Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv
Contributors

Marcelo Taube, Giuliano Losa, Kenneth McMillan, Oded Padon, Sharon Shoham

James R. Wilcox, Doug Woos
And Also

Anindya Benerjee, Neil Immerman, Shachar Itzhaky, Aleks Nanevsky  Aurojit Panda

http://microsoft.github.io/ivy/
Virtual Machine

• http://www.cs.tau.ac.il/~odedp/ivy-sri18.ova
Deductive Verification of Distributed Protocols in First-Order Logic

[CAV’13] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, MS: Effectively-Propositional Reasoning about Reachability in Linked Data Structures

[PLDI’16] Oded Padon, Kenneth McMillan, Aurojit Panda, MS, Sharon Shoham Ivy: Safety Verification by Interactive Generalization

[POPL’16] Oded Padon, Neil Immerman, Aleksandr Karbyshev, Sharon Shoham, MS Decidability of Inferring Inductive Invariants

[OOPSLA’17] Oded Padon, Giuliano Losa, MS, Sharon Shoham Paxos made EPR: Decidable Reasoning about Distributed Protocols

[PLDI’18] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, MS, Sharon Shoham, James R. Wilcox, Doug Woos: Modularity for Decidability of Deductive Verification with Applications to Distributed Systems
Agenda

• Today
  • Motivation
  • Deductive Verification in Ivy

• Wednesday
  • Decidable logics
  • Case study
    • Reasoning about linked list
    • Modularity and decidability
Why verify distributed protocols?

• Distributed systems are everywhere
  • Safety-critical systems
  • Cloud infrastructure
  • Blockchain

• Distributed systems are notoriously hard to get right
  • Even small protocols can be tricky
  • Bugs occur on rare scenarios
  • Testing is costly and not sufficient
Why verify distributed protocols?

• Distributed systems are everywhere
  • Safety-critical systems
  • Cloud infrastructure
  • Blockchain

• Distributed systems are notoriously hard to get right

SIGCOMM’01

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

Jon Stewia, Robert Morris, David Iben Nowell, David R. Karger, M. Frances Kaushoek, Frank Dabek, and Hari Bulakrishnan. 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-
Why verify distributed protocols?

- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchain
- Distributed systems are notoriously hard to get right
Zyzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kolla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong
Dept. of Computer Sciences
University of Texas at Austin

Zyzyva is a state machine replication protocol based on 
three protocols: (1) agreement, (2) view change, and (3) 
protocol orders requests for execution. The view change protocol coordinates
across all replicas to ensure that requests are executed in a consistent order.

We now proceed to demonstrate that the view-change 
mechanism in Zyzyva does not guarantee safety.

Revisiting Fast Practical Byzantine Fault Tolerance

Ittai Abraham, Guy Guea, Dahlia Malkhi
VMware Research
with:
Lorenzo Alvisi (Cornell),
Rama Kotla (Amazon),
Jean-Philippe Martin (Verily)
Proving distributed systems is hard

• Amazon [CACM’15] uses TLA+ for testing protocols, but no proofs
• IronFleet [SOSP’15] – verification of Multi-Paxos in Dafny (3.7 person-years)
• Verdi [PLDI’15] – verification of Raft in Coq (50,000 lines of proofs)

Our goal: reduce human effort while maintaining flexibility

Our approach: decompose verification into decidable problems

Automatic verification of infinite-state systems

Verification
Is there a behavior of \( S \) that violates \( \varphi \)?

Rice’s Theorem
I can’t decide!

Counterexample
Unknown / Diverge
Proof

"Program verification is the holy grail of computer science; always was; always will be”

Chet Murthy
Semi-automatic deductive verification
Deductive verification

System $S$   Invariant $Inv$   Property $\varphi$

Deductive Verification
Is $Inv$ an inductive invariant for $S$ that proves $\varphi$?
Are the logical verification conditions valid?

"Deduction is forever" Amir Pnueli
Inductive invariants

System $S$ is safe if all the reachable states satisfy the property $\neg \text{Bad}$.
System $S$ is safe if all the reachable states satisfy the property $\neg Bad$.

