Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv
Contributors
Marcelo Taube, Giuliano Losa, Kenneth McMillan, Oded Padon, Sharon Shoham

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

James R. Wilcox, Doug Woos
And Also

Anindya Benerjee    Yotam Feldman    Neil Immerman    Aurojit Panda

Shachar Itzhaky    Aleks Nanevsky    Orr Tamir    Robbert van Renesse
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
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
What about correctness of the low level implementation?
Automatic verification of infinite-state systems

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

Counterexample
Unknown / Diverge
Proof

System $S$

Property $\varphi$

Rice’s Theorem
I can’t decide!

“Formal methods are the future of computer science. Always have been, always will be.” William E. Aitken
Deductive verification

Counterexample to Induction

System $S$

Inductive argument $Inv$

Property $\varphi$

Deductive Verification

1) Is $Inv$ an inductive invariant for $S$?
2) Does $Inv$ entail $\varphi$?

Unknown / Diverge

Proof

Unknown / Diverge

Unknown / Diverge
Inductive invariants

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

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**)

**Inductive invariants**

**System State Space**

**Safety Property**

**Bad**

**Inv**

**Reach**

**Init**
Logic-based deductive verification

• Represent $Init$, $\rightarrow$, $Bad$, $Inv$ by logical formulas
  • Formula $\iff$ Set of states

• Automated solvers for logical satisfiability made huge progress
  • Propositional logic (SAT) – industrial impact for hardware verification
  • First-order theorem provers
  • Satisfiability modulo theories (SMT) – major trend in software verification
Deductive verification by reductions to First Order Logic

- Protocol: \( \text{Init}(V), \text{Tr}(V, V') \)
- Loop Invariant: \( \text{Inv}(V) \)
- Safety Property: \( \neg \text{Bad}(V) \)

Front-End

1) \( \text{SAT}(\text{Init}(V) \land \neg \text{Inv}(V))? \)
2) \( \text{SAT}(\text{Inv}(V) \land \text{Tr}(V, V') \land \neg \text{Inv}(V'))? \)
3) \( \text{SAT}(\text{Inv}(X) \land \text{Bad}(V))? \)

First Order SAT Solver

Y: Counterexample to Induction (CTI)
N: Proof
Challenges in deductive verification

• Formal specification
  • Modeling the system and property in a logical formalism

• Checking inductiveness
  • Undecidability of satisfiability checking (unbounded state, arithmetic)

• Inference: finding inductive invariants [PLDI’16, POPL’16, JACM’17]

[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

[JACM’17] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, Sharon Shoham:
Property-Directed Inference of Universal Invariants or Proving Their Absence
Proving distributed systems is hard

Verdi
Verification of Raft in Coq
50,000 lines of manual proof

IronFleet
Verification of Multi-Paxos
12,000 lines and 3.7 person-years
Uses solver for undecidable SMT checks

SAT Modulo Theory (SMT)

• Extend first order logic with theories
  • Linear arithmetic \( \exists X:Z. 3X + 2 = 0 \)
  • Bitvectors
  • Theory of arrays
  • …
• Hides complexity from the user
  • Works in many cases
• Great tools: Yices, Z3, CVC, Boolector, …
• Essential in Dafny, Sage, Klee, Rossete, F*, …
• But unpredictable!
  • Can fail on tiny inputs
  • Tuning requires knowledge in the heuristics
  • The butterfly effect
Ivy’s 1st Principle: First Order Abstraction

- Abstracts states as finite (uninterpreted) first order structures
  - Unbounded relations
  - No other data structures
  - Abstract integers, sets, cardinalities, ...
- Arbitrary loops and procedures
- Express program meaning as first order transition systems:
  - \( r(X, Y) := \exists Z. \ p(X, Z) \land q(Z, Y) \equiv \forall X, Y. \ r'(X, Y) \iff \exists Z. \ p(X, Z) \land q(Z, Y) \)
- “A step towards decidability”
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?

