Interactive Verification of Distributed Protocols

Sharon Shoham

Tel Aviv University

Supervised Verification of Infinite-State Systems
Why Verify Distributed Protocols?

• Distributed systems are everywhere
  • e.g., safety-critical systems

• Distributed systems are notoriously hard to get right

• Testing is costly and not sufficient
  • Bugs occur on rare scenarios
  • Testing covers tiny fraction of behaviors
  • Leaves most bugs for production
  • Amazon employs TLA+ for testing protocols, but scaling is an issue
Verifying Distributed Protocols is Hard

- Infinite state-space
  - unbounded number of objects
  - unbounded number of threads
  - unbounded number of messages

- Asymptotic complexity of program verification
  - The halting problem
  - Rice theorem
  - The ability of simple programs to represent complex behaviors
State of the art in formal verification

• Automatic techniques
  • Model checking
    • Exploit finite state / finite abstraction
  • Abstract Interpretation
    • Sound abstraction
  • Limited for infinite state systems due to undecidability

• Deductive techniques
  • Use SMT for deduction with manual program annotations (e.g. Dafny)
    • Requires programmer effort to provide inductive invariants
    • SMT solver may diverge (matching loops, arithmetic)
  • Interactive theorem provers (e.g. Coq, Isabelle/HOL, LEAN)
    • Programmer gives inductive invariant and proves it
    • Huge programmer effort (~10-50 lines of proof per line of code)
“the proofs consisted of about 5000 lines and assumed several nontrivial invariants of the Raft protocol. This paper discusses the verification of Raft as a whole, including all the invariants from the original Raft paper [32]. These new proofs consist of about 45000 additional lines” [Verdi, CPP’16]
"but our input language cannot compete in generality with mechanized proof methods that rely heavily on human expertise, e.g., IVY [55], Verdi [68], IronFleet [38], TLAPS [16]” [Konnov et al, POPL’17]
State of the art in formal verification

Proof Assistants

- Ultimately limited by human
- proof/code: Verdi: ~10
- IronFleet: ~4

Supervised Verification

- proof/code: IVy ~1/10
- Ultimately limited by undecidability

Model Checking

Static Analysis

Supervised Verification of Infinite-State Systems
# IVy: Verified Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model (# LOC)</th>
<th>Property (# Literals)</th>
<th>Invariant (# Literals)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader in Ring</td>
<td>59</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>50</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>DB Chain Replication</td>
<td>143</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>Chord</td>
<td>155</td>
<td>35</td>
<td>46</td>
</tr>
<tr>
<td>Lock Server (500 Coq lines [Verdi])</td>
<td>122</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Distributed Lock (1 week [IronFleet])</td>
<td>41</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>Single Decree Paxos</td>
<td>85</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>Multi Paxos</td>
<td>102</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>Vertical Paxos</td>
<td>123</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>Fast Paxos</td>
<td>117</td>
<td>3</td>
<td>59</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>Stoppable Paxos</td>
<td>130</td>
<td>6</td>
<td>60</td>
</tr>
<tr>
<td>Virtually Synchronous Paxos</td>
<td></td>
<td></td>
<td>Work in progress</td>
</tr>
</tbody>
</table>
Automatic Verification

- Unpredictable, often diverges
- Restricted expressivity
Supervised Verification

- Predictable: solves decidable problems
- High expressivity

How to divide the problem between the human and the machine?
How to conduct the interaction?

Supervised Verification of Infinite-State Systems
IVy

IVy: https://github.com/Microsoft/ivy


Safety Verification

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

System S is safe iff there exists an **inductive invariant** $\text{Inv}$:

- $\text{Inv} \Rightarrow P = \neg \text{Bad}$ (Safety)
- $\text{Init} \Rightarrow \text{Inv}$ (Initiation)
- if $\sigma \models \text{Inv}$ and $\text{TR}(\sigma, \sigma')$ then $\sigma' \models \text{Inv}$ (Consecution)
System S is \textbf{safe} if all the reachable states satisfy the property \( P = \neg \text{Bad} \).

