Proving correctness of a dynamic atomicity analysis in Coq

Caitlin Sadowski Jaeheon Yi Kenneth Knowles Cormac Flanagan
University of California at Santa Cruz
caitlin/jaeheon/kknowles/cormac@cs.ucsc.edu

Writing and reasoning about concurrent programs remains notoriously difficult despite the proliferation of type systems, static analyses, and dynamic analyses targeting concurrent programs. There are examples of verified developments for concurrent languages and programs (Chou and Peled 1996; Affeldt and Kobayashi 2004; Feng et al. 2007; Hobor et al. 2008) but most analyses – especially dynamic analyses – have not been subjected to mechanical rigor. We report on our partial mechanization in Coq of the recently-released Velodrome dynamic atomicity checker (Flanagan et al. 2008).

Velodrome examines a trace of a program to ensure that its atomic blocks are serializable. We specify the Velodrome analysis using a multi-threaded operational semantics with a collection of invariants relating the state of the analysis data structures to properties of the program trace. On top of an axiomatization of well-known facts about relations and program traces, we prove that Velodrome accurately reconstructs the *transactional happens-before* relation, the quotient of the usual happens-before relation where operations in the same transaction are identified. Although this proof is still a work in progress, we were able to identify a bug in the paper proof through the formalization process. An overview of our architecture is shown in Figure 1. Building on some utility code of Aydemir et al. (2008) we have written about 200 lemmas, 100 definitions, and a bit over 4000 lines of Coq code.

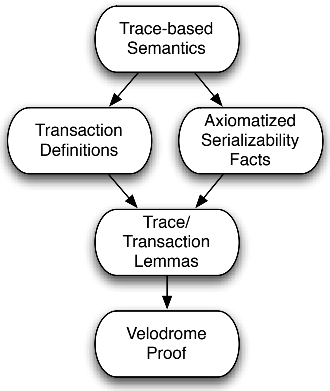

Figure 1. Module Organization

In contrast to work on calculi where binding structure is the primary challenge, the most difficult part of our language-agnostic trace-based approach is the complexity of run-time invariants, which are more involved than type preservation. As an example of the simplest of our invariants, consider the following

```
Definition inv3
  (alpha:trace) (phi:analysis_state) : Prop :=
    forall C L U R W H x, phi = state C L U R W H ->
    stores_recent W (is_write_op x) alpha x.
```

This states that the `W` component of the analysis state stores, for each variable `x` and each thread, the ID of the most recent transaction to write to `x`. We have five such invariants using the higher-order predicate `stores_recent`: One each for the most recent read, write, lock release, transaction begin, and transaction commit operations.

The more complex invariants relate the `H` component of the analysis state, which is a compact representation of the transactional happens-before relation, to the actual happens-before relation. The following invariant states that if operation `a` happens before `b` then the transaction of `a` is reachable from the transaction of `b` in the graph represented by `H`.

```
Definition inv7
  (alpha:trace) (phi:analysis_state) : Prop :=
    forall C L U R W H  a b ta tb,
    phi = state C L U R W H ->
    happens_before_intransitive alpha a b ->
    trans alpha a = Some ta ->
    trans alpha b = Some tb ->
    ta <> tb ->
    reachable H ta tb .
```

In all, there are seven invariants over six operations. Of these 42 cases, 15 are trivial, and five of the proofs remain unfinished. Through many iterations, we have distilled an expression of these invariants that lends itself well to mechanized proof, and we hope to share our experience and receive feedback on how to improve them further. In addition, we intend to organize our collection of definitions and lemmas into a reusable library for trace-based analyses, shaped by interaction with other researchers interested in mechanizing proofs of their concurrent programs and analyses.

A current snapshot of our code is available at
`http://slang.soe.ucsc.edu/velodrome-coq.tar.gz`

References

R. Affeldt and N. Kobayashi. A Coq library for verification of concurrent programs. In *LFM*, pages 17–32, 2004.

B. Aydemir, A. Bohannon, B. Pierce, J. Vaughan, D. Vytiniotis, S. Weirich, and S. Zdancewic. Using proof assistants for programming language research or, how to write your next popl paper in coq, 2008. `http://www.cis.upenn.edu/~plclub/popl08-tutorial/`.

C.-T. Chou and D. Peled. Formal verification of a partial-order reduction technique for model checking. In *TACAS*, pages 241–257, 1996.

X. Feng, R. Ferreira, and Z. Shao. On the relationship between concurrent separation logic and assume-guarantee reasoning. In *ESOP*, pages 173 – 188, 2007.

C. Flanagan, S. N. Freund, and J. Yi. Velodrome: A sound and complete dynamic atomicity checker for multithreaded programs. In *PLDI*, 2008.

A. Hobor, A. W. Appel, and F. Z. Nardelli. Oracle semantics for concurrent separation logic. In *ESOP*, 2008.