Verdi:A Framework for Implementing and Formally Verifying Distributed Systems.

Wilcox et al. PLDI 2015

Verdi, unpacked

- What are the languages / stack / process?
  - Coq : *not* an automated theorem prover
- Need to provide (dependently types functional programs):
  - A specification (formula): safety property (over state)
  - An implementation: “Coq language”
  - A manual proof (in Coq theorem prover that implementation satisfies the specification), possibly using tactics (proof ~ type checking)
- Coq extracts to *OCaml* (functional) dialect of ML, which can be compiled, and deployed/executed

Verdi, the guarantees

- At end of day: *get a system that is fault tolerant*
  - Verdi adds fault tolerance automagically
- "Formally verifying" = implementation consistent with spec (satisfies the safety property, in a mathematical sense)
- Axioms/Assuming ("what am I trusting to get above?"): trust Coq, extraction, trust OCaml compiler, trust Verdi shims (OCaml piece to provide I/O)
- Trust their network semantics!

Verdi, the point

- Who doesn't want *correctness*? Distributed systems are difficult, let's prove them correct!
- Contribution (reusable Coq parts)
- Formalize network semantics (good to have!)
- Modularity of semantics that can be layered, with little effort on the human side
- Output code that runs! (This is a recent trend in verification: get a real system at end of process). *SEL4 for OS (20 person years)*

Verdi

- Network semantics.. are these right?? (Do they reflect reality?)
- Disjoint semantics.. but doesn't transformer solves this? Composability - layer these semantical layers on top of one another.
    - S1 -> S2 -> S3 (to satisfy both msg duplication, and dropping)
- *Abstractions*: "bag of packets for in-flight", "abstract data types, not bits", "no modelling of time" ~ small-step semantics, "set of failing nodes", "nodes atomically transition to/ from failure"
    - Real systems send buffers of bits, they contains real-time timeouts, packet reordering occurs on nearby packets, node failures take time (to detect and to occur)
    - Solid attempt! Mismatches reality, because reality != math
    - *Real q: is this the right level at which to stop modelling distributed systems? (A domain expert expert — i.e., distributed systems engineers)*
- (Finn: fyi, integers are a *tree* in Coq)

Verdi transformers!

- VST: *<u>for free</u>* transform model written for 1 net semantics into model written for a different semantics && transform the property into an updated property && automates the process of proving the updated property based on original proof
    - Not having to re-prove is a huge win!
- Counter-intuitive: Go from stronger semantics (reliable delivery) to weaker semantics (drops && re-orderings) for free.
    - Builds on the intuition that you make up for weaker network semantics (dozens of years of experimental evidence)
    - Cherry on top: do this automatically && retain the proof
- Two types of failures — network, node
    - Primary-backup VST: magic automatic replication (to a single node)
    - Raft replication VST = consensus VST: **<u>SMR for free</u>** with linearalizability guarantees
        - First paper to show linearalizability for Raft (Paxos same)

OCaml

- General purpose PL
- Not generally used for (distributed) systems
- OCaml ~ Haskell: same ballpark, high abstraction, maintained by INRIA?
  - Academically focused
  - Biggest project using OCaml is Coq (Junfeng)
  - JaneStreet hires OCaml devs (trick to hire smart people)

Linearalizability

- Sequential consistency (serializability) weaker than Lin.
- R/W ops on an object
- (Section 7.2) Operation O corresponding to a request that arrives at time T cannot be ordered before any operation that was already made visible to a client prior to T.

Verdi eval

- Table 2 VST numbers are *additional* LOC (on top of existing spec/impl that the developer provided)
- Metrics
  - Proof effort (LOC), but no comparison and no time estimate *(person years)*
  - Proof v. Spec LOC comparison
  - Leave out proof time for their proofs
  - Throughput and latency against etcd (open source KV store)
    - Ballpark numbers, demonstrates “not inefficient”?
    - Etcd is much more feature-full, production-level, different lang
    - 100 reqs sent (total!?) ~ 3s of runtime …. Not a great eval to demonstrate perf.

Verdi v. Mace PL

- Both aim to reduce effort
    - Both introduce abstractions! (Both PLDI papers)
        - Objects/aspects/layers in Mace
        - Network semantics and VST in Verdi
- Verdi focus on composability (for proof!). Mace is focused on layered architecture (composability?).
- Verdi for verification (debug your model)
    - Proof that the things it generates are correct (Coq)
    - High effort
- Mace restricts designers to a structure that helps with reasoning (leads to potential tools to support development, like model checking, logging, causal tracing).
    - There is no proof that it actually helps. Just anecdotal evidence.
    - Lower effort than Verdi (but perhaps higher than typical C++)
    - Expressiveness
- Granularity: Mace is fine-grained and Verdi is coarse grained (netw. semantics swap in/out as a module)

Next: MODIST

- MODIST is a blackbox model checker: ambitious!
- What's the trade-off with Verdi+Mace in MODIST: guarantees / effort / other?
- NSDI 2009 (a top networked systems conference)
- If you know Vaastav's (MSc grad) *Dara* work: it's inspired by MODIST