System S is safe iff there exists an \textbf{inductive invariant} \( \text{Inv} \):

\[
\begin{align*}
\text{Inv} & \Rightarrow P = \neg \text{Bad} & \text{(Safety)} \\
\text{Init} & \Rightarrow \text{Inv} & \text{(Initiation)} \\
\text{if } \sigma \models \text{Inv} \text{ and } \text{TR}(\sigma, \sigma') & \text{ then } \sigma' \models \text{Inv} & \text{(Consecution)}
\end{align*}
\]
Simple Example: Loop Invariants

\[ \begin{align*}
x &:= 1; \\
y &:= 2; \\
\text{while } \ast \text{ do } \\
&\quad \text{assert } \neg \text{even}[x]; \\
&\quad x := x + y; \\
&\quad y := y + 2 \\
\end{align*} \]
Simple Example: Loop Invariants

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

Counterexample to induction (CTI)
Simple Example: Loop Invariants

x := 1;
y := 2;
while * do {
  assert ¬even[x];
  x := x + y;
  y := y + 2
}

Inv = ¬even[x] ∧ even[y]
Challenges in Safety Verification

Infer inductive invariants for safety verification

1. **Formal specification**: reasoning about infinite-state systems
   - Modeling the system and the property (TR, Init, Bad)

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

3. **Inference**: inferring inductive invariants \((Inv)\)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable
IVy’s Approach: Supervised Verification

Infer inductive invariants for safety verification

1. Formal specification: reasoning about infinite-state systems
   - Modeling the system and the property (TR, Init, Bad)

2. Deduction: checking inductiveness
   - Undecidability of implication checking
     - Unbounded state (threads, messages), arithmetic, quantifiers,…

3. Inference: inferring inductive invariants (Inv)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable
How Does it Work?

• Specify systems and properties in **decidable fragment of first-order logic (EPR)**
  – Allows quantifiers to reason about unbounded sets
  – Decidable to check inductiveness
  – Finite counterexamples to induction, display graphically

• **Interact with the user to find inductive invariants**
  – by providing graphical UI for gradually strengthening the inductive invariant based on counterexamples to induction

• **Logic is mostly hidden**
  – Friendly to non-expert users
Example: Leader Election in a Ring

- Nodes are organized in a unidirectional ring
- Each node has a unique numeric id
- Protocol:
  - Each node sends its id to the next
  - A node that receives a message 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 the leader
- Theorem:
  - The protocol selects at most one leader

Modeling in IVy

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

Axiomatized in first-order logic

\[
\leq n_1 \neg L id_1 \quad \leq id_2 \\
\leq id_3 \quad \leq id_4 \\
\leq id_5 \\
\leq id_6
\]

\[

\frac{\langle n_5, n_1, n_3 \rangle \in I(\text{btw})}{pnd}
\]

[Diagram of protocol state and structure with nodes and arrows labeled with id and pending symbols]
Modeling in IVy

- \(\leq (ID, ID)\) – 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 msgs
- \(leader\) (Node) – node is a leader

```latex
action receive(n: Node, m: ID) {
  requires pending(m, n)
  pending(m, n) := *
  if id(n) = m then
    // found leader
    leader(n) := true
  else if id(n) \leq m then
    // pass message
    “s := next(n)”
    pending(id(n), s) := true
}

action send(n: Node) {
  “s := next(n)”
  pending(id(n), s) := true
}
```

protocol = (send | receive)*

assert I0 = \(\forall\) x, y: Node. leader(x) \& leader(y) \rightarrow x = y
Specify and verify the protocol for any number of nodes in the ring.
Example: Leader Election in a Ring

- Nodes are organized in a unidirectional ring.
- Each node has a unique numeric id.
- Protocol:
  - Each node sends its id to the next.
  - A node that receives a message 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 the 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.

Inductive Invariant for Leader Election

- \(\leq (\text{ID}, \text{ID})\) – 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 id
- \(\text{pending}(\text{ID}, \text{Node})\) – pending messages
- \(\text{leader}(\text{Node})\) – leader(n) means n is the leader

**Safety property:**

\[I_0 = \neg \text{Bad} = \forall x,y: \text{Node}. \ \text{leader}(x) \land \text{leader}(y) \rightarrow x = y\]

**Inductive invariant:**

\[\text{Inv} = I_0 \land I_1 \land I_2 \land I_3\]

\[I_1 = \forall n_1, n_2: \text{Node}. \ \text{leader}(n_2) \rightarrow \text{id}[n_1] \leq \text{id}[n_2]\]

