Formal Verification of the Stellar Consensus Protocol

Giuliano Losa

UCLA

giuliano@cs.ucla.edu

www.losa.fr

Goals

- Formal specification of SCP
  - A formal version of the Internet Draft

Goals

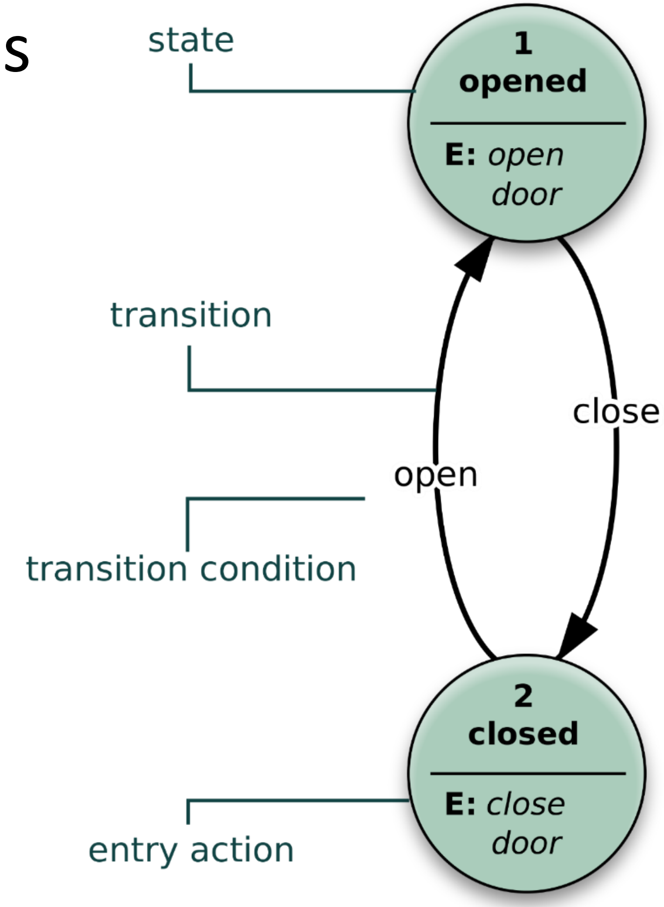
- Formal specification of SCP
  - A formal version of the Internet Draft
- Formal proofs that the SCP specification satisfies its intended properties

Goals

- Formal specification of SCP
  - A formal version of the Internet Draft
- Formal proofs that the SCP specification satisfies its intended properties
- Formally verified implementation

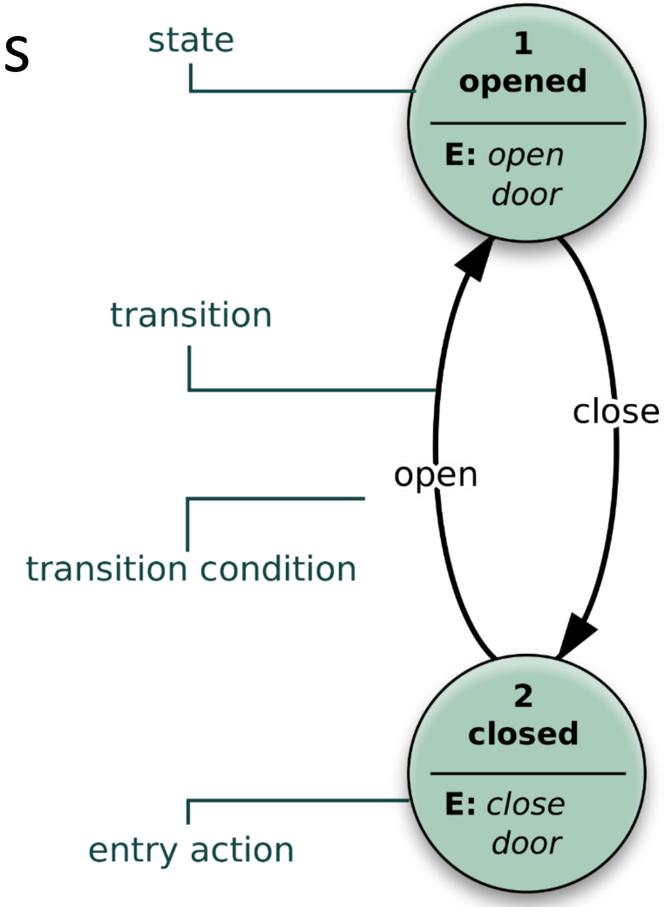
What is a formal specification?

- An abstract machine with states and transitions that specifies allowed behaviors



What is a formal specification?

- An abstract machine with states and transitions that specifies allowed behaviors
- A specification looks like a program, but
  - Has precise meaning
  - Is written for clarity
  - Specifies an envelope of allowed behaviors, leaving room for implementation choices



Why specify formally?

- Unambiguous protocol description
  - Given an API call trace, it is clear whether it satisfies the spec or not
- Advantages:
  - Communication between protocol designer and implementer: avoids interpretation errors
  - Can be used as test oracle
  - Intended properties of the specification can be formally verified
  - Can be used to formally verify implementations

Excerpts from the SCP specification in IVy

```
type statement = {commit, abort}
relation vote(V:node, B:ballot, S:statement)
relation accept(V:node, B:ballot, S:statement)
relation confirm(V:node, B:ballot, S:statement)
```

Excerpts from the SCP specification in IVy

```
type statement = {commit, abort}
relation vote(V:node, B:ballot, S:statement)
relation accept(V:node, B:ballot, S:statement)
relation confirm(V:node, B:ballot, S:statement)

action vote_commit(v:node, b:ballot) = {
       require b.n > 0;
       require forall C . C < b & C.x ≠ b.x -> confirm(v, C, abort);
       vote(v, b, commit) := true;
}
```

Excerpts from the SCP specification in IVy

```
type statement = {commit, abort}
relation vote(V:node, B:ballot, S:statement)
relation accept(V:node, B:ballot, S:statement)
relation confirm(V:node, B:ballot, S:statement)

action vote_commit(v:node, b:ballot) = {
       require b.n > 0;
       require forall C . C < b & C.x ≠ b.x -> confirm(v, C, abort);
       vote(v, b, commit) := true;
}

action confirm(v:node, b:ballot, s:statement, q:nodeset) = {
       require is_quorum(q);
       require forall V . member(V,q) -> accept(V, b, s);
       confirm(v, b, s) := true;
}
```

But, is the specification correct?

