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

```plaintext
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

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

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

Using Lightweight Modeling To Understand Chord

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:

\[(A \rightarrow B) \land (B \rightarrow C) \rightarrow (A \rightarrow C)\]
What is a formal proof?

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

\[
(A \rightarrow B) \land (B \rightarrow C) \rightarrow (A \rightarrow C)
\]

Proof of:

\[
\begin{align*}
A & \quad \text{(1)} \\
A \rightarrow B \\
B & \\
B \rightarrow C \\
C & \\
A \rightarrow C & \quad \text{(1)}
\end{align*}
\]
What is a formal proof?

• Like a mathematician’s proof, but much more detailed
• Machine-checked
Proving from first principles is hard

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

Proving from first principles is hard

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

Proving from first principles is hard

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

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."
Inductive Invariants

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

1. Initiation: Inv(s₀) holds in the initial state s₀
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 $\text{Inv}(s)$ such that:

1. Initiation: $\text{Inv}(s_0)$ holds in the initial state $s_0$
2. Consecution: If $\text{Inv}(s)$ holds and $s \rightarrow s'$, then $\text{Inv}(s')$ holds
3. Safety: $\text{Inv}(s)$ 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

Protocol specification in IVy: $Init, \rightarrow$

Invariant $Inv$

Safety Property $P$

Front-End

1) $Init(S) \land \neg Inv(S)$?
2) $Inv(S) \land S \rightarrow S' \land \neg Inv(S')$?
3) $Inv(S) \land \neg P(S)$?

First Order SAT Solver

Yes

No

Counterexample to Induction (CTI)

Proof
Deductive verification in First-Order Logic

Protocol specification in IVy: \( \text{Init}, \rightarrow \)

Invariant \( \text{Inv} \)

Safety Property \( P \)

Front-End

First-Order Logic Formulas

1) \( \text{Init}(S) \land \neg \text{Inv}(S) \)?
2) \( \text{Inv}(S) \land S \rightarrow S' \land \neg \text{Inv}(S') \)?
3) \( \text{Inv}(S) \land \neg P(S) \)?

First Order SAT Solver

Yes

No

Counterexample to Induction (CTI)

Proof

\(?\)
Deductive verification in First-Order Logic

Protocol specification in IVy: \(\text{Init, } \rightarrow\)

Invariant \(\text{Inv}\)

Safety Property \(P\)

Front-End

First-Order Logic Formulas

1) \(\text{Init}(S) \land \neg \text{Inv}(S)\)?
2) \(\text{Inv}(S) \land S \rightarrow S' \land \neg \text{Inv}(S')\)?:
3) \(\text{Inv}(S) \land \neg P(S)\)?:

First Order SAT Solver

Counterexample to Induction (CTI)

Yes

No

Proof

?
Deductive verification in First-Order Logic

Protocol specification in IVy: \textit{Init}, \rightarrow

Invariant \textit{Inv}

Safety Property \textit{P}

\begin{align*}
1) \Init(S) \land \neg \Inv(S)\
2) \Inv(S) \land S \rightarrow S' \land \neg \Inv(S')\
3) \Inv(S) \land \neg \P(S)
\end{align*}

First Order SAT Solver

Counterexample to Induction (CTI)

Proof
Deductive verification in First-Order Logic

Protocol specification in IVy: $\text{Init, } \rightarrow$

Invariant $\text{Inv}$

Safety Property $\text{P}$

First-Order Logic Formulas

1) $\text{Init}(S) \land \neg \text{Inv}(S)$?
2) $\text{Inv}(S) \land S \rightarrow S' \land \neg \text{Inv}(S')$?
3) $\text{Inv}(S) \land \neg \text{P}(S)$?

First Order SAT Solver

Yes

No

Proof

Counterexample to Induction (CTI)
Deductive verification in First-Order Logic

Protocol specification in IVy: $Init, \rightarrow$

Invariant $Inv$

Safety Property $P$

Front-End

First-Order Logic Formulas

1) $Init(S) \land \neg Inv(S)$?
2) $Inv(S) \land S \rightarrow S' \land \neg Inv(S')$?
3) $Inv(S) \land \neg P(S)$?

First Order SAT Solver

Use Decidable fragments of First-Order Logic

Counterexample to Induction (CTI)

Proof
Example: SCP’s inductive invariant

\[
\text{invariant \ \forall V_1, V_2, B_1, B_2 .
\quad \text{confirm}(V_1, B_1, \text{commit}) \land \text{confirm}(V_2, B_2, \text{commit}) \rightarrow B_1.x = B_2.x}
\]
Example: SCP’s inductive invariant

\[
\text{invariant \; forall \; V_1,V_2,B_1,B_2 . \;}
\text{confirm(V_1,B_1,commit) \; \& \; confirm(V_2,B_2,commit) \; \rightarrow \; B_1.x = B_2.x}
\]

\[
\text{invariant \; forall \; V,B . \; \sim \; accept(V,B,commit) \; \& \; accept(V,B,abort)}
\]
Example: SCP’s inductive invariant

\[ \text{invariant } \forall V_1, V_2, B_1, B_2 . \]
\[ \text{confirm}(V_1, B_1, \text{commit}) \land \text{confirm}(V_2, B_2, \text{commit}) \implies B_1.x = B_2.x \]

\[ \text{invariant } \forall V, B . \sim \text{accept}(V, B, \text{commit}) \land \text{accept}(V, B, \text{abort}) \]

\[ \text{invariant } \forall V, B, S . \text{confirm}(V, B, S) \implies (\exists Q . \text{is\_quorum}(Q) \land \forall V_2 . \text{member}(V_2, Q) \implies \text{accept}(V_2, B, S)) \]
Example: SCP’s inductive invariant

\textbf{invariant} \quad \text{forall } V_1, V_2, B_1, B_2. \quad \text{confirm}(V_1, B_1, \text{commit}) \& \text{confirm}(V_2, B_2, \text{commit}) \rightarrow B_1.x = B_2.x

\textbf{invariant} \quad \text{forall } V, B. \sim \text{accept}(V, B, \text{commit}) \& \text{accept}(V, B, \text{abort})

\textbf{invariant} \quad \text{forall } V, B, S. \quad \text{confirm}(V, B, S) \rightarrow (\exists Q. \text{is_quorum}(Q) \& \text{forall } V_2. \text{member}(V_2, Q) \rightarrow \text{accept}(V_2, B, S))

\textbf{invariant} \quad \text{forall } V, B_2. \quad \text{accept}(V, B_2, \text{commit}) \rightarrow \left( \left( \text{forall } B_1. \quad B_1 < B_2 \& B_1.x \neq B_2.x \rightarrow \right. \right.
\left. \exists Q. \text{is_quorum}(Q) \& (\text{forall } V. \text{member}(V, Q) \rightarrow \text{accept}(V, B_1, \text{abort})) \right)
\mid 
\left( \exists B_1. \quad B_1 < B_2 \& B_1.x = B_2.x \& \text{accept}(V, B_1, \text{commit}) \right) \right)
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