Ivy: Safety Verification by Interactive Generalization

Oded Padon

Verification Day
1-June-2016

Motivation

• Software is everywhere
• Distributed systems are everywhere
• Verification is needed to ensure safety of critical systems
Why Verify Distributed Systems?

• Distributed systems are notoriously hard to get right
• Bugs occur on rare scenarios
  • Hard to test/reproduce
  • Testing covers tiny fraction of behaviors
  • Leaves most bugs for production
• Even small protocols can be tricky  

Using Lightweight Modeling To Understand Chord

SIGCOMM’01
Chord: A Scalable Peer for Internet

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
System S is **safe** if no bad state is reachable

\[
R_0 = \text{Init} \quad \text{— Initial states, reachable in 0 transitions}
\]

\[
R_{i+1} = R_i \cup \{\sigma' \mid \sigma \xrightarrow{\sigma} \sigma' \text{ and } \sigma \in R_i\}
\]

\[
R = R_0 \cup R_1 \cup R_2 \cup \ldots
\]

Safety: \( R \cap \text{Bad} = \emptyset \)

K-Safety: \( R_K \cap \text{Bad} = \emptyset \)
System S is safe if no bad state is reachable
System S is safe iff there exists an inductive invariant \( \text{Inv} \) s.t.:

\[
\text{Inv} \cap \text{Bad} = \emptyset \quad (\text{Safety})
\]

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

if \( \sigma \in \text{Inv} \) and \( \sigma \xrightarrow{} \sigma' \) then \( \sigma' \in \text{Inv} \) (Consecution)
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:
- $\sigma \not\in \text{Inv}$

Weakening:
- $\sigma' \not\in \text{Inv}$
- $\sigma \in \text{Inv}$

Diagram:
- $\text{Inv}'$
- $\sigma$ to $\sigma'$
- $\sigma'$ to $\sigma$
Modeling with Logic

• SAT/SMT has made huge progress in the last decade
• Great impact on verification: Z3, Dafny, IronClad/IronFleet, and more

• State: finite first-order structure over vocabulary V
• Initial states and safety property (first-order formulas):
  • Init(V) – initial states
  • Bad(V) – bad states
• Transition relation:
  first-order formula TR(V, V’)
  V’ is a copy of V describing the next state

Inductive Invariant

\( \text{Inv} \) is an **inductive invariant** if:

- **Initiation:** \( \text{Init} \Rightarrow \text{Inv} \)
- **Safety:** \( \text{Inv} \Rightarrow \neg \text{Bad} \)
- **Consecution:** \( \text{Inv} \land \text{TR} \Rightarrow \text{Inv}' \)

\[ \begin{align*}
\text{Init} \land \neg \text{Inv} & \text{ unsat} \\
\text{Inv} \land \text{Bad} & \text{ unsat} \\
\text{Inv} \land \text{TR} \land \neg \text{Inv}' & \text{ unsat}
\end{align*} \]
Challenges

1. Formal specification:
   - Modeling the system (TR, Init)
   - Formalizing the safety property (Bad)
   - Specifying in logic

2. Deduction – Checking inductiveness
   - Undecidability of implication checking
     - Arithmetic, quantifier alternation, unbounded state

3. Inductive Invariants for Deductive Verification (Inv)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable
Existing approaches

• **Automated invariant inference**
  • Model checking
    • Exploit finite state / finite abstraction
  • Abstract Interpretation
    • Sound abstraction
    • Limited for infinite state systems due to undecidability
• 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)
  • Programmer gives inductive invariant and proves it
  • Huge programmer effort (~50 lines of proof per line of code)
Our Approach in Ivy

• Restrict the specification language for decidability
  • Deduction is decidable with SAT solvers
  • Challenge: model systems in a restricted language

• Finding inductive invariants:
  • Combine automated techniques with human guidance
  • Key: generalization from counterexamples to induction
  • Graphical user interaction
  • Decidability allows reliable automated checks
Expressiveness vs. Automation