System $S$ is safe iff there exists an inductive invariant $Inv$:

- $Init \subseteq Inv$ (Initiation)
- if $\sigma \in Inv$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in Inv$ (Consecution)
- $Inv \cap Bad = \emptyset$ (Safety)

Translated to VC’s
Counterexample To Induction (CTI)

- States $\sigma, \sigma'$ are a CTI of $\text{Inv}$ if:
  - $\sigma \in \text{Inv}$
  - $\sigma' \notin \text{Inv}$
  - $\sigma \rightarrow \sigma'$

- A CTI may indicate:
  - A bug in the system
  - A bug in the safety property
  - A bug in the inductive invariant
    - Too weak
    - Too strong
Strengthening & weakening from CTI

Strengthening

Weakening

\[ \sigma \in \text{Inv} \]

\[ \sigma' \in \text{Inv} \]

\[ \sigma \in \text{Inv}' \]

\[ \sigma' \in \text{Inv}' \]
Induction on a ball game

• Four players pass a ball:
  • A will pass to C
  • B will pass to D
  • C will pass to A
  • D will pass to B
• The ball starts at player A
• Can the ball get to D?
Induction on a ball game

• Four players pass a ball:
  • A will pass to C
  • B will pass to D
  • C will pass to A
  • D will pass to B
• The ball starts at player A
• Can the ball get to D?
Formalizing with induction

• $x_0 = A$

• $x_{n+1} = \begin{cases} 
C & \text{if } x_n = A \\
D & \text{if } x_n = B \\
A & \text{if } x_n = C \\
B & \text{if } x_n = D 
\end{cases}$

• Prove by induction $\forall n. x_n \neq D$
  
  • $x_0 \neq D$ ?
  
  • $x_m \neq D \Rightarrow x_{m+1} \neq D$ ?
Formalizing with induction

• $x_0 = A$

• $x_{n+1} = \begin{cases} 
C & \text{if } x_n = A \\ 
D & \text{if } x_n = B \\ 
A & \text{if } x_n = C \\ 
B & \text{if } x_n = D 
\end{cases}$

• Prove a stronger claim by induction $\forall n. x_n \neq B \land x_n \neq D$
  • $x_0 \neq B \land x_0 \neq D$
  • $x_m \neq B \land x_m \neq D \Rightarrow x_{m+1} \neq B \land x_{m+1} \neq D$
Simple example: loop invariants

\begin{align*}
  x &:= 1; \\
y &:= 2; \\
\text{while } * \text{ do } \{ \\
  \quad \text{assert } \neg \text{even}[x]; \\
  \quad x := x + y; \\
  \quad y := y + 2; \\
\} \\
\end{align*}
Simple example: loop invariants

\[ x := 1; \]
\[ y := 2; \]
\[ \text{while } * \text{ do } \{ \]
\[ \quad \text{assert } \neg \text{even}[x]; \]
\[ \quad x := x + y; \]
\[ \quad y := y + 2; \]
\[ \} \]

Counterexample to induction (CTI)

\[ \neg \text{even}[x] \]

\[ \text{even}[x] \]

\[ x = 1, y = 1 \]
\[ x = 1, y = 3 \]
\[ x = 2, y = 4 \]
\[ x = 2, y = 3 \]
\[ x = 2, y = 5 \]
\[ x = 4, y = 5 \]

\[ x = 1, y = 0 \]
\[ x = 3, y = 0 \]
\[ x = 3, y = 2 \]
\[ x = 7, y = 6 \]

\[ x = 1, y = 2 \]
\[ x = 5, y = 4 \]
Simple example: loop invariants

\begin{verbatim}
x := 1;
y := 2;
while * do {
  assert \neg even[x];
  x := x + y;
  y := y + 2;
}
\end{verbatim}

\[ \text{Inv} = \neg \text{even}[x] \land \text{even}[y] \]
Simple example: loop invariants

\[ \text{Inv} = \neg \text{even}[x] \land \text{even}[y] \]

\[
\begin{align*}
x &:= 1; \\
y &:= 2; \\
\text{while } * \text{ do } \{ & \\
&\quad \text{assert } \neg \text{even}[x]; \\
&\quad x := (x*x - y*y)/(x - y); \\
&\quad y := y + 2; \\
&\}\]
\]
Simple example: loop invariants

\[ \text{Inv} = y^2 - 2y - 4x + 4 = 0 \]

\[
\begin{align*}
x & := 1; \\
y & := 2; \\
\text{while } * \text{ do } & \{ \\
& \quad \text{assert } \neg \text{even}[x]; \\
& \quad x := x + y; \\
& \quad y := y + 2; \\
& \}
\end{align*}
\]
Dafny [Leino’17]

Is \( \text{Inv} \) an inductive invariant for \( S \) that proves \( \varphi \)?