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

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

The leader has the highest ID

Only highest id can be self-pnd

Cannot bypass higher nodes
Inductive Invariant for Leader Election

- $\leq (\text{ID, ID})$ – 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 id
- $\text{pending}(\text{ID, Node})$ – pending messages
- $\text{leader}(\text{Node})$ – leader(n) means n is the leader

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

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

How can we come up with an inductive invariant?
Invariant Inference in IVy

- Model
- Candidate Inductive Invariant $\text{Inv} = I_0 \land I_1 \land ... \land I_k$

- Inductive?
  - Yes: Inductive Invariant Found
  - No: Find "minimal" counterexample to induction (CTI)

- Modify candidate invariant

- Decidable
The figures illustrate the concepts of strengthening and weakening from CTI. A CTI of $\text{Inv}$ involves:

1. $\sigma \in \text{Inv}$
2. $\sigma' \not\in \text{Inv}$
3. $\sigma \rightarrow \sigma'$

- **Strengthening**
  - Moving from $\sigma$ to $\sigma'$ by including $\sigma$ in $\text{Inv}$.
  - Example: $\sigma$ from outside $\text{Inv}$ to inside $\text{Inv}$.

- **Weakening**
  - Moving from $\sigma'$ back to $\sigma$ by excluding $\sigma'$ from $\text{Inv}$.
  - Example: $\sigma'$ from inside $\text{Inv}$ to outside $\text{Inv}$.
σ, σ' are a CTI of Inv if:
- σ ∈ Inv
- σ' ∉ Inv
- σ → σ'

Strengthening & Weakening from CTI

σ, σ' are a CTI of Inv if:
- σ ∈ Inv
- σ' ∉ Inv
- σ → σ'

Strengthening

Weakening

Add a conjecture
Inv' := Inv ∧ “avoid(σ)"

Key Challenge: Generalization
Generalization using Diagram

Use **diagrams** to generalize from states

- state $\sigma$ is a **finite** first-order structure

$$\text{Diag}(\sigma) = \exists \ x \ y. \ x \neq y \land L(x) \land \neg L(y) \land \leq(x, y) \land \neg \leq(y, x) \land \leq(x, x) \land \leq(y, y)$$

$\sigma' \models \text{Diag}(\sigma)$ iff $\sigma$ is a substructure of $\sigma'$

$\sigma$ is obtained from $\sigma'$ by removing elements and projecting relations on remaining elements

Use **diagrams** to generalize:

- state $\sigma$ is a **finite** first-order

$$\text{Diag}(\sigma) = \exists x, y. \ x \neq y \land L(x) \land \neg L(y)$$
$$\land \leq (x, y) \land \neg \leq (y, x)$$
$$\land \leq (x, x) \land \leq (y, y)$$

$\sigma' \models \text{Diag}(\sigma) \iff \sigma$ is a substructure of $\sigma'$

$\sigma$ is obtained from $\sigma'$ by removing elements and projecting relations on remaining elements

avoid(\(\sigma\)) = \!-\! gen(\(\text{Diag}(\sigma)\))

\[
\begin{align*}
\text{gen}(\text{Diag}(\sigma)) &= \exists x, y \in \mathbb{E} \exists x \neq y \\
&\quad (x, y) \models (x \neq y) \\
&\quad (x, y) \models \langle x, x \rangle \leq \langle y, y \rangle \\
&\quad (x, y) \models \langle x, x \rangle \leq (x, y) \\
&\quad (x, y) \models \langle x, y \rangle \leq (x, y) \\
&\quad \exists x, y \models x \neq y \\
&\quad \exists x, y \models (x, y) \models \langle x, x \rangle \leq \langle y, y \rangle \\
&\quad \exists x, y \models (x, y) \models \langle x, x \rangle \leq (x, y) \\
&\quad \exists x, y \models (x, y) \models \langle x, y \rangle \leq (x, y)
\end{align*}
\]

\[
\begin{align*}
\text{Diag}(\sigma) &= \exists x, y \in \mathbb{E} x \neq y \\
&\quad \exists x := (x, y) \models \langle x, x \rangle \leq \langle y, y \rangle \\
&\quad \exists x := (x, y) \models \langle x, x \rangle \leq (x, y) \\
&\quad \exists x := (x, y) \models \langle x, y \rangle \leq (x, y) \\
&\quad \exists x := (x, y) \models x \neq y \\
&\quad \exists x := (x, y) \models (x, y) \models \langle x, x \rangle \leq \langle y, y \rangle \\
&\quad \exists x := (x, y) \models (x, y) \models \langle x, x \rangle \leq (x, y) \\
&\quad \exists x := (x, y) \models (x, y) \models \langle x, y \rangle \leq (x, y)
\end{align*}
\]

