Verification of Distributed Protocols Using Decidable Logic

Sharon Shoham

Tel Aviv University

Programming Languages Mentoring Workshop 2019

The research leading to these results has received funding from the European Research Council under the European Union’s Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS])
Why verify distributed protocols?

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

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

Verify distributed protocols for any number of nodes and resources

• Infinite state-space
  • unbounded #processes
  • unbounded #messages
  • unbounded #objects

• Asymptotic complexity of verification
  • Rice theorem

I can’t decide!
System S is \textbf{safe} if all the \textit{reachable} states satisfy the property \(P = \neg \textit{Bad}\).
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 $\text{Inv}$:

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

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

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

\[
\text{Inv} \cap \text{Bad} = \emptyset \quad \text{(Safety)}
\]
• N pairs of players pass a ball:
  – 1↑ will pass to 1↓
  – 1↓ will pass to 1↑
  – 2↑ will pass to 2↓
  – 2↓ will pass to 2↑ …
Example

- N pairs of players pass a ball:
  - \(1\uparrow\) will pass to \(1\downarrow\)
  - \(1\downarrow\) will pass to \(1\uparrow\)
  - \(2\uparrow\) will pass to \(2\downarrow\)
  - \(2\downarrow\) will pass to \(2\uparrow\)  
  ...
Example

- N pairs of players pass a ball:
  - $1^\uparrow$ will pass to $1^\downarrow$
  - $1^\downarrow$ will pass to $1^\uparrow$
  - $2^\uparrow$ will pass to $2^\downarrow$
  - $2^\downarrow$ will pass to $2^\uparrow$ ...
- The ball starts at player $1^\uparrow$
- Can the ball get to $2^\downarrow$?
Example

- N pairs of players pass a ball:
  - $1^\uparrow$ will pass to $1^\downarrow$
  - $1^\downarrow$ will pass to $1^\uparrow$
  - $2^\uparrow$ will pass to $2^\downarrow$
  - $2^\downarrow$ will pass to $2^\uparrow$ ...

- The ball starts at player $1^\uparrow$

- Can the ball get to $2^\downarrow$?

- Is “the ball is not at $2^\downarrow$” an inductive invariant?
Example

- N pairs of players pass a ball:
  - $1^\uparrow$ will pass to $1^\downarrow$
  - $1^\downarrow$ will pass to $1^\uparrow$
  - $2^\uparrow$ will pass to $2^\downarrow$
  - $2^\downarrow$ will pass to $2^\uparrow$ ...
- The ball starts at player $1^\uparrow$
- Can the ball get to $2^\downarrow$?
- **Is “the ball is not at $2^\downarrow$” an inductive invariant? No!**
  - Counterexample to induction
Example

• N pairs of players pass a ball:
  – 1↑ will pass to 1↓
  – 1↓ will pass to 1↑
  – 2↑ will pass to 2↓
  – 2↓ will pass to 2↑ ...

• The ball starts at player 1↑
• Can the ball get to 2↓?
• Is “the ball is not at 2↓” an inductive invariant? No!
  – Counterexample to induction
• Inductive invariant: “the ball is not at 2↑ nor 2↓”
Logic-based verification

Provers/solvers for different logics made huge progress

• Propositional logic (SAT) – industrial impact for hardware verification
• Satisfiability modulo theories (SMT) – major trend in software verification
• Automated first-order theorem provers
• Interactive theorem provers
• Z3, CVC4, iProver, Vampire, Coq, Isabelle/HOL ....
Inv(V) is an inductive invariant if the verification conditions (VCs) are valid:

**Initiation** \( \text{Init}(V) \Rightarrow \text{Inv}(V) \)  
\( \text{unsat}( \text{Init}(V) \land \neg \text{Inv}(V) ) \)

**Cons.** \( \text{Inv}(V) \land \text{TR}(V,V') \Rightarrow \text{Inv}(V') \)  
\( \text{unsat}( \text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V') ) \)

**Safety** \( \text{Inv}(V) \Rightarrow \neg \text{Bad}(V) \)  
\( \text{unsat}( \text{Inv}(V) \land \text{Bad}(V) ) \)

Represent \( \text{Init}, \text{Tr}, \text{Bad}, \text{Inv} \) by logical formulas: **Formula \( \Leftrightarrow \) Set of states**
Challenges for logic-based verification

**Formal specification**
Modeling the system and its invariants

**Deduction**
Checking validity of the VCs

**Inference**
Finding an inductive invariant
Inv(V) is an **inductive invariant** if the following **verification conditions** are valid:

- **Initiation**: Init(V) \implies Inv(V)\quad \text{unsat}(\ Init(V) \land \neg Inv(V) )
- **Cons.**: Inv(V) \land TR(V,V') \implies Inv(V')\quad \text{unsat}(\ Inv(V) \land TR(V,V') \land \neg Inv(V') )
- **Safety**: Inv(V) \implies \neg Bad(V)\quad \text{unsat}(\ Inv(V) \land Bad(V) )

**Are the logical VC's valid?**

- **Counterexample**
- **Unknown / Diverge**
- **Proof**

---

**Church’s Theorem**

I can’t decide!
Interactive theorem provers (Coq, Isabelle/HOL, LEAN)
- Programmer proves the inductive invariant
- Huge programmer effort (~10-50 lines of proof per line of code)

Automatic solvers/provers (e.g. Z3, CVC4, Vampire)
- VCs discharged automatically
- Tools may diverge (for SMT: matching loops, arithmetic)
- Unpredictability (butterfly effect)

e.g. Verdi

e.g. Ironfleet
Logic-based verification approaches

- Interactive theorem provers
  - Huge programmer effort (~10-50 lines of proof per line of code)

- Automated deductive verification
  - SMT solver may diverge (matching loops, arithmetic)
  - Unpredictability, butterfly effect

- Model Checking, Abstract Interpretation
  - Limited due to undecidability
Logic-based verification approaches

Expressiveness

- Interactive theorem provers
- Huge programmer effort (~10-50 lines of proof per line of code)
- SMT solver may diverge (matching loops, arithmetic)
- Unpredictability, butterfly effect
- Limited due to undecidability

Desired
- Expressiveness
- High degree of automation
- Predictability
- Comprehensibility for users
- Efficiency/scalability

Automation
This talk: Restrict VC’s to decidable logic

Inv(V) is an inductive invariant if the following verification conditions are valid:

- **Initiation** \( \text{Init}(V) \implies \text{Inv}(V) \) \( \text{unsat}( \text{Init}(V) \land \neg \text{Inv}(V) ) \)
- **Cons.** \( \text{Inv}(V) \land \text{TR}(V,V') \implies \text{Inv}(V') \) \( \text{unsat}( \text{Inv}(V) \land \text{TR}(V,V') \land \neg \text{Inv}(V') ) \)
- **Safety** \( \text{Inv}(V) \implies \neg \text{Bad}(V) \) \( \text{unsat}( \text{Inv}(V) \land \text{Bad}(V) ) \)

Are the logical VC’s valid? ∈ Decidable logic

With good tool support
Challenges for verification with decidable logic

Formal specification
Modeling in a decidable logic

Deduction
Checking validity of the VC’s

Invariant inference
Finding an inductive invariant
This talk

Logic: **EPR** – decidable fragment of first order logic

**Formal specification**
- Surprisingly expressive

**Invariant inference**
- Automatic (based on PDR)
  - Semi-algorithm: may diverge
- Interactive
  - Based on graphically displayed counterexamples to induction
Effectively Propositional Logic – EPR

Decidable fragment of first order logic

+ Quantification ($\exists^* \forall^*$) - Theories (e.g., arithmetic)

- Allows quantifiers to reason about unbounded sets
  - $\forall x,y. \text{leader}(x) \land \text{leader}(y) \rightarrow x = y$

- Satisfiability is decidable $\Rightarrow$ Deduction is decidable

- Small model property $\Rightarrow$ Finite cex to induction

- Turing complete modeling language

- Limited language for safety and inductive invariants
  - Suffices for many infinite-state systems
Successful verification with EPR

- **Shape Analysis**
  [Itzhaky et al. CAV’13, POPL’14, CAV’14, CAV’15]

- **Software-Defined Networks**
  [Ball et al. PLDI’14]

- **Distributed protocols**
  [Padon et al. PLDI’16, OOPSLA’17, POPL’18, PLDI’18]

- **Concurrent Modification Errors in Java programs**
  [Frumkin et al. VMCAI’17]
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 with EPR

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

Axiomatized in EPR
Modeling with EPR

- **State**: finite first-order structure over vocabulary $V$ (+ axioms)

- **Initial states and safety** property: EPR formulas over $V$
  - $\text{Init}(V)$ – initial states, e.g., $\forall \text{id, n. \neg pending}(\text{id, n})$
  - $\text{Bad}(V)$ – bad states, e.g., $\exists n_1, n_2. \text{leader}(n_1) \land \text{leader}(n_2) \land n_1 \neq n_2$

- **Transition relation**: expressed as EPR formula $\text{TR}(V, V')$, e.g.:
  - $\exists n, s. \text{“s = next(n)”} \land \forall x, y. \text{pending’}(x, y) \leftrightarrow (\text{pending}(x, y) \lor (x = \text{id}[n] \land y = s))$
  - $\lor \exists n. \text{pending (id}[n], n) \land \forall x. \text{leader’}(x) \leftrightarrow (\text{leader}(x) \lor x = n)$
  - ...