→ Are the logical verification conditions valid?

Deductive verification

Church’s Theorem

I can’t decide!

Deductive Verification
Is $Inv$ an inductive invariant for $S$ that proves $\varphi$?
→ Are the logical verification conditions valid?

Counter-model
Unknown / Diverge
Proof
Effects of undecidability

- The verifier may fail on tiny programs
- No explanation when tactics fails
- Counterproofs
- The butterfly effect
- Observed in the IronFleet Project
Challenges in deductive verification

1. **Formal specification**: formalizing infinite-state systems and their properties

2. **Deduction**: checking inductiveness
   - Undecidability of implication checking
     - Unbounded state (threads, messages), arithmetic, quantifier alternation

3. **Inference**: finding inductive invariants \((\text{Inv})\)
   - Hard to specify
   - Hard to maintain
   - Hard to infer
     - Undecidable even when deduction is decidable
State of the art in formal verification

Proof Assistants

Expressiveness

Ultimately limited by human

Verdi: \(~10\)
IronFleet: \(~4\)

Decidable deduction
Finite counterexamples
proof/code: \(~0.2\)

Decidable Models
Model Checking
Static Analysis

Automated

Ultimately limited by undecidability
Modularity

Original system

Original inductive argument

Original property
Verification of each module

- subsystem
- Partial argument
- Property

Verification tool

- Incorrect
  - Finds bug
- Correct
  - Finds proof

NO UNDECIDABILITY
Ivy’s principles

• Modularity
  • The user breaks the verification system into small problems expressed in decidable logics
  • The system explores circular assume/guarantee reasoning to prove correctness

• Inductive invariants and transition systems are expressed in decidable logics
  • Turing complete imperative programs over unbounded relations
  • Allows quantifiers to reason about unbounded sets
    • But no arbitrary quantifier alternations and theories
  • Checking inductiveness is decidable

• Display CTIs as graphs (similar to Alloy)
## Languages and Verification

<table>
<thead>
<tr>
<th>Language</th>
<th>Executable</th>
<th>Expressiveness</th>
<th>Inductiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, Java, Python...</td>
<td>✓</td>
<td>Turing-Complete</td>
<td>Undecidable</td>
</tr>
<tr>
<td>SMV</td>
<td>✗</td>
<td>Finite-state</td>
<td>Temporal Properties</td>
</tr>
<tr>
<td>TLA+</td>
<td>✗</td>
<td>Turing-Complete</td>
<td>Manual</td>
</tr>
<tr>
<td>Coq, Isabelle/HOL</td>
<td>✓</td>
<td>Turing-Complete</td>
<td>Manual with tactics</td>
</tr>
<tr>
<td>Dafny</td>
<td>✓</td>
<td>Turing-Complete</td>
<td>Undecidable with lemmas</td>
</tr>
<tr>
<td>Ivy</td>
<td>✓</td>
<td>Turing-Complete</td>
<td>Decidable(EPR)</td>
</tr>
</tbody>
</table>
Example: Leader election in a ring

- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node’s own id
  - A node that receives its own id becomes a leader
- Theorem: The protocol selects at most one leader
  - Inductive? NO

Example: Leader election in a ring

• Unidirectional ring of nodes, unique numeric ids

• Protocol:
  • Each node sends its id to the next
  • Upon receiving a message, a node passes it (to the next) if
    the id in the message is higher than the node’s own id
  • A node that receives its own id becomes a leader

• Theorem: The protocol selects at most one leader

Proposition: This algorithm detects one and only one highest number.

Argument: By the circular nature of the configuration and the consistent direction of messages, any message must meet all other processes before it comes back to its initiator. Only one message, that with the highest number, will not encounter a higher number on its way around. Thus, the only process getting its own message back is the one with the highest number.