Can generalize more
- remove facts/conjuncts

Generalization using Diagram

...
From Diagrams to Invariants

$$\exists \bar{x}. \neg l_{1,1}(\bar{x}) \land \ldots \land \neg l_{1,m}(\bar{x})$$
From Diagrams to Invariants

\[
\neg \exists \bar{x}. (\neg l_{1,1}(\bar{x}) \land \ldots \land \neg l_{1,m}(\bar{x}))
\]

conjecture
From Diagrams to Invariants

conjecture

\[ \text{Inv} \equiv \neg \exists \bar{x}. (\neg l_{1,1}(\bar{x}) \land \ldots \land \neg l_{1,m}(\bar{x})) \land \ldots \land \neg \exists \bar{x}. (\neg l_{n,1}(\bar{x}) \land \ldots \land \neg l_{n,m}(\bar{x})) \]

Q: How to select which facts to remove in the generalization?

IVy: **interact** with the user to identify **irrelevant facts**
Leader Election: Iteration 3

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

Safety property:

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

Inductive invariant: \( Inv = I0 \land I1 \land I2 \land I3 \)

\( I_1 = \forall n_1,n_2: \text{Node}. \text{leader}(n_2) \rightarrow \text{id}[n_1] \leq \text{id}[n_2] \)

\( I_2 = \forall n_1,n_2: \text{Node}. \text{pnd}(\text{id}[n_2], n_2) \rightarrow \text{id}[n_1] \leq \text{id}[n_2] \)

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

\[ \rightarrow \text{id}[n_3] \leq \text{id}[n_2] \]

1. Each node sends its id to the next
2. A node that receives a msg passes it to the next node in the ring if the id in the msg \( \geq \) the node’s id
3. A node that receives its own id becomes the leader
Leader Election: Iteration 3

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

Safety property:

\[ I_0 = \neg \text{Bad} = \forall x,y: \text{Node.} \ \text{leader}(x) \land \text{leader}(y) \rightarrow x = y \]

Inductive invariant:

\[ I_1 = \forall n_1,n_2: \text{Node.} \ \text{leader}(n_2) \rightarrow \text{id}[n_1] \leq \text{id}[n_2] \]

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

\[ I_3 = \forall n_1,n_2,n_3: \text{Node.} \ \text{btw}(n_1,n_2,n_3) \land \text{pending}(\text{id}[n_2], n_1) \rightarrow \text{id}[n_3] \leq \text{id}[n_2] \]
IVy: Check Inductiveness

Leader Protocol

Init \land \neg Inv

Inv(V) \land TR(V,V') \land \neg Inv(V')

Inv(V) \land \text{Bad}(V)

VC Generator

Inv = I_0 \land I_1 \land I_2

I_0: leader is unique
I_1: The leader has the highest ID
I_2: Only highest id can be self-pnd

EPR Solver

I_0 \land I_1 \land I_2

rcv(1, id(2))

CTI

id_1 \leq id_2 \leq id_3

\neg I_2

id_1 \leq id_2 \leq id_3

id_1 \leq id_2 \leq id_3
IVy: Generalize from CTI

1. Each node **sends** its id to the next
2. A node that **receives** a msg passes it to the next node in the ring if the id in the msg ≥ the node’s id
3. A node that receives its own id becomes the **leader**

id[n₂] is pending for n₁, had to go through n₃
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization

1 ≤ id
2 ≤ id
3 ≤ id

I0∧I1∧I2

id1 ≤ id2 ≤ id3

Cannot bypass nodes with higher ids

User’s Generalization
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

User’s Generalization
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

User’s Generalization

I0 ∧ I1 ∧ I2
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

Project to \{\leq, id, pnd\}

\begin{itemize}
\item \leq
\item id
\item pnd
\item L
\end{itemize}
IVy: Generalize from CTI

\[ \neg \exists n_1, n_2, n_3 : \text{Node.} \neq (n_1, n_2, n_3) \land \\
\neq (id[n_1], id[n_2], id[n_3]) \land \\
id[n_1] \leq id[n_2] \leq id[n_3] \land \\
pnd(id[n_2], n_1) \]