Modeling with EPR

- **State**: finite first-order structure over vocabulary V (+ axioms)

  Propose(n): send(id(n), next(n))
  
  Recv(n,msg): if msg = id(n) then leader(n) := true
  if msg > id(n) then send(msg,next(n))

- **Transition relation**: expressed as EPR formula TR(V, V’), e.g.: 

  \( \exists n, s. \ s = \text{next}(n) \land \forall x, y. \ \text{pending}'(x,y) \leftrightarrow (\text{pending}(x,y) \lor (x = \text{id}[n] \land y = s)) \)

  \( \lor \ \exists n. \ \text{pending} (\text{id}[n], n) \land \forall x. \ \text{leader}'(x) \leftrightarrow (\text{leader}(x) \lor x = n) \)

  ...
Modeling with EPR

- **State**: finite first-order structure over vocabulary $V$ (+ axioms)

- **Initial** states and **safety** property: EPR formulas over $V$
  - $\text{Init}(V)$ – initial states, e.g., $\forall \text{id}, \text{n. } \neg \text{pending}(\text{id}, \text{n})$
  - $\text{Bad}(V)$ – bad states, e.g., $\exists \text{n}_1, \text{n}_2. \text{leader}(\text{n}_1) \land \text{leader}(\text{n}_2) \land \text{n}_1 \neq \text{n}_2$

- **Transition relation**: expressed as EPR formula $\text{TR}(V, V')$, e.g.:
  - $\exists \text{n}, \text{s. } \text{“s = next(n)” } \land \forall \text{x, y. } \text{pending’}(\text{x, y}) \leftrightarrow (\text{pending}(\text{x, y}) \lor (\text{x = id}[\text{n}] \land \text{y = s}))$
  - $\lor \exists \text{n. } \text{pending’}(\text{id}[\text{n}], \text{n}) \land \forall \text{x. } \text{leader’}(\text{x}) \leftrightarrow (\text{leader}(\text{x}) \lor \text{x = n})$
  - ...

Modeling with EPR

- **State**: finite first-order structure over vocabulary $V$ (+ axioms)

- **Initial** states and **safety** property: EPR formulas over $V$
  - $\text{Init}(V)$ – initial states, e.g., $\forall \text{id, n. } \neg \text{pending(id, n)}$
  - $\text{Bad}(V)$ – bad states, e.g., $\exists n_1, n_2. \text{leader}(n_1) \land \text{leader}(n_2) \land n_1 \neq n_2$

Specify and verify the protocol for **any** number of nodes in the ring.
Using EPR for Verification

- System model $\text{Init}(V), \text{Bad}(V), \text{TR}(V, V') \in \text{EPR}$

- Inductive invariant $\text{Inv}(V) \in \forall^*$

- Verification conditions
  
  - Initiation $\text{Init}(V) \implies \text{Inv}(V)$
  
  - Cons. $\text{Inv}(V) \land \text{TR}(V, V') \implies \text{Inv}(V')$
  
  - Safety $\text{Inv}(V) \implies \neg \text{Bad}(V)$

Verification conditions $\in \text{EPR}$

$\Rightarrow$ Decidable to check
Inductive Invariant for Leader Election

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

Inductive? No!

- $\leq (\text{ID}, \text{ID})$ – 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
Inductive Invariant for Leader Election

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

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

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

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

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

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

The reason for using “btw” instead of “next”
Inductive Invariant for Leader Election

Safety property:

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

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

- \( I_1 = \forall n_1, n_2: \text{Node. } \text{leader}(n_1) \rightarrow id[n_2] \leq id[n_1] \)
- \( I_2 = \forall n_1, n_2: \text{Node. } \text{pnd}(id[n_1], n_1) \rightarrow id[n_2] \leq id[n_1] \)
- \( I_3 = \forall n_1, n_2, n_3: \text{Node. } \text{btw}(n_1, n_2, n_3) \land \text{pnd}(id[n_1], n_1) \rightarrow id[n_3] \)