But, is the specification correct?

A formal proof would ensure that all possible executions of the specification satisfy its intended properties

But, is the specification correct?

A formal proof would ensure that all possible executions of the specification satisfy its intended properties

For SCP:

But, is the specification correct?

A formal proof would ensure that <u>all possible executions</u> of the specification satisfy its intended properties

For SCP:

- Definitions: the quorums of <u>intertwined</u> nodes intersect at well-behaved nodes; <u>intact</u> nodes are intertwined nodes that are part of a quorum consisting only of intact nodes.

But, is the specification correct?

A formal proof would ensure that all possible executions of the specification satisfy its intended properties

For SCP:

- Definitions: the quorums of intertwined nodes intersect at well-behaved nodes; intact nodes are intertwined nodes that are part of a quorum consisting only of intact nodes.
- SCP is Safe: no two intertwined nodes externalize different values for the same slot

But, is the specification correct?

A formal proof would ensure that all possible executions of the specification satisfy its intended properties

For SCP:

- Definitions: the quorums of intertwined nodes intersect at well-behaved nodes; intact nodes are intertwined nodes that are part of a quorum consisting only of intact nodes.
- SCP is Safe: no two intertwined nodes externalize different values for the same slot
- SCP is non-blocking: intact nodes always remain able to externalize a value

Why prove formally?

Why prove formally?

Distributed protocol are notoriously hard to get right

Informal prose arguments do not suffice

~13000 citations

SIGCOMM 2001

Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Member, IEEE*

Attractive features of Chord include its simplicity, provable correctness, and provable performance even in the face of concurrent node arrivals and departures. It continues to func-

~13000 citations

SIGCOMM 2001

Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Member, IEEE*

Attractive features of Chord include its simplicity, provable correctness, and provable performance even in the face of concurrent node arrivals and departures. ...

CCR 2012

Using Lightweight Modeling To Understand Chord

Pamela Zave
AT&T Laboratories—Research
Florham Park, New Jersey USA
pamela@research.att.com

Under the same assumptions made in the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. The results are presented by means of counterexamples to the invariants in Section 4. In preparation for the results, Section 2 gives a

Are formal proofs a realistic goal?

Yes; complex systems (even implementations) have been formally proved correct:

- CompCert: C compiler
- seL4: Hypervisor
- Project Everest: cryptography in Firefox
- GRAT toolchain: SAT solver
- FSCQ: journaling file system
- and many other examples…

What is a formal proof?

- Like a mathematician’s proof, but much more detailed
- Machine-checked

What is a formal proof?

- Like a mathematician's proof, but much more detailed
- Machine-checked

Proof of:

$$(A \to B) \land (B \to C) \to (A \to C)$$

What is a formal proof?

- Like a mathematician’s proof, but much more detailed
- Machine-checked

Proof of:

$$(A \to B) \land (B \to C) \to (A \to C)$$

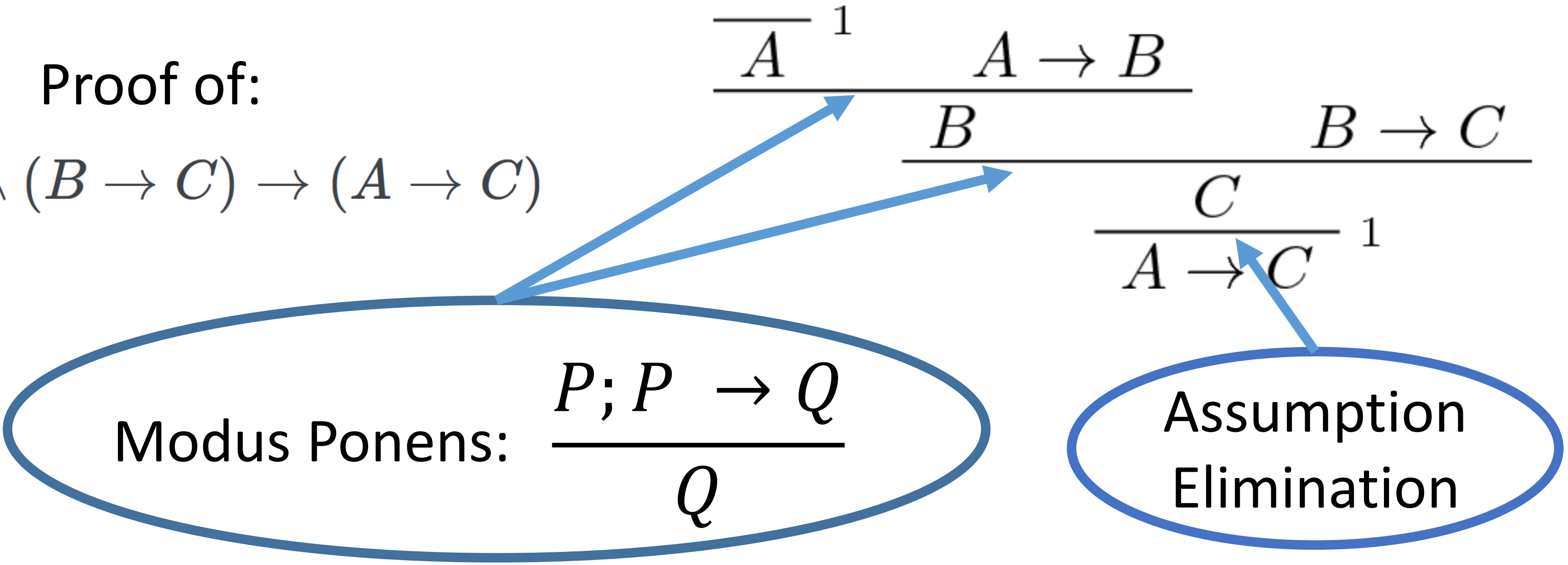
$$\dfrac{\dfrac{\dfrac{\dfrac{}{A}\,{}^{1} \qquad A \to B}{B} \qquad B \to C}{C}}{A \to C}\,{}^{1}$$

What is a formal proof?

