|galois|

Formal Verification of Production Distributed Protocols

Mike Dodds, Giuliano Losa

HCSS, September 2020

Distributed protocols are important

Distributed protocols are becoming widespread, but we’re a long way from routinely verifying them.

Here are four distinctive characteristics of these systems:

- Distributed, asynchronous control and computation
- Collaboration between nodes
- Key objectives are collective, not single-node
- Nodes and networks may be faulty / adversarial

These properties hold most obviously for cloud services, but also other categories such as cyber-physical systems.

We need formal methods to increase our confidence in these systems.

Galois is verifying blockchain protocols

Galois' work on distributed protocols has focused on blockchain.

Blockchain protocols are a great domain to refine verification techniques.

That's because:

- Researchers are rapidly developing new blockchain protocols.
- Blockchains suffer from severe assurance problems.
- Blockchain companies will pay for assurance, *if it works.*

Introducing the Stellar Consensus Protocol

The purpose of a consensus protocol is to achieve agreement between nodes. For example, what’s the next block in the blockchain?

Open blockchains must prevent adversaries from flooding the system with fake participants. Bitcoin and Ethereum solve this by raising the cost of participation - ‘proof-of-work’ / ‘proof-of-stake’.

The Stellar Consensus Protocol (SCP) solves this problem by letting participants define its own trust relationships. As a result, different parties can come to different conclusions.

SCP embodies networks with flexible notions of trust.

Understanding the SCP approach

Each node defines a set of slices, representing a trusted set.

A quorum for node *n* is some set *Q* such that:

- Some slice of *n* is included in *Q*; and
- each member of *Q* also has a slice included in *Q*.

As a result, each node has a personal set of quorums.

This has some surprising consequences:

- Quorums are not global to the system.
- A node's quorums depend on other nodes declared slices.
- Some nodes may not share enough trust to reach consensus.

Verification objectives

We want to prove two properties for mutually trusting nodes:

- Safety - nodes that reach agreement can't disagree on a decision
- Liveness - nodes can eventually reach agreement.

We also have two requirements for the proof *method*.

First, we want to verify the protocol for any size and execution length. This is because the space of possibilities for a distributed protocol is highly complex.

Second, we need a tool that helps engineers be productive when writing a proof. This is because formal methods must compete with many other assurance techniques.

The Ivy verification tool

Ivy is a compromise between two sorts of tool.

One type of tool: lightweight modelling languages like TLA+ and Alloy.

- Advantage: very easy to develop and experiment with models
- Disadvantage: can only examine small model instances.

The other: theorem provers like Coq and Isabelle:

- Advantage: very powerful, almost anything can be proved.
- Disadvantage: very expensive proofs, very limited automation.

Ivy tries to take a middle path between these two sorts of tool.

Productivity and automation in proofs

For automation to help with proofs, we need several properties:

- Predictability of performance.
- Stability / continuity: small changes don’t affect performance.
- Transparency: failures have understandable explanations.

Whether this holds depends on the characteristics of the logical queries.

Ivy restricts queries to decidable logic. This means a solver is guaranteed to answer whether a logical formula is true.

Decidablility severely restricts the kinds of proofs that can be written. But fortunately many important proofs can still be expressed.

The toy-agreement protocol

The Stellar Consensus Protocol is too complex to explain here.
Instead, I'll show you the toy-agreement protocol.

Protocol:

- Each node nondeterministically chooses and broadcasts a value.
- Nodes reach agreement if some strict majority picks the value.

Properties:

- Nodes won't necessarily reach agreement.
- No two nodes can decide on a different value (safety).
- If a majority picks the same value, all nodes reach a decision (liveness)

Specifying toy-agreement in Ivy

```
type node
type value
type quorum
relation member(N:node, Q:quorum)
```

We represent quorums through an explicit relation, not as sets.

```
axiom is_quorum ∀ Q1,Q2 . ∃N . member(N,Q1) & member(N,Q2)
```

This axiom is weaker than the definition of a majority. It’s sufficient to verify the protocol, but can be encoded into decidable logic.

Encoding properties into decidable logic

Every part of the model has to be expressed in decidable logic. The fragment I'll talk about is called EPR.