<table>
<thead>
<tr>
<th>Coq</th>
<th>Dafny</th>
<th>Ivy</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant</td>
<td>User</td>
<td>User</td>
<td>User + System</td>
</tr>
<tr>
<td>Deduction</td>
<td>User</td>
<td>System (Z3) + “User”</td>
<td>System (EPR Z3)</td>
</tr>
</tbody>
</table>
Relational Modeling Language (RML)

- Designed to make verification tasks decidable
  - Yet expressive enough to model systems
- Finite relations and stratified function symbols
  - Used to describe system state
    - E.g., pending packets, nodes’ local data structures
  - Also used to record ghost information
  - Stratification: function $f: A \to B$, so no function $g: B \to A$
- Universally quantified axioms
  - Total orders, partial orders, lists, trees, rings, quorums, ...
- No numerics
- Simple (quantifier-free) updates
- Imperative constructs with non-determinism
- Turing-Complete
- Universal inductive invariants are decidable to check
Effectively Propositional Logic – EPR
a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic
  - Restricted quantifier prefix: $\exists^* \forall^* \phi_{Q.F.}$
  - No $\forall^* \exists^*$
- No function symbols
  - Possible to add *stratified* function symbols
- No arithmetic
- Small model property
  - $\exists x_1, \ldots, x_n. \forall y_1, \ldots, y_m. \phi_{Q.F.}$ has a model iff it has a model of at most $n+k$ elements ($k$ - number of constant symbols)
- Satisfiability is decidable
  - NEXPTIME / $\Sigma_2$
- Supported by theorem provers (e.g., Z3, iProver, Vampire)

Using EPR for Verification

• System Model
  • $V$ – vocabulary with relations and stratified function symbols
  • $TR(V, V')$ – transition relation
  • $Init(V)$ – initial states
  • $Bad(V)$ – bad states (e.g. assertion violation)
  • $Inv(V)$ is an inductive invariant if:
    • $Init(V) \Rightarrow Inv(V)$
    • $Inv(V) \land TR(V, V') \Rightarrow Inv(V')$
    • $Inv(V) \Rightarrow \neg Bad(V)$

Alternation-Free: Boolean combination of closed $\exists^* \land \forall^*$ formulas

Decidable to check

$\text{SAT}(\exists^* \land \forall^*)$
Example: Leader Election in a Ring

• Nodes are organized in a 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

Example: Leader Election in a Ring

- Nodes are organized in a 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.
Leader Election Protocol (RML)

- $\preceq (ID, ID) –$ 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

while true do {

\[
\begin{align*}
\text{// send(n1)} \\
\text{n1 := *;} \\
\text{n2 := *;} \\
\text{assume next(n1, n2);} \\
\text{pending.insert(id[n1], n2)}
\end{align*}
\]

\[
\begin{align*}
\text{// receive(n1)} \\
\text{m, n1 := pending.remove();} \\
\text{if id[n1] = m then:} \\
\text{\hspace{1cm} // found leader} \\
\text{leader.insert(n1)} \\
\text{else if id[n1] \preceq m then:} \\
\text{\hspace{1cm} // pass message} \\
\text{\hspace{1cm} n2 := *;} \\
\text{\hspace{1cm} assume next(n1, n2);} \\
\text{\hspace{1cm} pending.insert(m, n2)}
\end{align*}
\]

conjecture $I_0 = \neg \exists x, y : \text{Node. } x \neq y \land \text{leader}(x) \land \text{id}[x] \preceq \text{id}[y]$
Bounded Model Checking (BMC)

Leader Protocol

Bound $k$

Safety Property $I_0$: At most one leader

BMC VC Generator

Verification Condition:
\[ \text{Init}(V_0) \land \text{TR}(V_0, V_1) \land \ldots \land \text{TR}(V_{k-1}, V_k) \land \neg I_0(V_k) \]