- Like a mathematician's proof, but much more detailed
- Machine-checked

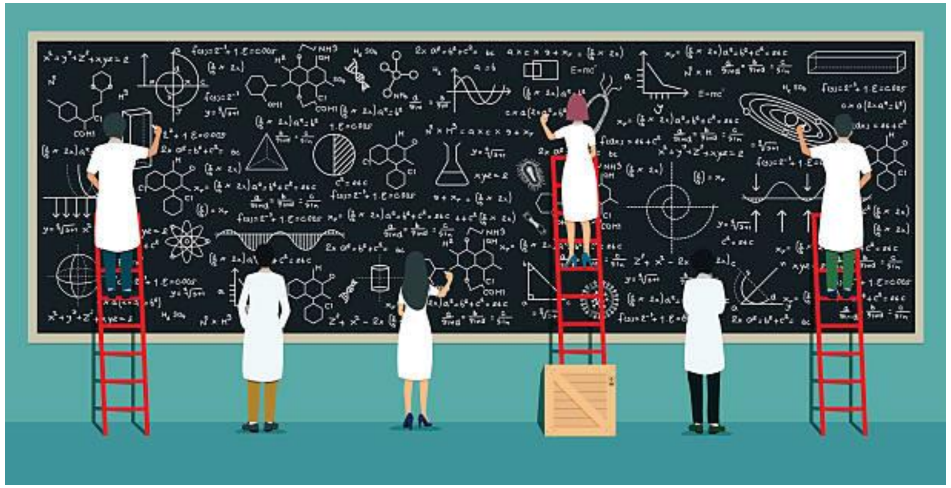
Proof of:

$$(A \to B) \land (B \to C) \to (A \to C)$$



Proving from first principles is hard

Example: safety proof of Raft implementation with Verdi: 50 000 lines of proof for 500 lines of code



Woos, Doug, et al. "Planning for change in a formal verification of the Raft consensus protocol." *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. ACM, 2016.

Proving from first principles is hard

Example: safety proof of Raft implementation with Verdi:
50 000 lines of proof for 500 lines of code

Woos, Doug, et al. "Planning for change in a formal verification of the Raft consensus protocol." *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. ACM, 2016.

Proving from first principles is hard

Example: safety proof of Raft implementation with Verdi:
50 000 lines of proof for 500 lines of code

Automated Solvers

Woos, Doug, et al. "Planning for change in a formal verification of the Raft consensus protocol." *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. ACM, 2016.

Proving from first principles is hard

Example: safety proof of Raft implementation with Verdi:
50 000 lines of proof for 500 lines of code

Automated Solvers

Z3

(and (or (and (= x0 y0) (= y0 x1)) ... z0 ... y1 ... (= ... z1 ... y2 x3)) (and (= x2 z2) ... z2 x3))) (not (= x0 x3)))

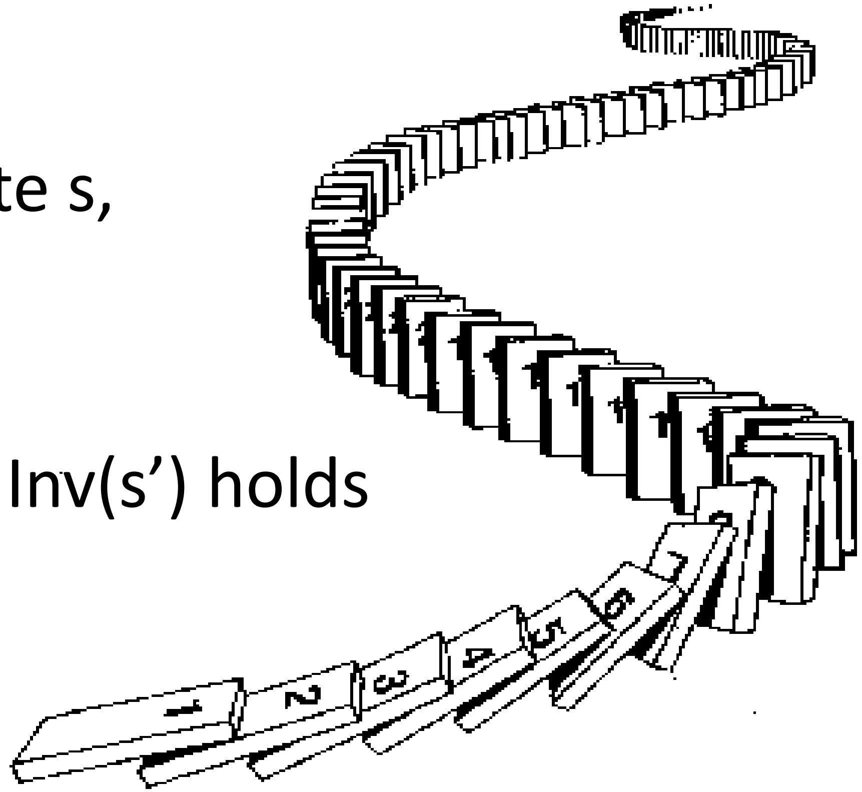
CVC4

SPASS

Amazing tools, but that can still fail…

Woos, Doug, et al. "Planning for change in a formal verification of the Raft consensus protocol." *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. ACM, 2016.