The leader has the highest id
Only highest id can be self-pnd
Cannot bypass higher nodes

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

**Proof**
# Axioms: Leader Election Protocol

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

<table>
<thead>
<tr>
<th>Intention</th>
<th>EPR Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node ID’s</strong></td>
<td>Integers</td>
</tr>
<tr>
<td>$\forall i:\text{ID. } i \leq i$ Reflexive</td>
<td>$\forall x, y: \text{Node. } \text{id}(x) = \text{id}(y) \rightarrow x = y$ Injective</td>
</tr>
<tr>
<td>$\forall i, j, k: \text{ID. } i \leq j \land j \leq k \rightarrow i \leq k$ Transitive</td>
<td></td>
</tr>
<tr>
<td>$\forall i, j: \text{ID. } i \leq j \land j \leq l \rightarrow i = j$ Anti-Symmetric</td>
<td></td>
</tr>
<tr>
<td>$\forall i, j: \text{ID. } i \leq j \lor j \leq i$ Total</td>
<td></td>
</tr>
<tr>
<td>$\forall x, y: \text{Node. } \text{id}(x) = \text{id}(y) \rightarrow x = y$ Injective</td>
<td></td>
</tr>
<tr>
<td><strong>Ring Topology</strong></td>
<td>Next edges + Transitive closure</td>
</tr>
<tr>
<td>$\forall x, y, z: \text{Node. } \text{btw}(x, y, z) \rightarrow \text{btw}(y, z, x)$ Circular shifts</td>
<td></td>
</tr>
<tr>
<td>$\forall x, y, z, w: \text{Node. } \text{btw}(w, x, y) \land \text{btw}(w, y, z) \rightarrow \text{btw}(w, x, z)$ Transitive</td>
<td></td>
</tr>
<tr>
<td>$\forall x, y, w: \text{Node. } \text{btw}(w, x, y) \rightarrow \neg \text{btw}(w, y, x)$ Anti-Symmetric</td>
<td></td>
</tr>
<tr>
<td>$\forall x, y, z, w: \text{Node. } \text{distinct}(x, y, z) \rightarrow \text{btw}(w, x, y) \lor \text{btw}(w, y, x)$</td>
<td></td>
</tr>
<tr>
<td>“$\texttt{next}(a) = b$” $\equiv$ $\forall x: \text{Node. } x \neq a \land x \neq b \rightarrow \text{btw}(a, b, x)$</td>
<td></td>
</tr>
</tbody>
</table>
So far

Formal specification with EPR

• Surprisingly expressive
  • Integers: numeric id’s expressed with $\leq$
  • Transitive closure: ring topology expressed with btw
  • Network semantics: pending messages
• Sets and cardinalities (for consensus protocols) [OOPSLA’17]
• Liveness properties [POPL’18, FMCAD’18]
• Implementations [PLDI’18]

Not in this talk
Invariant inference: finding inductive invariants

(1) Automatically
   – Adapt techniques from finite-state model checking (PDR)

(2) Interactively
   – Based on graphically displayed counterexamples to induction
How can we find a *universally quantified* inductive invariant?
## Inductive Invariant for Leader Election

<table>
<thead>
<tr>
<th>( I_0 )</th>
<th>( \neg \text{Bad} )</th>
<th>( \forall n_1, n_2: \text{Node.} \ \text{leader}(n_1) \land \text{leader}(n_2) \rightarrow n_1 = n_2 )</th>
<th>At most one leader elected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \exists n_1, n_2: \text{Node.} \ \text{leader}(n_1) \land n_1 \neq n_2 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( I_1 )</th>
<th>( \forall n_1, n_2: \text{Node.} \ \text{leader}(n_1) \rightarrow \text{id}[n_2] \leq \text{id}[n_1] )</th>
<th>The leader has the highest id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \exists n_1, n_2: \text{Node.} \ \text{leader}(n_1) \land \text{id}[n_2] &gt; \text{id}[n_1] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( I_2 )</th>
<th>( \forall n_1, n_2: \text{Node.} \ \text{pnd}(\text{id}[n_1], n_1) \rightarrow \text{id}[n_2] \leq \text{id}[n_1] )</th>
<th>Only highest id can be self-pnd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \exists n_1, n_2: \text{Node.} \ \text{pnd}(\text{id}[n_1], n_1) \land \text{id}[n_2] &gt; \text{id}[n_1] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( I_3 )</th>
<th>( \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] )</th>
<th>Cannot bypass higher nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \neg \exists 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) \land \text{id}[n_3] &gt; \text{id}[n_2] )</td>
</tr>
</tbody>
</table>
Construct Inv by excluding “bad” states