EPR Solver

Counterexample Trace

Proof
BMC(2)
Leader Protocol – 2nd attempt

Axiom \( \forall x, y: \text{Node}. \ id[x] = id[y] \Rightarrow x = y \)

- BMC(1) – OK
- BMC(2) – OK
- BMC(3) – OK
- BMC(4) – OK
- BMC(5) – OK
- BMC(6) – OK
- BMC(7) – OK
- BMC(8) – OK

Looks good, let’s find an inductive invariant!
Invariant Inference in Ivy

Model

Candidate Inductive Invariant

Inductive Invariant Found

Inductive?

Yes

Modify candidate invariant

No

Find “minimal” CTI

Generalize from CTI

User

Automation
1. Generalize by removing facts to form a conjecture
   • User graphically selects which facts to remove
2. Check if the conjecture is true up to K: BMC(K)
   • User determines the right K to use
   • Ivy uses a SAT solver - sound & complete
3. Automatically remove more facts: Interpolate(K)
   • Ivy uses the SAT solver to discover more facts that can be removed
   • User examines the result – it could be wrong
Algorithmic Deductive Verification (1)

Leader Protocol

Inv = $I_0$

Bad = $\neg I_0$

VC Generator

$\text{Init} \land \neg \text{Inv}$

$\text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V')$

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

EPR Solver

CTI

$n_1 \rightarrow \neg L$ -> $\text{id}_1$ -> $\text{id}$ -> $n_2 \rightarrow \neg L$ -> $\text{id}_2$

$n_1 \rightarrow \neg L$ -> $\text{id}_1$ -> $\text{id}$ -> $n_2 \rightarrow \neg L$ -> $\text{id}_2$

$pnd$ -> $\text{id}$ -> $\text{id}$

$\text{id}$ -> $\text{id}$

$\text{rcv}(n_1)$ -> $\text{id}$

next

next
1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node’s own id
3. A node that receives its own id becomes the leader
Generalize from CTI (1)

Only the highest id can be self pending

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

C_1 = \neg \exists n_1, n_2: \text{Node. } n_1 \neq n_2 \land \text{pnd}(id[n_1], n_1) \land id[n_1] \neq id[n_2] \land id[n_1] \leq id[n_2]

BMC VC Generator (K=3, C_1)

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_1(V_3)

EPR Solver

Proof
Generalize from CTI (1)

Only the highest id can be self pending

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

Lookd good, add to the invariant as \(I_1\)

\[I_1 = \neg \exists n_1, n_2: \text{Node. } pnd(id[n_1], n_1) \land id[n_1] \neq id[n_2] \land id[n_1] \leq id[n_2]\]

\[C_1 = \neg \exists n_1, n_2: \text{Node. } n_1 \neq n_2 \land pnd(id[n_1], n_1) \land id[n_1] \neq id[n_2] \land id[n_1] \leq id[n_2]\]

\[EPR \text{ Solver}\]

Proof + Minimal UNSAT core
Algorithmic Deductive Verification (2)

Leader Protocol

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

Bad = \neg I_0

VC Generator

\[ \text{Init} \land \neg \text{Inv} \]

\[ \text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V') \]

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

EPR Solver

CTI

\[ I_0 \land I_1 \]

\[ \neg I_1 \]

\( \text{id}_1 \)

\( \text{id}_2 \)

\( \text{id}_2 \)

\( \text{n}_1 \) \( \neg \text{L} \)

\( \text{n}_2 \) \( \neg \text{L} \)

\( \text{n}_3 \) \( \neg \text{L} \)

\( \text{id} \)

\( \text{next} \)

\( \text{pnd} \)

\( \text{rcv}(n_1) \)

\( \text{id} \)

\( \text{next} \)

\( \text{id} \)

\( \text{next} \)

\( \text{id} \)
1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node’s own id
3. A node that receives its own id becomes the leader
Generalize from CTI (2)