EPR is very restrictive:

- Constructs of propositional logic - and / or / negation.
- Universal and existential quantification over elements - ∀, ∃
- No built in theories - no arithmetic / bit-vectors / arrays
- Restrictions on how quantifiers can alternate in a formula.

This is why we wrote the domain model as we did:

- We can't quantify over a set, but we can create an entity representing a set.
- We don't have arithmetic to define a majority, but we can define an axiom about overlapping elements in a set.

Modelling the toy-agreement protocol (1)

```
relation proposal(N:node,V:value)
relation received(N:node,M:node,V:value)
relation decision(N:node,V:value)
```

These relations represent the protocol state and change as the protocol executes.

```
after init {
  proposal(N,V) := false;
  received(N,M,V) := false;
  decision(N,V) := false;
}
```

We initialize all these relations to be empty.

Modelling the toy-agreement protocol (2)

```
action propose(n:node, v:value) = {
    assume ¬proposal(n,V); # never proposed
    proposal(n,v) := true;
}

action receive(n:node, m:node, v:value) = {
    assume proposal(m,v); # node m proposed v
    received(n,m,v) := true;
    if ∃ Q. ∀ N. member(N,Q) -> received(n,N,v) { # check quorum
        decision(n,v) := true;
    }
}

∀N1,N2,V1,V2 . decision(N1,V1) & decision(N2,V2) -> V1 = V2
```

Modelling the toy-agreement protocol (2)

```
action propose(n:node, v:value) = {
    assume ¬proposal(n,V); # never proposed
    proposal(n,v) := true;
}

action receive(n:node, m:node, v:value) = {
    assume proposal(m,v); # node m proposed v
    received(n,m,v) := true;
    if ∃ Q. ∀ N. member(N,Q) -> received(n,N,v) { # check quorum
        decision(n,v) := true;
    }
}

∀N1,N2,V1,V2 . decision(N1,V1) & decision(N2,V2) -> V1 = V2
```

Modelling the toy-agreement protocol (2)

```
action propose(n:node, v:value) = {
    assume ¬proposal(n,V); # never proposed
    proposal(n,v) := true;
}

action receive(n:node, m:node, v:value) = {
    assume proposal(m,v); # node m proposed v
    received(n,m,v) := true;
    if ∃ Q. ∀ N. member(N,Q) -> received(n,N,v) { # check quorum
        decision(n,v) := true;
    }
}
```

```
∀N1,N2,V1,V2 . decision(N1,V1) & decision(N2,V2) -> V1 = V2
```

Proving correctness in Ivy

We prove safety using an inductive invariant, I:

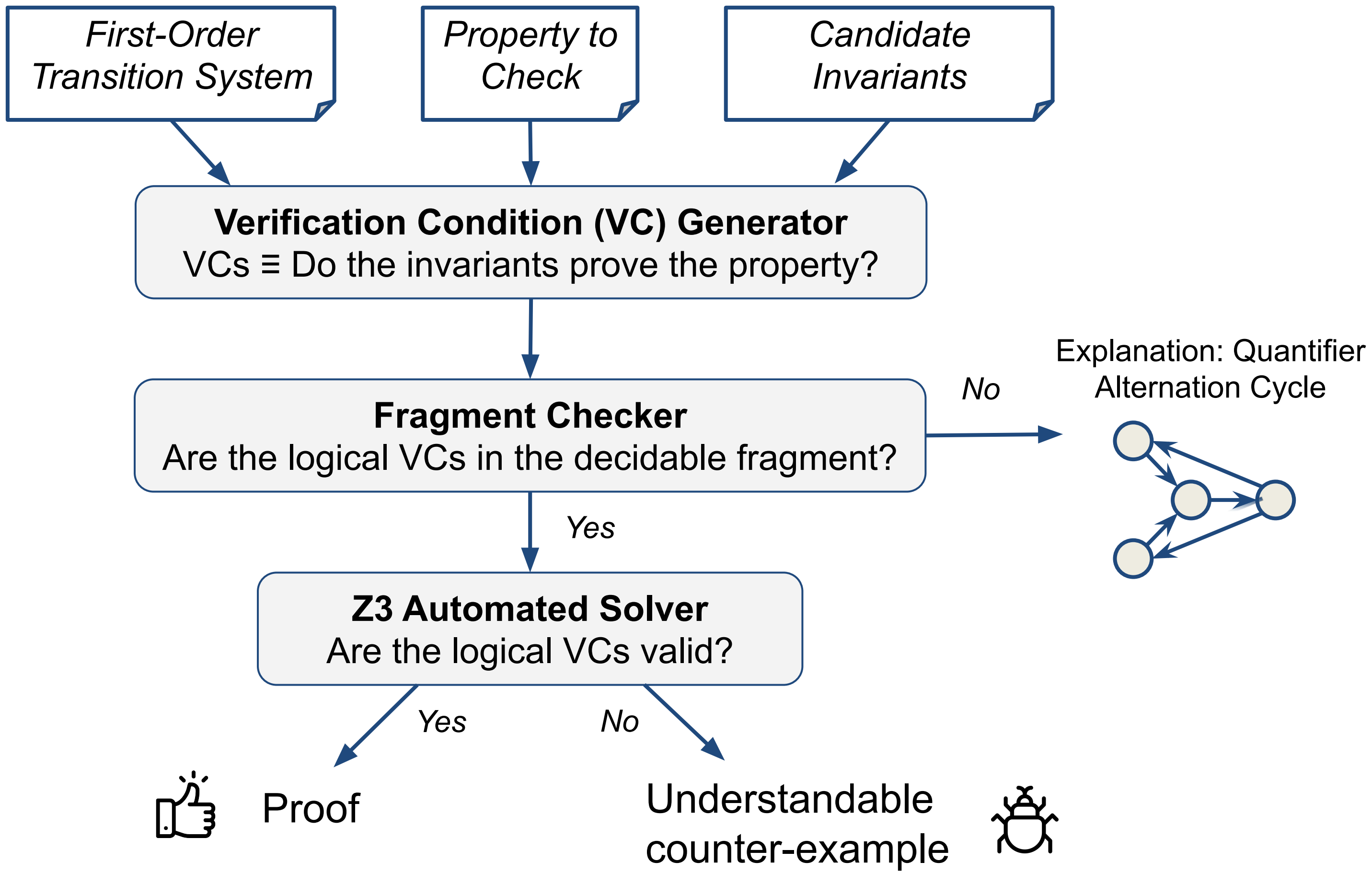
- The initial state of the system satisfies I
- Any transition from a state in I, reaches a state also in I

```
invariant ∀ N,V1,V2. proposal(N,V1) & proposal(N,V2) -> V1 = V2

invariant ∀ N,M,V. received(N,M,V) -> proposal(M,V)

invariant ∀ N1,V. (exists N1 . decision(N1,V))
       -> exists Q . forall N . member(N,Q) -> proposal(N,V)
```

The verification process in Ivy


First-Order Transition System
Property to Check
Candidate Invariants
Verification Condition (VC) Generator
VCs ≡ Do the invariants prove the property?
Fragment Checker
Are the logical VCs in the decidable fragment?
No
Explanation: Quantifier Alternation Cycle
Yes
Z3 Automated Solver
Are the logical VCs valid?
Yes
No
Proof
Understandable counter-example

Verifying the SCP protocol

We applied the same Ivy verification process to SCP.

The main difficulty lay in defining the quorum axioms. Recall for toy agreement we had this:

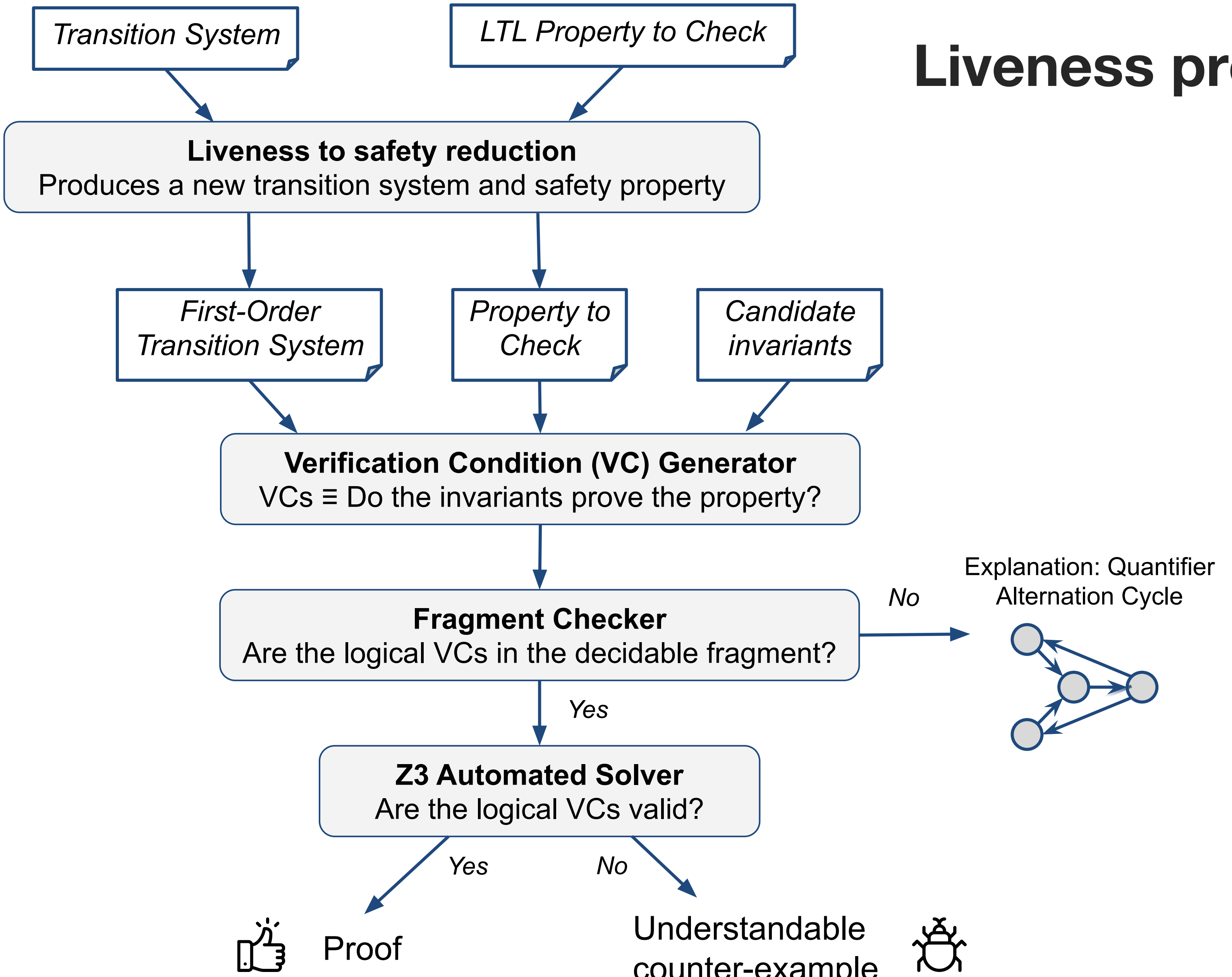
```
    axiom is_quorum ∀ Q1,Q2 . ∃N . member(N,Q1) & member(N,Q2)
```

For SCP, quorums are more dynamic, so we needed a more complex set of axioms.

Two-step process:

- We defined axioms in decidable logic and then verified the protocol in Ivy.
- We modelled the protocol in Isabelle and verified the axioms

Liveness proof


Transition System
LTL Property to Check
Liveness to safety reduction
Produces a new transition system and safety property
First-Order Transition System
Property to Check
Candidate invariants
Verification Condition (VC) Generator
VCs ≡ Do the invariants prove the property?
Fragment Checker
Are the logical VCs in the decidable fragment?
No
Explanation: Quantifier Alternation Cycle
Yes
Z3 Automated Solver
Are the logical VCs valid?
Yes
No
Proof
Understandable counter-example

Properties of our proof

The model of SCP consists of 150 lines of Ivy code

The safety proof took about a dozen invariants

The liveness proof took approximately 50 invariants. Liveness took about a month to verify - the majority of our proof effort

For more details, see our recent FMBC workshop paper:

https://losa.fr/research/fmbc/camera-ready.pdf

Closing: choosing the right tool

galois

miked@galois.com
giuliano@galois.com