Inductive Invariants



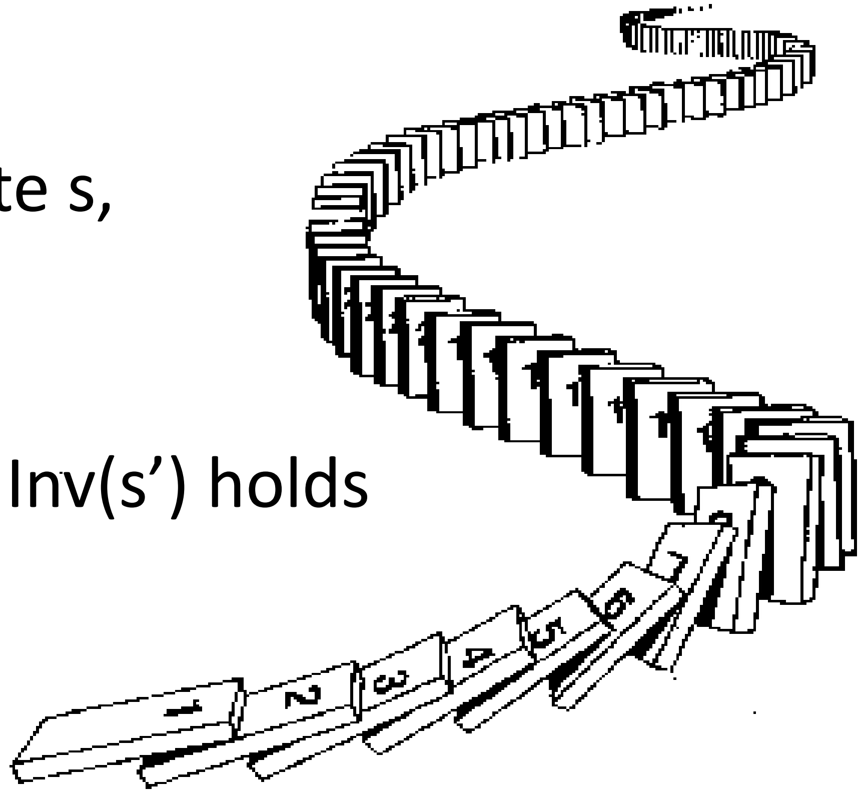
To prove that P(s) holds for every reachable state s,
find predicate Inv(s) such that:

1. Initiation: Inv(s_0) holds in the initial state s_0
2. Consecution: If Inv(s) holds and s → s', then Inv(s') holds
3. Safety: Inv(s) implies P(s)

Inductive Invariants

To prove that P(s) holds for every reachable state s,
find predicate Inv(s) such that:

1. Initiation: Inv(s_0) holds in the initial state s_0
2. Consecution: If Inv(s) holds and s → s', then Inv(s') holds
3. Safety: Inv(s) implies P(s)