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
- Undecidable to check inductiveness
  - Unbounded nodes, messages
  - Arithmetic
  - Transitive closure

Modeling in first-order logic

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

Axiomatized in first-order logic

\[
\begin{align*}
\text{id}_1 & \leq \text{id}_3 < \text{id}_5 \leq \text{id}_4 \leq \text{id}_2 \leq \text{id}_6 \\
<n_5, n_1, n_3> & \in I(\text{btw})
\end{align*}
\]
Modeling in first-order logic

State: finite first-order structure over vocabulary V:

- $\leq (\text{ID}, \text{ID})$ – 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

Specify and verify the protocol for any number of nodes in the ring
Modeling in first-order logic

- **State**: finite first-order structure over vocabulary \( V (+\text{ axioms}) \)

- **Initial states and safety** property expressed as formulas:
  - \( \text{Init}(V) \) – initial states, e.g., \( \forall x,y. \neg \text{pending}(x,y) \)
  - \( \text{Bad}(V) \) – bad states, e.g., \( \exists n_1, n_2. \text{leader}(n_1) \wedge \text{leader}(n_2) \wedge n_1 \neq n_2 \)

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

- EPR Protocol: $\text{Init}(V), \text{Tr}(V, V')$
- EPR Loop Invariant: $\text{Inv}(X)$
- EPR Safety Property: $\neg \text{Bad}(X)$

Front-End:

1) $\text{SAT}((\text{Init}(V) \land \neg \text{Inv}(V))$?
2) $\text{SAT}((\text{Inv}(V) \land \text{Tr}(V, V') \land \neg \text{Inv}(V'))$?
3) $\text{SAT}((\text{Inv}(X) \land \text{Bad}(V))$?

EPR Solver:

- Y: Counterexample to Induction (CTI)
- N: Proof
Leader election protocol – inductive invariant

Inductive invariant: $Inv = I_0 \land I_1 \land I_2$

$I_0 = \forall n_1, n_2: \text{Node. } leader(n_1) \land leader(n_2) \rightarrow n_1 = n_2$

Unique leader

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

The leader has the highest ID

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

Only the leader can be self-pending

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

EPR Solver

Init($V$) $\land \neg Inv(V)$

$\neg Inv(V') \land Inv(V) \land \neg Inv(V)$

I can decide EPR!
Ivy: check inductiveness

Leader Protocol

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

Check Inductiveness

CTI

EPR

\[ I_0 \land I_1 \land I_2 \]

rcv(1, id(2))
Leader election protocol – inductive invariant

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

- **\( I_0 = \forall n_1, n_2: \text{Node. } leader(n_1) \land leader(n_2) \rightarrow n_1 = n_2 \)**
  - Unique leader
- **\( I_1 = \forall n_1, n_2: \text{Node. } leader(n_2) \rightarrow id[n_1] \leq id[n_2] \)**
  - The leader has the highest ID
- **\( I_2 = \forall n_1, n_2: \text{Node. } pending(id[n_2], n_2) \rightarrow id[n_1] \leq id[n_2] \)**
  - Only the leader can be self-pending
- **\( I_3 = \forall n_1, n_2, n_3: \text{Node. } btw(n_1, n_2, n_3) \land pending(id[n_2], n_1) \rightarrow id[n_3] \leq id[n_2] \)**
  - Cannot bypass higher nodes

**EPR Solver**

- **\( \preceq (ID, ID) \rightarrow \text{total order on node id's} \)**
  - \( Init(V) \land \neg Inv(V) \)
- **\( btw(Node, Node) \rightarrow \text{the ring topology} \)**
- **\( id: Node \rightarrow ID \rightarrow \text{relate a node to its unique id} \)**
- **\( pending(ID, Node) \rightarrow \text{pending messages} \)**
- **\( leader(Node) \rightarrow \text{leader(n) means n is the leader} \)**