1. How to find these states?
2. How to generalize into conjectures?
Use **diagrams** as generalization:

- 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

\[
\text{exclude}(\sigma) = \neg \text{Diag}(\sigma)
\]

Generalization using Diagram :

Generalize even more if $\sigma$ is a partial structure

$\text{Diag}(\sigma) = \exists \ x \ y. \ x \neq y$

$\wedge \leq (x, y) \wedge \neg \leq (y, x)$

$\wedge \leq (x, x) \wedge \leq (y, y)$

$\text{exclude}(\sigma) = \neg \text{Diag}(\sigma)$

∀* Invariant - excluded substructures

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

Clause / conjecture

[Diagram of a cube]

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

Cube
Leader election example

The leader has the highest ID

At most one leader

Only the leader can be self-pending

Cannot bypass higher nodes

How to find the (partial) states to generalize from?
(1) Automatic inference: UPDR

- Based on Bradley’s IC3/PDR [VMCAI11,FMCAD11]
  - SAT-based verification of finite-state systems
- Abstracts concrete states using their logical diagram
- Backward traversal performed over diagrams
- Blocking of CTI excludes a *generalization* of its diagram $\Rightarrow$ generates universally quantified lemmas

---

UPDR: Possible outcomes

- Universal inductive invariant found
  - System is safe

Used to infer inductive invariants / procedure summaries of:
  - Heap-manipulating programs, e.g.
    - Singly/Doubly/Nested linked list
    - Iterators in Java - Concurrent modification error (CME)
  - Distributed protocols
    - Spanning tree
    - Learning switch
    - ...

No need for user-defined predicates/templates!
UPDR: Possible outcomes

- Universal inductive invariant found
  - System is safe

- Proof that no universal inductive invariant exists
  - Safety not determined*

* can use Bounded Model Checking to find real counterexamples
UPDR: Possible outcomes

- **Universal inductive invariant found**
  - System is safe

- **Proof that no universal inductive invariant exists**
  - Safety not determined*

- **Divergence**
  - In general, inferring universal ind. inv. is undecidable
  - For linked lists it is decidable, UPDR will also terminate
    - Proof uses well-quasi-order and Kruskal’s tree theorem

---

Automatic Inference (e.g., UPDR)

Ultimately limited by undecidability
(2) Interactive Inference

- Let the user guide the tool
  - User has intuition about the essence of the proof
  - Computer is good at handling corner cases

Supervised Verification of Infinite-State Systems
Interactive Inference

What is the human’s role?
What is the machine’s role?
How do they interact?

Deductive verification

Ultimately limited by human

proof/code:
Verdi: ~10
IronFleet: ~4

Ultimately limited by undecidability

Model Checking
Static Analysis

Expressiveness

Supervised Verification of Infinite-State Systems
Ivy: Interactive Generalization

\[ \text{Inv} = I_0 \land \cdots \land I_k \]

- Displays “minimal” CTI to exclude
- Generalizes to a partial state
  - Removes “irrelevant” facts (graphical interface - checkboxes)
- Translates to universally quantified conjecture (via diagram)
- Provides auxiliary automated checks:
  1. BMC(K): uses SAT solver to check if conjecture is true up to K
     - User determines the right K to use
  2. ITP(K): uses SAT solver to discover more facts to remove
- Examines the proposed conjecture – it could be wrong
  - Adds \( I_{k+1} \)

https://github.com/Microsoft/ivy
Interactive 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
Summary 1

Verification with decidable logic

• EPR - decidable fragment of FOL
  • Deduction is decidable
  • Finite counterexamples

• Can be made surprisingly powerful
  • Transitive closure: linked lists, ring topology [PLDI’16]
  • Paxos, Multi-Paxos, [OOPSLA’17]
  • Liveness and Temporal Properties [POPL’18]
  • Developing verified implementations [PLDI’18]
Invariant Inference

• Automatic inference: UPDR [CAV’15, JACM]
• Interactive inference: Ivy [PLDI’16]

• Use logical diagram to infer Inv ∈ ∀*
• Can also prove absence of Inv ∈ ∀*
Take away

• Decidable logic is useful
  - facilitates automation

• We need ways to guide verification tools

• How to divide the problem between human and machine?
• Different inference schemes
• Different Forms of interaction
• Other logics
• Theoretical understanding of limitations and tradeoffs

Supervised Verification of Infinite-State Systems