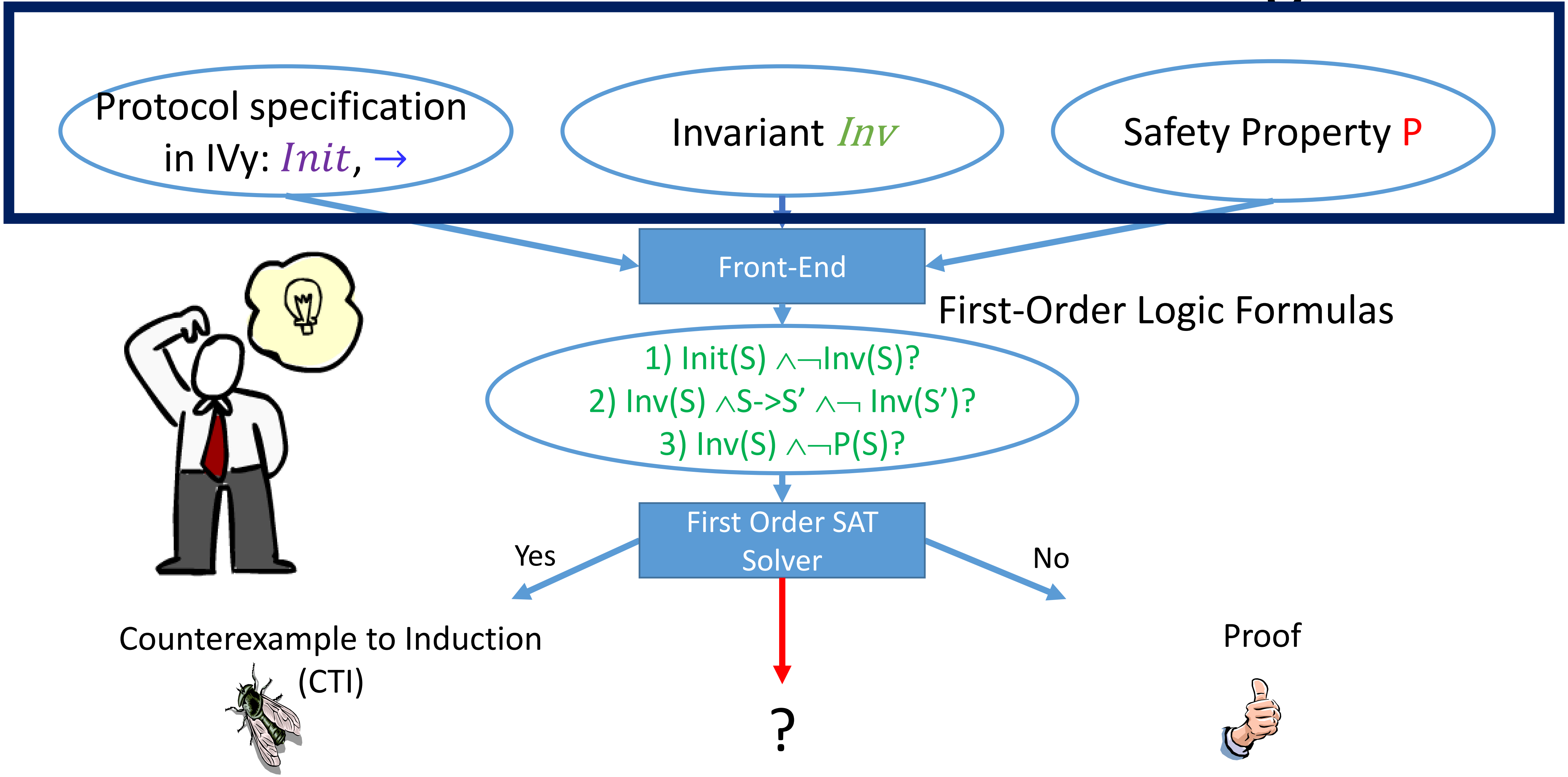


This is just proof by induction!

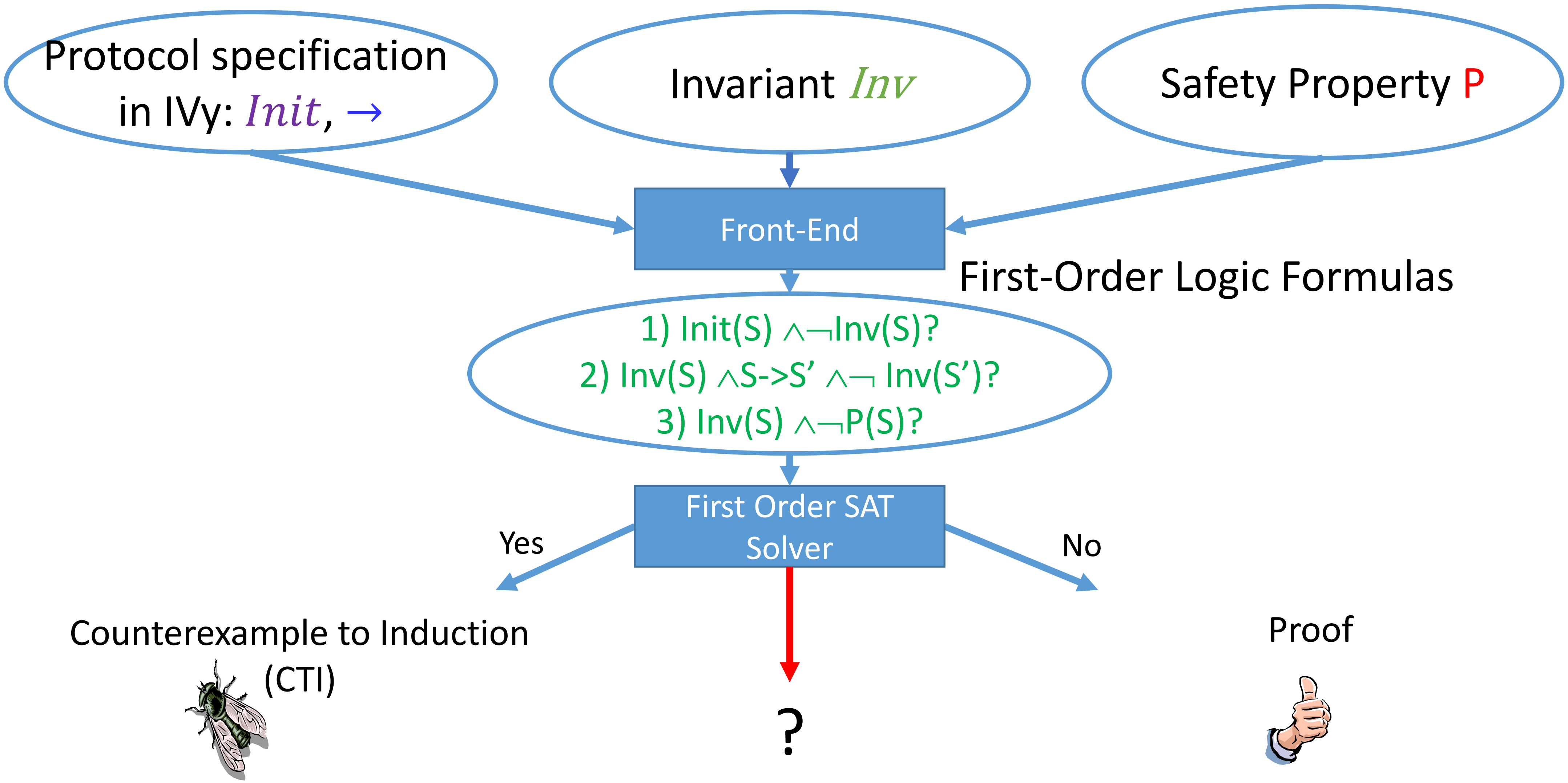
Deductive Verification

- The human provides insight in the form of an inductive invariant
- The automated prover “crunches the numbers” and automatically checks initiation, consecution, and safety