**Proof**

I can decide EPR!
Skolemization

• Procedure that transforms a first order formula $\varphi$ over vocabulary $V=<S, C, R, F>$ into a universal formula $Sk(\varphi)$ over vocabulary $V’=<S, C \cup C’, R, F \cup F’>$
  • $\varphi$ is satisfiable $\iff Sk(\varphi)$ is satisfiable

• Example
  • $\forall X: S1. \exists y:S2. r(X, Y) \land q(Y)$
    
    $= \text{SAT}$
    $\forall X: S1. r(X, f(X)) \land q(f(X))$
Why is SMT undecidable?

• Theories
  • $2 \times X^4 + 5 \times X^2 - 3 \times X + 2 = 0$

• Quantifier-alternation and function symbols (cycles)
  • $\forall x: N. \exists y: N. x < y$
  
  • $\forall x: N. \ x < f(x)$

  • $\forall x: A. \exists y: B. \ Q(x, y) \land \forall z: B. \ \exists w: A. \ P(z, w)$

Also happens without theories

  • $\forall x: A. \ Q(x, h(x)) \land \forall z: B. \ P(z, g(z))$
    
    $h: A \rightarrow B$ and $g: B \rightarrow A$
Infinite Structures

• \( \forall x. \text{le}(x, x) \) Reflexive
• \( \forall x, y, z. \text{le}(x, y) \land \text{le}(y, x) \Rightarrow \text{le}(x, z) \) Transitive
• \( \forall x, y. \text{le}(x, y) \land \text{le}(y, x) \Rightarrow x=y \) Antisymmetric
• \( \forall x, y. \text{le}(x, y) \lor \text{le}(y, x) \) Total
• \( \forall x. \text{le}(\text{zero}, x) \) Non-empty
• \( \forall x. \exists y. \text{le}(x, y) \land x \neq y \) Successor

For finite models validity is co-R.E.
Effectively Propositional Logic – EPR
a.k.a. Bernays-Schönfinkel-Ramsey class

• Limited fragment of first-order logic
  • No function symbols
  • No theories
  • Restricted quantifier prefix: \( \exists^* \forall^* \phi_{Q.F.} \)
    • No \( \forall^* \exists^* \)
EPR Sat

\[ \exists x, y. \forall z. r(x, z) \iff r(z, y) \]

\[ \models_{\text{sat}} \forall z . \ r(c_1, z) \iff r(z, c_2) \]

\[ \models_{\text{sat}} (r(c_1, c_1) \iff r(c_1, c_2)) \land (r(c_1, c_2) \iff r(c_2, c_2)) \]

\[ \models_{\text{sat}} (P_{11} \iff P_{12}) \land (P_{12} \iff P_{22}) \]
SAT becomes undecidable

- $\forall x. \text{le}(x, x)$  
  Reflexive

- $\forall x, y, z. \text{le}(x, y) \land \text{le}(y, z) \Rightarrow \text{le}(x, z)$  
  Transitive

- $\forall x, y. \text{le}(x, y) \land \text{le}(y, x) \Rightarrow x = y$  
  Antisymmetric

- $\forall x, y. \text{le}(x, y) \lor \text{le}(y, x)$  
  Total

- $\forall x. \text{le}($zero$, x)$  
  Non-empty

- $\forall x. \exists y. \text{le}(x, y) \land x \neq y$  
  Successor
Effectively Propositional Logic – EPR
a.k.a. Bernays-Schönfinkel-Ramsey class

• Limited fragment of first-order logic w/o theories
  • No function symbols
  • Restricted quantifier prefix: $\exists^* \forall^* \phi_{Q.F.}$
    • No $\forall^* \exists^*$

• Small model property
  • A formula is satisfiable iff it is holds on models of size (number of constant symbols + existential variables)
Decidable Fragments in Ivy

• EPR
• EPR++ allow **acyclic** function and quantifier alternations
  • E.g., f:A→B, so cannot have g:B→A
  • Maintains small model property of EPR
  • Finite complete instantiations