Cannot bypass nodes with higher ids

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

C_2 = \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 
\text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \land 
pnd(\text{id}[n_2], n_1)

BMC VC Generator (K=3, C_2)

Init(V_0) \land TR(V_0, V_1) \land TR(V_1, V_2) \land TR(V_1, V_3) \land \neg C_2(V_3)

EPR Solver

Counterexample Trace
Cannot bypass nodes with higher ids

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

C_2 = \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 
pnd(\text{id}[n_2], n_1) \land \text{btw}(n_1, n_2, n_3)

BMC VC Generator (K=3, C_2)

\text{Init}(V_0) \land TR(V_0, V_1) \land TR(V_1, V_2) \land TR(V_1, V_3) \land \neg C_2(V_3)

EPR Solver

Proof
Generalize from CTI (2)

Cannot bypass nodes with higher ids

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

This looks good, add to the invariant as \( I_2 \)

\( I_2 : = \neg \exists n_1, n_2, n_3 : \text{Node. } \text{btw}(n_1, n_2, n_3) \land \text{pnd}(id[n_2], n_1) \land id[n_2] \leq id[n_3] \)
Algorithmic Deductive Verification (3)

$\text{Inv} = I_0 \land I_1 \land I_2$

$\text{Bad} = \neg I_0$

$\text{VC Generator}$

$\text{Init} \land \neg \text{Inv}$

$\text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V')$

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

$\text{EPR Solver}$

Proof

$I_0 \land I_1 \land I_2$ is an inductive invariant for the leader protocol, which proves the protocol is safe
Completeness and Interaction Complexity

- Any generalization from CTI adds one universal clause to the invariant
- The invariant is constructed in CNF
- If there is a universal invariant with $N$ clauses, it can be obtained by the user in $N$ generalization steps
  - Assuming the user is optimal
- If the user is sub-optimal, backtracking (weakening) may be needed
## Verified Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model Types</th>
<th>Relations &amp; Functions</th>
<th>Property (# Literals)</th>
<th>Invariant (# Literals)</th>
<th>CTI Gen. Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader in Ring</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>DB Chain Replication</td>
<td>4</td>
<td>13</td>
<td>11</td>
<td>35</td>
<td>7</td>
</tr>
<tr>
<td>Chord</td>
<td>1</td>
<td>13</td>
<td>35</td>
<td>46</td>
<td>4</td>
</tr>
<tr>
<td>Lock Server 500 Coq lines [Verdi]</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>21</td>
<td>8 (1h)</td>
</tr>
<tr>
<td>Distributed Lock 1 week [IronFleet]</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>26</td>
<td>12 (1h)</td>
</tr>
<tr>
<td>Paxos</td>
<td></td>
<td></td>
<td></td>
<td>Work in progress</td>
<td></td>
</tr>
<tr>
<td>Raft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ivy Summary & Lessons Learned

• Ivy:
  • RML – modeling language that makes deduction decidable
  • Interactive generalization for finding inducting invariants
  • Application to the domain of distributed protocols
• User intuition and machine heuristics complement each other:
  • User has the ability to ignore irrelevant facts and intuition that leads to better generalizations
  • Machine is better at finding bugs and corner cases
• The safety of many protocols can be proven w/o reasoning about arithmetic operations or set cardinalities
  • Many important properties can be captured in a (parametric) model that abstracts the actual numerical values
  • Unbounded topologies
  • Unique paths (ring, trees, ...)

• Ivy:
  • RML – modeling language that makes deduction decidable
  • Interactive generalization for finding inducting invariants
  • Application to the domain of distributed protocols

Current & Future Work

- More expressiveness, keeping decidability
  - System, Spec, Invariant (proof)
- Verifying more systems (Paxos and Raft are next)
- Inferring inductive invariants (more) automatically
- Better theoretical understanding of limitations and tradeoffs