Leader election protocol – first-order logic

- $\leq (\text{ID, ID})$ – total order on node id’s
- $\text{btw} (\text{Node, Node, Node})$ – the ring topology
- $\text{id}: \text{Node} \rightarrow \text{ID}$ – relate a node to its unique id
- $\text{pending}(\text{ID, Node})$ – pending messages
- $\text{leader}(\text{Node})$ – leader(n) means n is the leader

Axiomatized in first-order logic

protocol state

first-order structure

$\langle n_5, n_1, n_3 \rangle \in I(\text{btw})$
Leader election protocol – first-order logic

- \( \leq (\text{ID}, \text{ID}) \) – total order on node id’s
- btw (Node, Node, Node) – the ring topology
- id: Node \( \rightarrow \) ID – relate a node to its unique id
- pending(ID, Node) – pending messages
- leader(Node) – leader(n) means n is the leader

Specify and verify the protocol for any number of nodes in the ring

Axiomatized in first-order logic
Leader election protocol – first-order logic

- \(\leq\) (ID, ID) – total order on node id’s
- btw (Node, Node, Node) – the ring topology
- id: Node \(\rightarrow\) ID – relate a node to its unique id
- pending(ID, Node) – pending messages
- leader(Node) – leader(n) means n is the leader

```plaintext
action send(n: Node) = {
    "s := next(n)";
    pending(id(n), s) := true
}
```

Action receive(n: Node, m: ID) = {
    requires pending(m, n);
    if id(n) = m then
        // found leader
        leader(n) := true
        pending(m, s) := true
    else if id(n) \(\leq\) m then
        // pass message
        "s := next(n)";
        pending(m, s) := true
}

\(\mathcal{T}_R\) (send):
\[\exists n, s: \text{Node. } "s = \text{next}(n)" \land \forall x: \text{ID}, y: \text{Node. } \text{pending}'(x,y) \leftrightarrow (\text{pending}(x,y) \lor (x = \text{id}(n) \land y = s))\]

Bad:
assert I\(\emptyset\) = \forall x, y: \text{Node. } \text{leader}(x) \land \text{leader}(y) \rightarrow x = y
Leader election protocol – inductive invariant

**Safety property:** $I_0$

$I_0 = \forall x, y: \text{Node}. \ \text{leader}(x) \land \text{leader}(y) \rightarrow x = y$

**Inductive invariant:** $\text{Inv} = I_0 \land I_1 \land I_2 \land I_3$

$I_1 = \forall n_1: \text{Node}. \ \text{leader}(n_1) \rightarrow \text{id}[n_1] = \text{id}[n_1]$

$I_2 = \forall n_1, n_2: \text{Node}. \ \text{pending}(\text{id}[n_2], n_2) \rightarrow \text{id}[n_1] < \text{id}[n_2]$

$I_3 = \forall n_1, n_2, n_3: \text{Node}. \ \text{btw}(n_1, n_2, n_3) \land \text{pending}(\text{id}[n_2], n_1) \rightarrow \text{id}[n_3] < \text{id}[n_2]$

---

**EPR Solver**

I can decide EPR!

Proof

---

**How can we find an inductive invariant without knowing it?**

- $\leq (\text{ID}, \text{ID})$ – total order on node id’s
- $\text{btw}(\text{Node}, \text{Node}, \text{Node})$ – the ring topology
- $\text{id}: \text{Node} \rightarrow \text{ID}$ – relate a node to its unique id
- $\text{pending}(\text{ID}, \text{Node})$ – pending messages
- $\text{leader}(\text{Node})$ – leader(n) means n is the leader
Interactive invariant inference [PLDI’16]

Model

Candidate Inductive Invariant

Inductive Invariant Found

Inductive?

Yes

No

Find “minimal” CTI

Generalize from CTI

User

Automation

Modify candidate invariant

I can decide EPR!

EPR
Leader Protocol

Inv = I_0 \land I_1 \land I_2

Check Inductiveness

Ivy: check inductiveness

CTI

rvc(1, id(2))

EPR

¬I_2
Ivy: check inductiveness

Leader Protocol

$Inv = I_0 \land I_1 \land I_2 \land I_3$

$Bad = \neg I_0$

VC Generator

$Init(V) \land \neg Inv(V)$
$Inv(V) \land TR(V, V') \land \neg Inv(V')$
$Inv(V) \land Bad(V)$

EPR Solver

Proof

$I_0 \land I_1 \land I_2 \land I_3$ is an inductive invariant for the leader protocol, proving its safety
∀* invariant – excluded substructures

\[ \text{Init} \subseteq \text{Inv} \ (\text{Initiation}) \]

if \( \sigma \in \text{Inv} \) and \( \sigma \rightarrow \sigma' \) then \( \sigma' \in \text{Inv} \) (Consecution)

\[ \text{Inv} \cap \text{Bad} = \emptyset \ (\text{Safety}) \]
## Principle: first-order abstractions/modularity