• QFLIA – Quantifier Free Linear Integer Arithmetic
• FAU – Finite Almost Uninterpreted [CAV’07]
  • Allow limited arithmetic + acyclic quantifier alternations
  • Maintains finite complete instantiations

[CAV’07] Ge & de Moura: *Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories*
EPR++ based verification

Predictability
• Decidable inductiveness check
• Finite counterexamples
  • Can be minimized
• Easy to display graphically
• Arbitrary first order updates
• No more butterfly effect

Challenges
• Expressiveness of first order logic
  • Paths
  • Sets & Cardinalities
• Quantifier alternation cycles
• Not closed under conjunction and negation
• Gap to low level implementation
First-order axiomatization of ring paths

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

- Cannot express in first-order from “next” relation!
- Key enabler: use \text{btw} and not \text{next}

relation \text{btw} (Node, Node, Node)
axiom \forall x, y, z: \text{Node. } \text{btw}(x, y, z) \rightarrow \text{btw}(y, z, x) \text{ circular}
axiom \forall x, y, z, w: \text{Node. } \text{btw}(w, x, y) \land \text{btw}(w, y, z) \rightarrow \text{btw}(w, x, z) \text{ transitive}
axiom \forall x, y, w: \text{Node. } \text{btw}(w, x, y) \rightarrow \neg \text{btw}(w, y, x) \text{ anti-symmetric}
axiom \forall x, y, w: \text{Node. } \neq(w, x, y) \rightarrow \text{btw}(w, x, y) \lor \text{btw}(w, y, x) \text{ total}
macro “next(a)=b” \equiv \forall x: \text{Node. } x=a \lor x=b \lor \text{btw}(a,b,x) \text{ edges}
Key idea: representing deterministic paths

Alternative 1: maintain $s$
- $\leq$ defined by transitive closure of $s$
- not definable in first-order logic

Alternative 2: maintain $\leq$
- $s$ defined by transitive reduction of $\leq$
- Unique due to out degree 1
- Definable in first order logic

\[s(x)=y \equiv x < y \land \forall z. x < z \rightarrow y \leq z\]
\[x < y \equiv x \leq y \land x \neq y\]
For every class $C$ of finite graphs above:

- Axioms for path relation – universally quantified
- Successor formula – 1 universal quantifier
- Update formulas for node/edge addition and removal – universally quantified

**Soundness Theorem:** Every graph of class $C$ satisfies the axioms of $C$

**Edges agree with successor formula**

**Completeness Theorem:** Every finite structure satisfying the axioms of $C$ is isomorphic (paths and edges) to a graph of class $C$
For every class C of finite graphs above:

- **Axioms for path relation** – universally quantified
- **Successor formula** – 1 universal quantifier
- **Update formulas for node / edge addition and removal** – universally quantified

- **Soundness Theorem**: Every graph of class C satisfies the axioms of C.
  Edges agree with successor formula

- **Completeness Theorem**: Every finite structure satisfying the axioms of C is isomorphic (paths and edges) to a graph of class C.
Parameterized toy leader election

- $N$ processes choose a leader
  - Process may request vote by broadcast
  - Processes vote for a requester
  - Process with majority of votes is leader

Prove: at most one leader
First-order expressiveness issues

• To prove the toy protocol, we need an inductive invariant
• Problem: cardinality reasoning

\[
\text{if } |\text{votes}(p)| > \frac{|\text{all}|}{2} \text{ then send leader}(p)
\]

cardinality + arithmetic + uninterpreted + quantifiers = second order & undecidable!