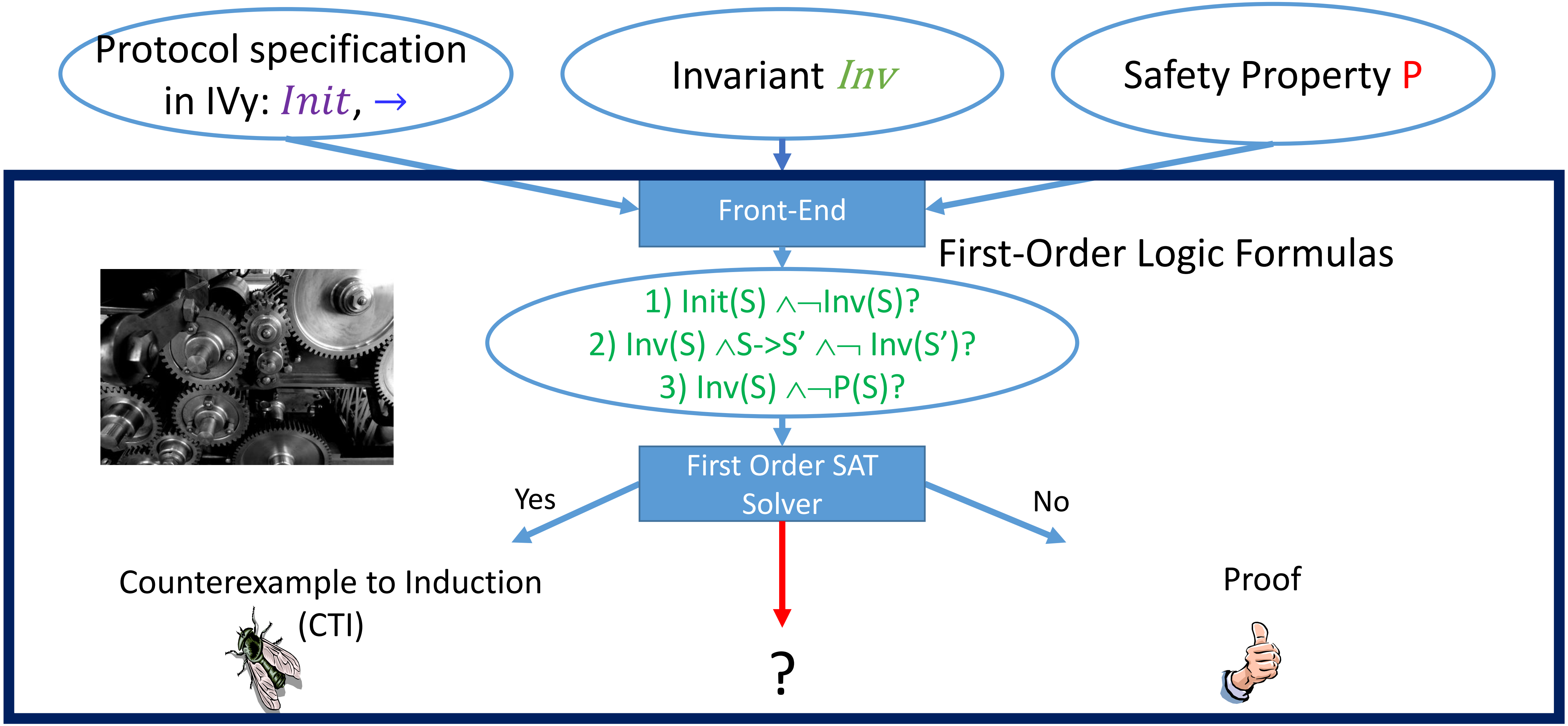
Deductive verification in First-Order Logic



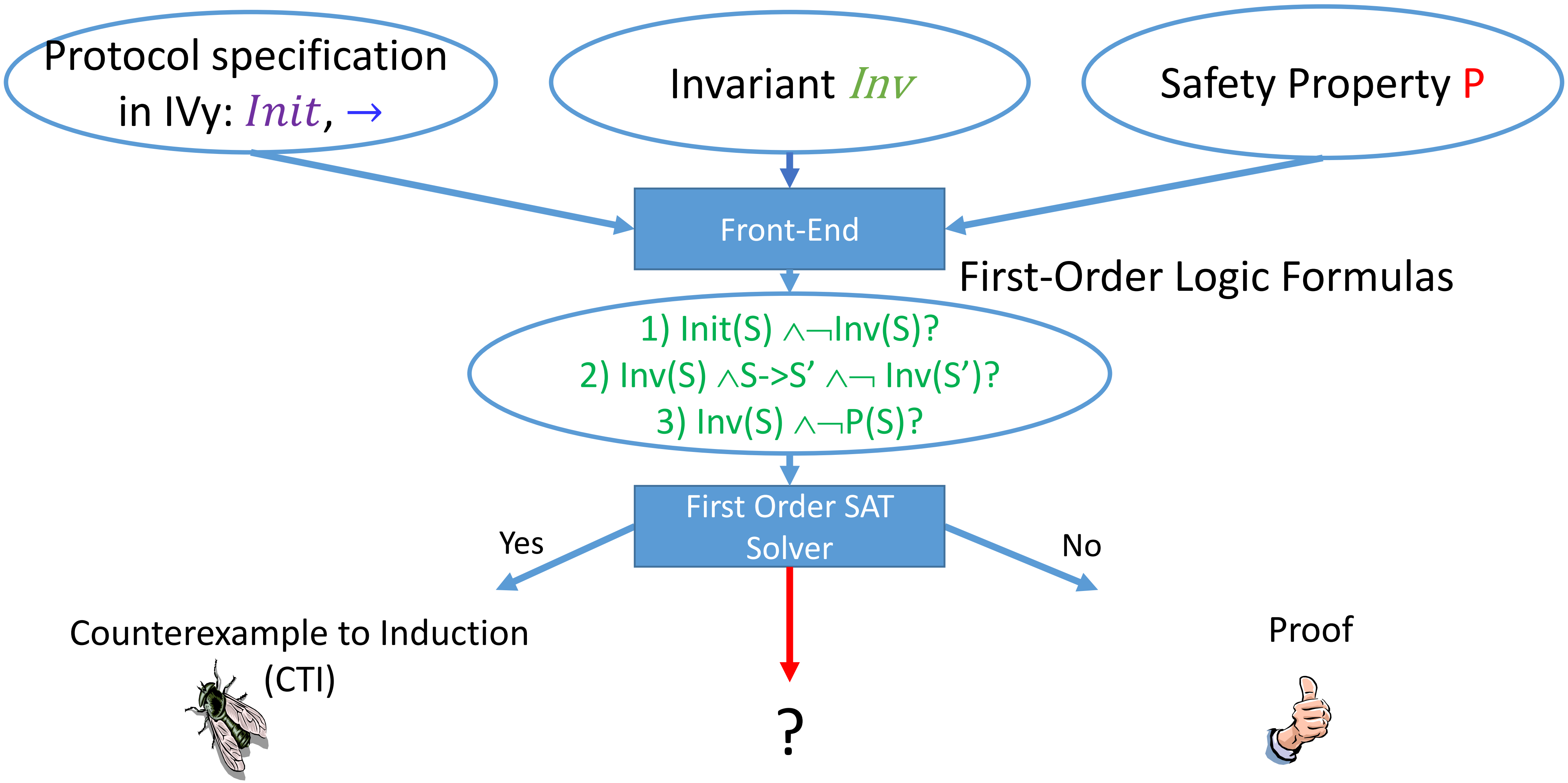
Deductive verification in First-Order Logic