<table>
<thead>
<tr>
<th>Concept</th>
<th>Intention</th>
<th>First-order abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node ID’s</strong></td>
<td><strong>Integers</strong></td>
<td><strong>function id: Node → ID</strong>&lt;br&gt;relation ≤(ID, ID)&lt;br&gt;axiom ∀x:ID. x ≤ x  reflexive&lt;br&gt;axiom ∀x,y,z:ID. x≤y ∧ y≤z → x ≤ z  transitive&lt;br&gt;axiom ∀x,y:ID. x≤y ∧ y≤x → x=y  anti-symmetric&lt;br&gt;axiom ∀x,y:ID. x≤y ∨ y ≤ x  total&lt;br&gt;axiom ∀x, y: Node. id(x) = id(y) → x=y  injective</td>
</tr>
<tr>
<td><strong>Ring Topology</strong></td>
<td><strong>Next edges + Transitive closure</strong>&lt;br&gt;relation btw (Node, Node, Node)&lt;br&gt;axiom ∀x, y, z: Node. btw(x, y, z) → btw(y, z, x)  circular&lt;br&gt;axiom ∀x, y, z, w: Node. btw(w, x, y) ∧ btw(w, y, z) → btw(w, x, z)  transitive&lt;br&gt;axiom ∀x, y, w: Node. btw(w, x, y) → ¬btw(w, y, x)  anti-symmetric&lt;br&gt;axiom ∀x, y, w: Node. ≠(w, x, y) → btw(w, x, y) ∨ btw(w, y, x)  total&lt;br&gt;macro “next(a)=b” ≡ ∀x: Node. x=a ∨ x=b ∨ btw(a,b,x)  edges</td>
<td></td>
</tr>
</tbody>
</table>
Challenge: How to use restricted first-order logic to verify interesting systems?

- Expressing transitive closure
  - Linked lists
  - Ring protocols

- Expressing sets and cardinalities
  - Paxos, Multi-Paxos
  - Reconfiguration
  - Byzantine Fault Tolerance

- Liveness and temporal properties
Key idea: representing deterministic paths
[Itzhaky SIGPLAN Dissertation Award 2016]

Alternative 1: maintain $n$
- $n^*$ defined by transitive closure of $n$
- not definable in first-order logic

Alternative 2: maintain $n^*$
- $n$ defined by transitive reduction of $n^*$
- Unique due to outdegree $\leq 1$
- Definable in first order logic (for roots)
  - $n^+(a,b) \iff n^*(a,b) \land a \neq b$
  - $n(a,b) \iff n^+(a,b) \land \forall z: n^+(a,z) \rightarrow n^*(b,z)$

Not first order expressible
First order expressible
Challenge: How to use restricted first-order logic to verify interesting systems?

• Expressing transitive closure
  • Linked lists
  • Ring protocols

• Expressing sets and cardinalities
  • Paxos and its variants
  • Byzantine Fault Tolerance
  • Reconfiguration

• Liveness and temporal properties
Paxos

• Single decree Paxos – consensus
  lets nodes make a common decision despite node crashes and packet loss

• Paxos family of protocols – state machine replication
  variants for different tradeoffs, e.g., Fast Paxos is optimized for low
  contention, Vertical Paxos is reconfigurable, etc.

• Pervasive approach to fault-tolerant distributed computing
  • Google Chubby
  • VMware NSX
  • Amazon AWS
  • Many more...
Challenge: sets and cardinalities in FOL