• Solution: axiomatize cardinalities in first-order logic

\[
\forall s, t. \text{majority}(s) \land \text{majority}(t) \rightarrow \exists p. \text{member}(p, s) \land \text{member}(p, t)
\]
An ADT for pid sets

datatype set(pid) = {
    relation member (pid, set)
    relation majority(set)
    procedure empty returns (s:set)
    procedure add(s:set,e:pid) returns (r:set)
}

specification {
    procedure empty ensures $\forall p. \neg \text{member}(p, s)$
    procedure add ensures $\forall p. \text{member}(p, r) \leftrightarrow (\text{member}(p, s) \lor p = e)$
    property [maj] $\forall s, t. \text{majority}(s) \land \text{majority}(t) \rightarrow \exists p. \text{member}(p, s) \land \text{member}(p, t)$
}

We have hidden the cardinality and arithmetic

The key is to recognize that the protocol only needs property maj
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
  • Amazon AWS
  • VMware NSX
  • Many more...
Inductive invariant of Paxos

# safety property
\textbf{invariant} \texttt{decision(N1,R1,V1) \& decision(N2,R2,V2) -> V1 = V2}

# proposals are unique per round
\textbf{invariant} \texttt{proposal(R,V1) \& proposal(R,V2) -> V1 = V2}

# only vote for proposed values
\textbf{invariant} \texttt{vote(N,R,V) -> proposal(R,V)}

# decisions come from quorums of votes:
\textbf{invariant} \texttt{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
\textbf{invariant} \texttt{one\_b\_max\_vote(N,R,\texttt{\texttt{2,none,V1}}) \& \simle(R2,R1) -> \simvote(N,R1,V2)}
\textbf{invariant} \texttt{one\_b\_max\_vote(N,R,RM,V) \& RM =\:\texttt{\texttt{none}} -> \simle(R,RM) \& \texttt{vote(N,RM,V)}}
\textbf{invariant} \texttt{one\_b\_max\_vote(N,R,RM,V) \& RM =\:\texttt{\texttt{none}} \& \simle(R,RO) \& \simle(RO,RM) -> \simvote(N,RO,VO)}

# property of choosable and proposal
\textbf{invariant} \texttt{\simle(R2,R1) \& proposal(R2,V2) \& V1 = V2 -> exists N. member(N,Q) \& left\_rnd(N,R1) \& \simvote(N,R1,V1)}

# property of one\_b, left\_rnd
\textbf{invariant} \texttt{one\_b(N,R2) \& \simle(R2,R1) -> left\_rnd(N,R1)}
### Paxos made EPR: Proof size and verification time

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model [LOC]</th>
<th>Invariants</th>
<th>Verification time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>85</td>
<td>11</td>
<td>2.2</td>
</tr>
<tr>
<td>Multi-Paxos</td>
<td>98</td>
<td>12</td>
<td>2.6</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*</td>
<td>132</td>
<td>16</td>
<td>5.4</td>
</tr>
</tbody>
</table>

*first mechanized verification

Abstraction and 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.
## Protocol Verification

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model [LOC]</th>
<th>Invariants</th>
<th>Verification time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stoppable Paxos*</td>
<td>132</td>
<td>16</td>
<td>5.4</td>
</tr>
</tbody>
</table>

---

(1.7. NoneChoosableAfter(i, b, v')

PROOF: We assume v ∈ 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) = v.

PROOF: Assumption (1.1.3) implies E4(i, b, Q, v), so the assumption sval2a(i, b, Q) = v implies E4(i, b, Q, v), and the case assumption.

(3.2. Done2a(i, mbal2a(i, b, Q), v)

PROOF: (3.1) The case assumption and the definition of sval2a imply val2a(i, b, Q) = v for some acceptor a in Q, which by Lemma 1.3 implies Done2a(i, mbal2a(i, b, Q), v).

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

(3.3. Case: c < mbal2a(i, b, Q)

PROOF: By the case assumption and the assumptions v ∈ StopCmd and j > i, this implies NotChoosable(j, c, w).