Cannot bypass nodes with higher ids
IVy: Generalize from CTI

\[ \text{Project to } \{\leq, \text{id}, \text{pnd}\} \]

\[ \neg \exists n_1, n_2, n_3 : \text{Node. } \neq (n_1, n_2, n_3) \land \]
\[ \neq (\text{id}[n_1], \text{id}[n_2], \text{id}[n_3]) \land \]
\[ \text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \land \]
\[ \text{pnd}(\text{id}[n_2], n_1) \]

BMC VC Generator (K=3, \neg C_3)

Init(V_0) \land \text{TR}(V_0, V_1) \land \text{TR}(V_1, V_2) \land \text{TR}(V_1, V_3) \land \neg C_3(V_3)

Counterexample Trace

Cannot bypass nodes with higher ids
IVy: Generalize from CTI

¬∃n₁, n₂, n₃: Node. n₁ ≠ (n₂, n₃) \land id[n₁] ≤ id[n₂] \land id[n₂] ≤ id[n₃] \land pnd(id[n₂], n₁) \land btw(n₁, n₂, n₃)

Cannot bypass nodes with higher ids
IVy: Generalize from CTI

Cannot bypass nodes with higher ids

¬∃n₁, n₂, n₃: Node. ≠(n₁,n₂,n₃) ∧
≠(id[n₁],id[n₂],id[n₃]) ∧
id[n₁] ≤ id[n₂] ≤ id[n₃] ∧
pnd(id[n₂], n₁) ∧ btw(n₁, n₂, n₃)

BMC VC Generator (K=3, ¬C₃)

Init(V₀) ∧ TR(V₀,V₁) ∧ TR(V₁,V₂) ∧ TR(V₁,V₃) ∧ ¬C₃(V₃)

EPR Solver

Proof
IVy: Generalize from CTI

\[ \text{Cannot bypass nodes with higher ids} \]

\[ \neg \exists n_1, n_2, n_3: \text{Node. } \not= (n_1,n_2,n_3) \land \]
\[ \not= (\text{id}[n_1],\text{id}[n_2],\text{id}[n_3]) \land \]
\[ \text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \land \]
\[ \text{pnd}(\text{id}[n_2], n_1) \land \text{btw}(n_1, n_2, n_3) \]

Proof
IVy: Generalize from CTI

\[\text{I0} \land \text{I1} \land \text{I2}\]

\[\begin{array}{c}
\text{id}_1 \\
1
\end{array} \leq_p \begin{array}{c}
\text{id}_2 \\
2
\end{array} \leq_p \begin{array}{c}
\text{id}_3 \\
3
\end{array}
\]

\[\text{Cannot bypass nodes with higher ids}\]

Project to \{\leq, \text{id}, \text{pnd}, \text{btw}\}

\[\neg \exists n_1, n_2, n_3 : \text{Node. } \not\equiv (n_1, n_2, n_3) \land \not\equiv (\text{id}[n_1], \text{id}[n_2], \text{id}[n_3]) \land \text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \land \text{pnd} (\text{id}[n_2], n_1) \land \text{btw} (n_1, n_2, n_3)\]

Interp(3)

\[\neg \exists n_1, n_2, n_3 : \text{Node. } \text{id}[n_2] \leq \text{id}[n_3] \land \text{btw} (n_1, n_2, n_3) \land \text{pnd} (\text{id}[n_2], n_1)\]

Proof
IVy: Generalize from CTI

\[ \text{Cannot bypass nodes with higher ids} \]

\[ \neg \exists n_1, n_2, n_3 : \text{Node. } \neq (n_1, n_2, n_3) \wedge \]
\[ \neq (\text{id}[n_1], \text{id}[n_2], \text{id}[n_3]) \wedge \]
\[ \text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \wedge \]
\[ \text{pnd(\text{id}[n_2], n_1)} \wedge \text{btw}(n_1, n_2, n_3) \]

\[ \text{Looks good, add to the invariant as } I_3 \]

\[ \neg \exists n_1, n_2, n_3 : \text{Node. } \text{id}[n_2] \leq \text{id}[n_3] \wedge \]
\[ \text{btw}(n_1, n_2, n_3) \wedge \text{pnd(\text{id}[n_2], n_1)} \]
IVy: Check Inductiveness

Leader protocol

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

VC Generator

Init \land \neg Inv

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, which proves the protocol is safe
Recap: Supervised Verification in IVy

Decidable Problems
Predictable Automation

Proof intuition and creativity
Graphical interaction

Check inductiveness
BMC
Interpolation

Projection of relevant facts
BMC bounds
Examining conjectures
Challenge: How to use restricted first-order logic to verify interesting systems?