- Consensus algorithms use set cardinalities
  - Wait for messages from more than $N/2$ nodes
- Insight: set cardinalities are used to get a simple effect
  Can be modeled in first-order logic!
- Solution: axiomatize quorums in first-order logic
  sort `Quorum`
  relation `member` (Node, Quorum)
  — set membership (2\textsuperscript{nd}-order logic in first-order)
  axiom $\forall q_1, q_2 : Quorum. \exists n : Node. member(n, q_1) \land member(n, q_2)$

```action
propose(r:Round) {
  require "$N/2 \text{join\_msg's}"
  ...
}
```

```action
propose(r:Round) {
  require $\exists q. \forall n: member(n, q) \rightarrow \exists r', v'. join\_msg(n, r, r', v')$
  ...
}
```
<table>
<thead>
<tr>
<th>Concept</th>
<th>Intention</th>
<th>First-order abstraction</th>
</tr>
</thead>
</table>
| Quorums      | Majority sets                          | **relation** member(Node, Quorum)  
**axiom** $\forall q_1, q_2 : \text{Quorum} \exists n : \text{Node}. \text{member}(n, q_1) \land \text{member}(n, q_2)$ |
| Rounds       | Natural numbers                        | **relation** $\leq$(Round, Round)  
**axiom** $\forall x : \text{Round}. x \leq x$  reflexive  
**axiom** $\forall x, y, z : \text{Round}. x \leq y \land y \leq z \rightarrow x \leq z$  transitive  
**axiom** $\forall x, y : \text{Round}. x \leq y \land y \leq x \rightarrow x = y$  anti-symmetric  
**axiom** $\forall x, y : \text{Round}. x \leq y \lor y \leq x$  total |
| Messages     | Network with: dropping duplication reordering | **relation** start_msg(Round)  
**relation** join_msg(Node, Round, Round, Value)  
**relation** propose_msg(Round, Value)  
**relation** vote_msg(Node, Round, Value) |
Paxos in first-order logic

\[
\forall n_1,n_2 : \text{node}, r_1,r_2 : \text{round}, v_1,v_2 : \text{value. decision}(n_1,r_1,v_1) \land \text{decision}(n_2,r_2,v_2) \rightarrow v_1 = v_2
\]

\[
\forall r : \text{round}, v_1,v_2 : \text{value. propose_msg}(r,v_1) \land \text{propose_msg}(r,v_2) \rightarrow v_1 = v_2
\]

\[
\forall n : \text{node}, r : \text{round}, v : \text{value. vote_msg}(n,r,v) \rightarrow \text{propose_msg}(r,v)
\]

\[
\forall r : \text{round}, v : \text{value.} (\exists n : \text{node. decision}(n,r,v)) \rightarrow \exists q : \text{quorum.} \forall n : \text{node. member}(n,q) \rightarrow \text{vote_msg}(n,r,v)
\]

\[
\forall n : \text{node}, r',r'' : \text{round}, v,v' : \text{value. join_ack_msg}(n,r',v) \land r' \neq r \rightarrow \neg \text{vote_msg}(n,r'',v')
\]

\[
\forall n : \text{node}, r',r'' : \text{round}, v : \text{value. join_ack_msg}(n,r',v) \land r' \neq \bot \land r' < r \land \text{vote_msg}(n,r'',v)
\]

\[
\forall n : \text{node}, v : \text{value.} \neg \text{vote_msg}(n,\bot,v)
\]

\[
\forall r_1,r_2 : \text{round}, v_1,v_2 : \text{value}, q : \text{quorum. propose_msg}(r_2,v_2) \land r_1 < r_2 \land v_1 \neq v_2 \rightarrow
\]

\[
\exists n : \text{node}, r',r'' : \text{round}, v : \text{value. member}(n,q) \land \neg \text{vote_msg}(n,r_1,v_1) \land r' > r_1 \land \text{join_ack_msg}(n,r',r'',v)
\]
Quantifier alternation cycles

- Axiom
  \( \forall q_1, q_2: \text{Quorum}. \exists n: \text{Node}. \text{member}(n, q_1) \land \text{member}(n, q_2) \)

- Propose action precondition
  \( \exists q: \text{Quorum}. \forall n: \text{Node}. \text{member}(n, q) \rightarrow \exists r': \text{Round}, v': \text{Value}. \text{join\_msg}(n, r, r', v') \)

- Inductive invariant
  \( \forall r: \text{Round}, v: \text{Value}. \text{decision}(r, v) \rightarrow \exists q: \text{Quorum}. \forall n: \text{Node}. \text{member}(n, q) \rightarrow \text{vote\_msg}(n, r, v) \)
Paxos made EPR [OOPSLA’17]

Methodology for decidable verification of infinite-state systems

Modeling

Protocol

1

Formal specification in first-order logic

Transforming

Formal specification with decidable VC

First-order abstractions
Domain knowledge

I can decide EPR!