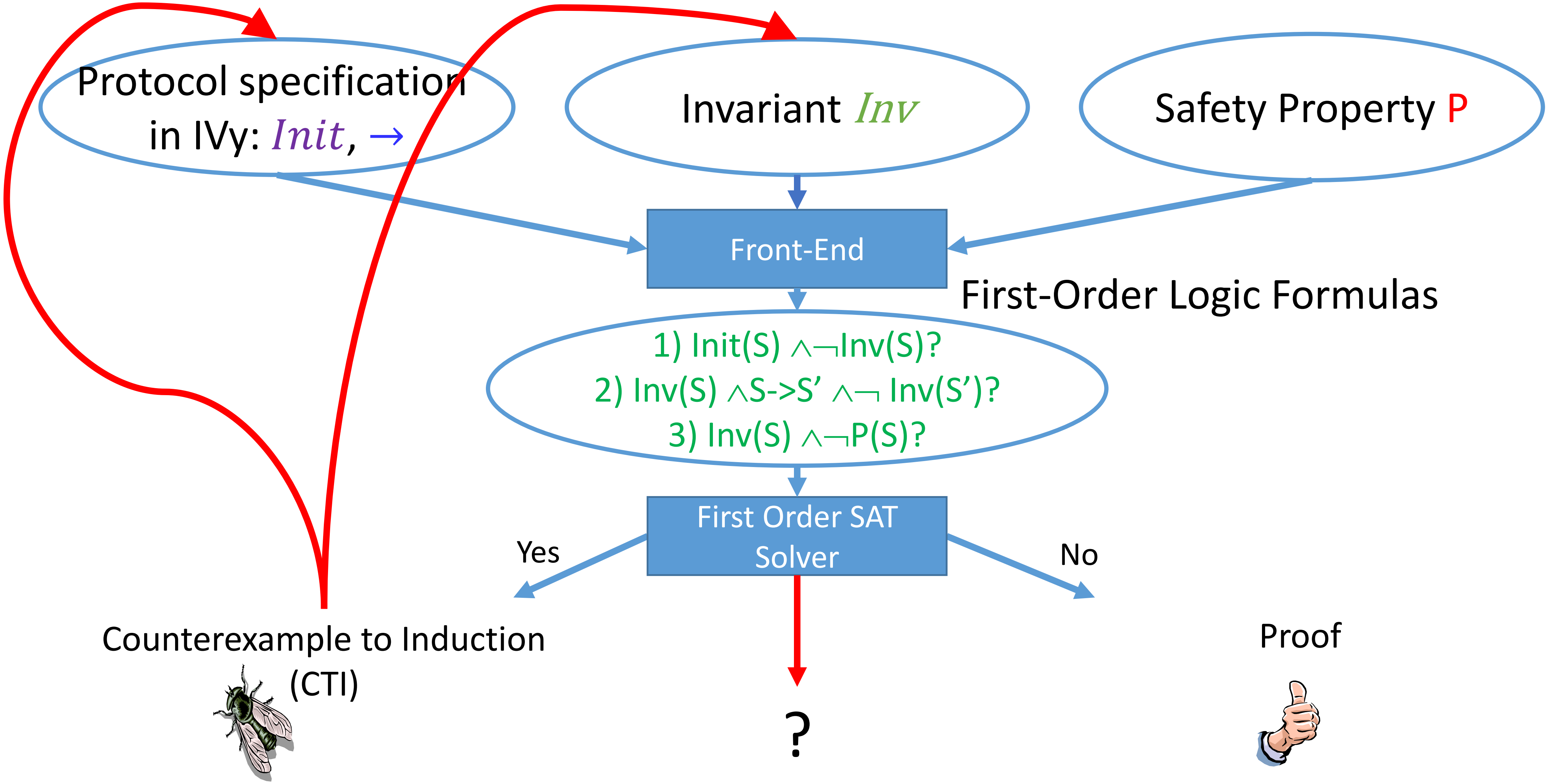
Deductive verification in First-Order Logic