(1) Limitations of first-order logic
   • Expressing transitive closure
     • Ring protocols
   • Expressing arithmetic
     • Node id’s

Domain knowledge and axioms
# Axioms: Leader Election Protocol

- $\leq$ (ID, ID) – total order on node id’s
- $\text{btw}$ (a: Node, b: Node, c: 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

<table>
<thead>
<tr>
<th>Node ID’s</th>
<th>Intention</th>
<th>EPR Modeling</th>
</tr>
</thead>
</table>
| **Integers** | $\forall i: \text{ID. } i \leq i$ Reflexive  
$\forall i, j, k: \text{ID. } i \leq j \land j \leq k \rightarrow i \leq k$ Transitive  
$\forall i, j: \text{ID. } i \leq j \land j \leq l \rightarrow i \equiv j$ Anti-Symmetric  
$\forall i, j: \text{ID. } i \leq j \lor j \leq i$ Total  
$\forall x, y: \text{Node. } \text{id}(x) = \text{id}(y) \rightarrow x = y$ Injective  
| **Next edges + Transitive closure** | $\forall x, y, z: \text{Node. } \text{btw}(x, y, z) \rightarrow \text{btw}(y, z, x)$ Circular shifts  
$\forall x, y, z, w: \text{Node. } \text{btw}(w, x, y) \land \text{btw}(w, y, z) \rightarrow \text{btw}(w, x, z)$ Transitive  
$\forall x, y, w: \text{Node. } \text{btw}(w, x, y) \rightarrow \neg \text{btw}(w, y, x)$ Anti-Symmetric  
$\forall x, y, z, w: \text{Node. } \text{distinct}(x, y, z) \rightarrow \text{btw}(w, x, y) \lor \text{btw}(w, y, x)$ |

“next(a)=b” $\equiv \forall x: \text{Node. } x \neq a \land x \neq b \rightarrow \text{btw}(a, b, x)
Challenge: How to use restricted first-order logic to verify interesting systems?

(1) Limitations of first-order logic
   • Expressing transitive closure
     • Ring protocols
   • Expressing arithmetic
     • Node id’s
   • Expressing Consensus
     • Paxos, Multi-Paxos, Reconfiguration

(2) Restrictions for decidability
   • Restricted quantification

Domain knowledge and axioms

Derived relations and rewrites

## IVy: Verified Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model (# LOC)</th>
<th>Property (# Literals)</th>
<th>Invariant (# Literals)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader in Ring</td>
<td>59</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>50</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>DB Chain Replication</td>
<td>143</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>Chord</td>
<td>155</td>
<td>35</td>
<td>46</td>
</tr>
<tr>
<td>Lock Server (500 Coq lines [Verdi])</td>
<td>122</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Distributed Lock (1 week [IronFleet])</td>
<td>41</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>Single Decree Paxos</td>
<td>85</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>Multi Paxos</td>
<td>102</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>Vertical Paxos</td>
<td>123</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>Fast Paxos</td>
<td>117</td>
<td>3</td>
<td>59</td>
</tr>
<tr>
<td>Flexible Paxos</td>
<td>88</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>Stoppable Paxos</td>
<td>130</td>
<td>6</td>
<td>60</td>
</tr>
<tr>
<td>Virtually Synchronous Paxos</td>
<td></td>
<td></td>
<td>Work in progress</td>
</tr>
</tbody>
</table>
Summary

- Safety verification by
  - Automatic deduction
  - Interactive inference of invariants, graphical interaction
- Use decidable fragment of FOL
  - Deduction is decidable
  - Finite Counterexamples
- Interact with a user based on counterexamples to induction
- Surprisingly powerful
  - Paxos, Multi-Paxos, Reconfiguration, ... [OOPSLA’17]
  - Liveness and Temporal Properties [POPL’18]

Supervised Verification of Infinite-State Systems
Future Work

• More distributed systems
• Other logics
• Other inference schemes
• Other forms of interaction
• More automation in inferring inductive invariants
• Theoretical understanding of limitations and tradeoffs

Seeking postdocs and students

Supervised Verification of Infinite-State Systems