Z3
Inductive invariant of Paxos

# safety property
conjecture decision(N1,R1,V1) & decision(N2,R2,V2) -> V1 = V2

# proposals are unique per round
conjecture proposal(R,V1) & proposal(R,V2) -> V1 = V2

# only vote for proposed values
conjecture vote(N,R,V) -> proposal(R,V)

# decisions come from quorums of votes:
conjecture forall R, V. (exists N. decision(N,R,V)) -> exists Q. forall N. member(N, Q) -> vote(N,R,V)

# properties of one_b_max_vote
conjecture one_b_max_vote(N,R2,none,V1) & ~le(R2,R1) -> ~vote(N,R1,V2)
conjecture one_b_max_vote(N,R,RM,V) & RM ~= none -> ~le(R,RM) & vote(N,RM,V)
conjecture one_b_max_vote(N,R,RM,V) & RM ~= none & ~le(R,RO) & ~le(RO,RM) -> ~vote(N,RO,VO)

# property of choosable and proposal
conjecture ~le(R2,R1) & proposal(R2,V2) & V1 ~= V2 -> exists N. member(N,Q) & left_rnd(N,R1) & ~vote(N,R1,V1)

# property of one_b, left_rnd
conjecture one_b(N,R2) & ~le(R2,R1) -> left_rnd(N,R1)
## Paxos made EPR: experimental evaluation

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model [LOC]</th>
<th>Invariant [Conjectures]</th>
<th>EPR [sec] $\mu$</th>
<th>EPR [sec] $\sigma$</th>
<th>RW [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
<td>1.2</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
<td>1.4</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
<td>-</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
<td>1.5</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0</td>
<td>1.2</td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
<td>1.6</td>
</tr>
</tbody>
</table>

*first mechanized verification

Transformation to EPR reusable across all variants!
Paxos made EPR: experimental evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>( \mu )</td>
<td>( \sigma )</td>
</tr>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

*first mechanized verification

Transformation to EPR reusable across all variants!

Proof / code ratio:
IronFleet: \(~4\)
Verdi: \(~10\)
Ivy: \(~0.2\)
Paxos made EPR: experimental evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

$\mu$ – mean
$\sigma$ – std. deviation

*first mechanized verification
Transformation to EPR reusable across all variants!
## Paxos made EPR: experimental evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
<td>1.2</td>
<td>2</td>
<td>1.2</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
<td>1.4</td>
<td>4</td>
<td>1.8</td>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
<td>-</td>
<td>8</td>
<td>107</td>
<td>129</td>
<td>30%</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
<td>1.5</td>
<td>16</td>
<td>229</td>
<td>110</td>
<td>70%</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0</td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*first mechanized verification

Transformation to EPR reusable across all variants!
# Paxos made EPR: experimental evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>1.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Fast Paxos*</td>
<td>117</td>
<td>17</td>
<td>4.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>3.8</td>
<td>0.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rounds</th>
<th>FOL [sec]</th>
<th>T.O.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>186</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>300</td>
<td>0</td>
</tr>
</tbody>
</table>

Stopable Paxos in FOL

*first mechanized verification

Transformation to EPR reusable across all variants!
have been chosen as the $j$th command for some $j < i$. Although the basic idea of the algorithm is not complicated, getting the details right was not easy.
(1.7) NoneChoosableAfter(\(i, b, v\))
PROOF: We assume \(v \in \text{StopCmd}, j > i, c < b, \) and \(w\) any command and we prove NotChoosable(\(j, c, w\)). By Lemma 1.7, it suffices to prove NotChoosable(\(j, c, w\)). We split the proof into two cases.

(2.1) CASE: sval2a(\(i, b, Q\)) = \(T\)

PROOF: Assumption (1.1.3) implies \(E4(i, b, Q, v)\), so the assumption \(v \in \text{StopCmd}\) implies \(E\!4_i(i, b, Q, v)\). The case assumption, the assumption \(j > i, \) and \(E\!4_i(i, b, Q, v)\) imply sval2a(\(j, b, Q\)) = \(T\). The assumption \(c < b\) and step (1.4) then imply NotChoosable(\(j, c, w\)).

(2.2) CASE: sval2a(\(i, b, Q\)) \(\neq T\)

(3.1) sval2a(\(i, b, Q\)) = val2a(\(i, b, Q\)) = \(v\)

PROOF: Assumption (1.1.3) implies \(E3(i, b, Q, v)\), which implies sval2a(\(i, b, Q\)) = \(v\). The case assumption and the definition of sval2a then implies val2a(\(i, b, Q\)) = \(v\).

