of a toy key-value store written in 700 lines of Coq code.

In our example, we have a specification of a key-value store modeled as a single automaton, and we have an implementation of a client communicating with a key-value server over channels that reorder messages, which can be seen as a simplified version of machines communicating over a protocol like UDP. The implementation adds send and receive mediators to the channels to make them effectively implement reliable channels (see Figure 1).

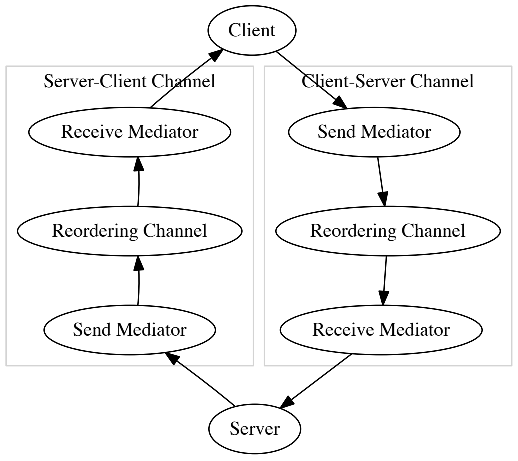

Figure 1. Client-server key-value store implementation over reordering channels.

We prove that the implementation satisfies the specification, and we use compositional reasoning to construct the proof. First, we prove that a mediated reordering channel implements a reliable channel. Next, we prove client communication to a key-value server correct on top of a specification of a reliable channel. Finally, given those two proofs, we use our composition theorems to prove that our system communicating over mediated reordering channels implements a key-value store.

Next, we plan on implementing a more realistic example within the CoqIOA framework: for example, the Memcached-MySQL composition.

Contributions

The main contribution of our work is a methodology for compositional reasoning about distributed systems in a proof assistant as well as CoqIOA, an implementation of this methodology in the Coq proof assistant. Specifically, the contributions of our work are as follows:

1. We formalize input/output automata in the Coq proof assistant, supporting specification, proof, and composition within the proof assistant.
2. We provide machine-checked proofs of the theory of IO automata, including refinement, simulation relations, and composition.
3. We evaluate the effectiveness of our system in enabling compositional reasoning through a case study of a toy system.

References

[1] K. Kingsbury, "Jepsen analyses." `http://jepsen.io/analyses`.

[2] D. Woos, J. R. Wilcox, S. Anton, Z. Tatlock, M. D. Ernst, and T. Anderson, "Planning for change in a formal verification of the Raft consensus protocol," in *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, (New York, NY, USA), pp. 154–165, ACM, 2016.

[3] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, "Verdi: A framework for implementing and formally verifying distributed systems," in *PLDI 2015: Proceedings of the ACM SIGPLAN 2015 Conference on Programming Language Design and Implementation*, (Portland, OR, USA), pp. 357–368, June 2015.

[4] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, "IronFleet: Proving practical distributed systems correct," in *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, (New York, NY, USA), pp. 1–17, ACM, 2015.

[5] N. A. Lynch and M. R. Tuttle, "An introduction to input/output automata," *CWI Quarterly*, vol. 2, pp. 219–246, 1989.

[6] Coq development team, *Coq Reference Manual, Version 8.6*. INRIA, Dec. 2016. `https://coq.inria.fr/distrib/V8.6/refman/`.