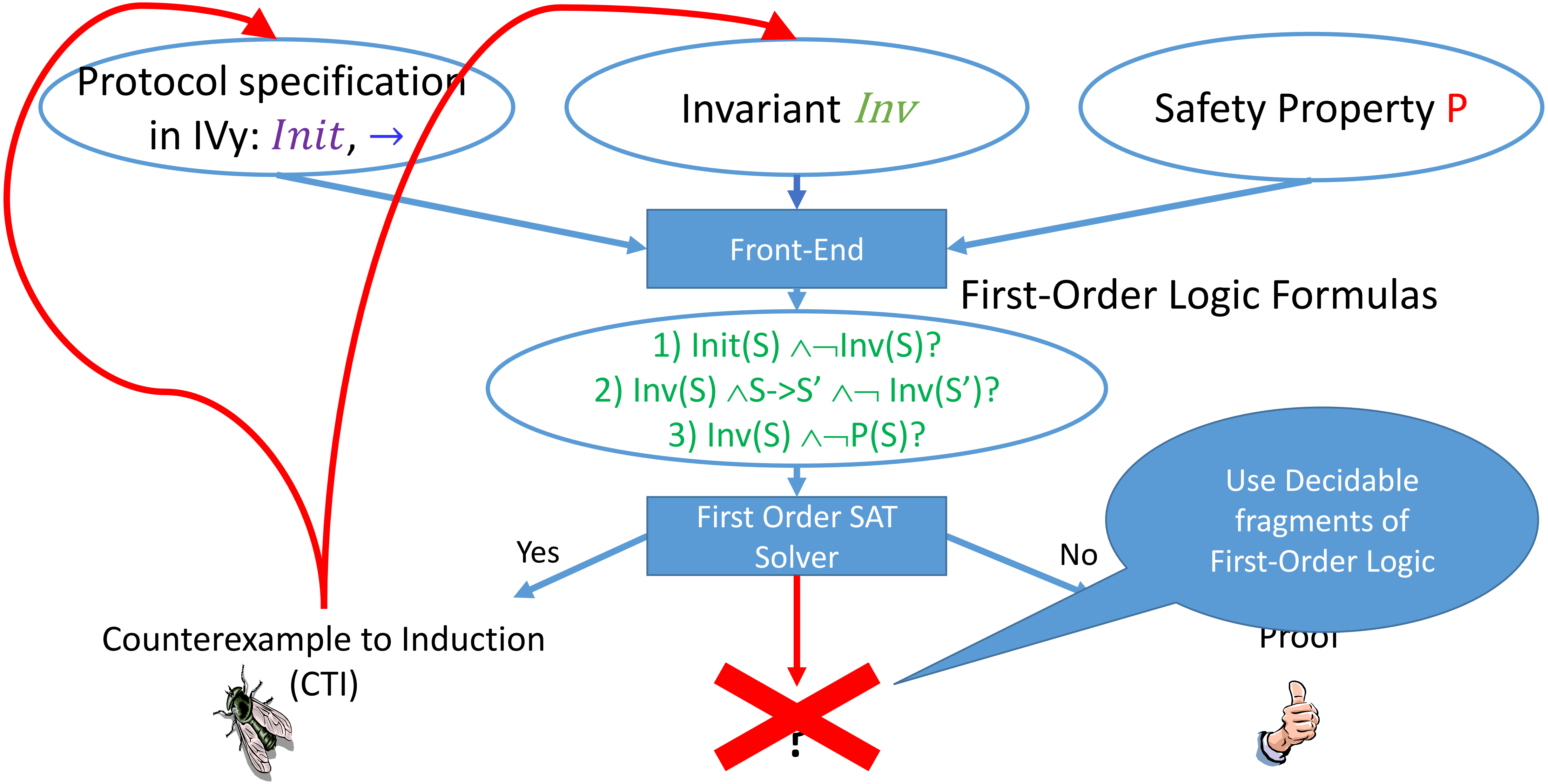
Deductive verification in First-Order Logic



Deductive verification in First-Order Logic



Deductive verification in First-Order Logic



Example: SCP’s inductive invariant

```
invariant forall V1,V2,B1,B2 .
      confirm(V1,B1,commit) & confirm(V2,B2,commit) -> B1.x = B2.x
```

Example: SCP's inductive invariant

```
invariant forall V1,V2,B1,B2 .
      confirm(V1,B1,commit) & confirm(V2,B2,commit) -> B1.x = B2.x

invariant forall V,B . ~ accept(V,B,commit) & accept(V,B,abort)
```

Example: SCP's inductive invariant

```
invariant forall V1,V2,B1,B2 .
      confirm(V1,B1,commit) & confirm(V2,B2,commit) -> B1.x = B2.x

invariant forall V,B . ~ accept(V,B,commit) & accept(V,B,abort)

invariant forall V,B,S . confirm(V,B,S) -> (exists Q . is_quorum(Q) &
      forall V2 . member(V2,Q) -> accept(V2,B,S))
```

Example: SCP's inductive invariant

```
invariant forall V1,V2,B1,B2 .
      confirm(V1,B1,commit) & confirm(V2,B2,commit) -> B1.x = B2.x

invariant forall V,B . ~ accept(V,B,commit) & accept(V,B,abort)

invariant forall V,B,S . confirm(V,B,S) -> (exists Q . is_quorum(Q) &
      forall V2 . member(V2,Q) -> accept(V2,B,S))

invariant forall V, B2 . accept(V,B2,commit) -> (
      (forall B1 . B1 < B2 & B1.x ≠ B2.x ->
            exists Q . is_quorum(Q) & (forall V . member(V,Q) -> accept(N,B1,abort))
      |
      (exists B1 . B1 < B2 & B1.x = B2.x & accept(V,B1,commit)) )
```

Current Status

- High-level specification of the ballot protocol has been proved safe
  https://github.com/nano-o/SCP-Verification
- Next
  - Produce a formal document that is readable along with the Internet Draft
  - Proof of non-blocking property
  - Verified (reference) implementation

More information on IVy and its verification techniques

- https://microsoft.github.io/ivy/
- Padon, Oded, et al. "Paxos made EPR: decidable reasoning about distributed protocols.“ OOPSLA 2017
- Padon, Oded, et al. "Reducing liveness to safety in first-order logic.“ POPL 2018
- Taube, Marcelo, et al. "Modularity for decidability of deductive verification with applications to distributed systems.“ PLDI 2018