(3.2) Done2a(\(i, \text{mbal}2a(i, b, Q), v)\)

PROOF: (3.1), assumption (1.1.4), and the definition of val2a imply vote_{\(i[a][\text{mbal}2a(i, b, Q)]\)} = \(v\) for some acceptor \(a\) in \(Q\), which by Lemma 1.3 implies Done2a(\(i, \text{mbal}2a(i, b, Q), v)\).

By the assumption \(c < b\), it suffices to consider the following two cases.

(3.3) CASE: \(c < \text{mbal}2a(i, b, Q)\)

PROOF: Step (3.2) and assumption (1.1.1) imply NoneChoosableAfter(\(i, \text{mbal}2a(i, b, Q), v)\). By the case assumption and the assumptions \(v \in \text{StopCmd}\) and \(j > i\), this implies NotChoosable(\(j, c, w\)).

(3.4) CASE: \(\text{mbal}2a(i, b, Q) \leq c < b\)

(4.1) \(\text{mbal}2a(j, b, Q) < \text{mbal}2a(i, b, Q)\)

PROOF: The assumption \(v \in \text{StopCmd}\) and (3.1) imply sval2a(\(i, b, Q\)) \(\in \text{StopCmd}\). Case assumption (2.2) and the definition of sval2a then imply \(\text{mbal}2a(k, b, Q) < \text{mbal}2a(i, b, Q)\) for all \(k > i\).

(4.2) NotChoosable(\(j, c, w\))

PROOF: (4.1) and case assumption (3.4) imply \(\text{mbal}2a(j, b, Q) < c < b\). By assumption (1.1.4), Lemma 3 implies NotChoosable(\(j, c, w\)).
Challenge: How to use restricted first-order logic to verify interesting systems?

• Expressing transitive closure
  • Linked lists
  • Ring protocols

• Expressing sets and cardinalities
  • Paxos and its variants
  • Byzantine Fault Tolerance
  • Reconfiguration

• Liveness and temporal properties [POPL’18]

[POPL’18] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, MS, Sharon Shoham
Reducing Liveness to Safety in First-Order Logic
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader in Ring</td>
<td>59</td>
<td>4</td>
<td>1.5</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>50</td>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>DB Chain Replication</td>
<td>143</td>
<td>9</td>
<td>1.7</td>
</tr>
<tr>
<td>Chord</td>
<td>155</td>
<td>12</td>
<td>2.4</td>
</tr>
<tr>
<td>Lock Server (500 Coq lines [Verdi])</td>
<td>122</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Distributed Lock (1 week [IronFleet])</td>
<td>41</td>
<td>7</td>
<td>1.4</td>
</tr>
<tr>
<td>Single Decree Paxos (+liveness)</td>
<td>85</td>
<td>11</td>
<td>10.7</td>
</tr>
<tr>
<td>Multi-Paxos (+liveness)</td>
<td>98</td>
<td>12</td>
<td>14.6</td>
</tr>
<tr>
<td>Vertical Paxos*</td>
<td>123</td>
<td>18</td>
<td>2.2</td>
</tr>
<tr>
<td>Fast Paxos</td>
<td>117</td>
<td>17</td>
<td>6.2</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>11</td>
<td>2.2</td>
</tr>
<tr>
<td>Stoppable Paxos (+liveness) *</td>
<td>132</td>
<td>16</td>
<td>18.4</td>
</tr>
<tr>
<td>Ticket Protocol (+liveness)</td>
<td>86</td>
<td>37</td>
<td>6</td>
</tr>
<tr>
<td>Alternating Bit Protocol (+liveness)</td>
<td>161</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>TLB Shootdown (+liveness) *</td>
<td>385</td>
<td>91</td>
<td>380 (FOL)</td>
</tr>
<tr>
<td>Practical Byzantine Fault Tolerance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconfiguration</td>
<td></td>
<td></td>
<td>Work in progress</td>
</tr>
</tbody>
</table>

**Proof / code ratio:**

IronFleet: ~4
Verdi: ~10
Ivy: ~0.2

* First mechanized liveness proof
Summary

• Distributed protocols are interesting for verification
  • But real distributed systems are more complex
• Decidable logics can be used to reason about interesting systems
  • No more butterfly effects
  • But some jagged corners
  • Details on Wednesday