(3.4. Case: mbal2a(i, b, Q) ≤ c < b

(4.1. mbal2a(j, b, Q) < mbal2a(i, b, Q)

PROOF: The assumption v ∈ StopCmd and (3.1) imply sval2a(i, b, Q) ∈ StopCmd. Case assumption (2.2) and the definition of sval2a then imply mbal2a(k, b, Q) < mbal2a(i, b, Q) for all k > i.

(4.2. NotChoosable(j, c, w)

PROOF: (4.1) and case assumption (3.4) imply mbal2a(j, b, Q) < c < b. By assumption (1.1.4), Lemma 3 implies NotChoosable(j, c, w).
Impact First Order Abstraction

First-Order Logic approach now used at Ethereum Dev UG
From ~1500 LOC to ~150 LOC (Isabelle/HOL proof)
Closing the gap

• Reasoning about abstract protocols (designs)
  • User provides axioms expressed in first order logic
  • Not checked by the system
  • Missing axioms can lead to false alarms

• Reasoning about implementations
  • Abstract total order $\rightarrow$ concreter domain, e.g., integers
  • Abstract sets with majorities $\rightarrow$ some data structure, e.g., arrays

• How can we verify that the user defined “axioms” are satisfied by the low-level implementation?
  • Solution: Modularity – wrap implementations in ADT’s
  • Each module may use a different decidable theory
Ivy 2\textsuperscript{rd} Principle: Scope Verification Conditions

• The user is responsible for breaking quantifier alternation cycles
  • Also in designs

• Leverage modularity (natural for distributed protocols)
  • Prove abstract protocol and use it as a lemma to prove concrete implementation
  • Sometimes functions are abstracted as relations
    • Allow more behaviors
  • Extract executable from the concrete implementation

• Axioms of the design must be fulfilled by the implementation
  • Theories are adds-on
Modularity

Original system

Original inductive argument

Original property

[Diagram with puzzle pieces]
Separate Verification of each module

- subsystem
- Partial argument
- Property

Verification tool

- Incorrect
  - Finds bug
- Correct
  - Finds proof

NO UNDECIDABILITY

😊
An ADT for pid sets

datatype set(pid) = {
    relation member (pid, set)
    relation majority(set)
    procedure empty returns (s:set)
    procedure add(s:set,e:pid) returns (r:set)
}

specification {
    procedure empty ensures ∀p. ¬member(p, s)
    procedure add ensures ∀p. member(p, r) ↔ (member(p, s) ∨ p = e)
    property [maj] ∀s, t. majority(s) ∧ majority(t) → ∃p. member(p, s) ∧ member(p, t)
}

We have hidden the cardinality and arithmetic

The key is to recognize that the protocol only needs property maj
Implementation of the set ADT

• Standard approach
  • Implement operations sets using array representation
    \[ \text{member}(p, s) \equiv \exists i. \text{repr}(s)[i] = p \]
  • Define cardinality of sets as a recursive function \(||: \text{set} \rightarrow \text{int}|
    • \text{majority}(s) \equiv |s| + |s| > |\text{all}|$
  • Prove lemma by induction on \(|\text{all}|$

\[ \forall s, t. |s| + |t| > |\text{all}| \rightarrow \exists p. \text{member}(p, s) \land \text{member}(p, t) \]

• The lemma implies property \text{maj}
• All the verification conditions are in EPR+++limited arithmetic (FAU)
Quantifier alternation cycles

• Protocol state
  voters: pid \rightarrow set

• Property maj
  \forall s, t: set. \exists p: pid. \text{majority}(s) \land \text{majority}(t) \Rightarrow \text{member}(p, s) \land \text{member}(p, t)

• Solution: Harness modularity
  • Create an abstract protocol model that doesn’t use voters
  • Prove an invariant using maj, then use this as a lemma to prove the concrete protocol implementation
Abstract protocol model

relation voted(pid, pid)
relation isleader(pid)
var quorum: set

procedure vote(v : pid, n : pid) = {
    require ∀ m. ¬voted(v, m);
    voted(v,n) := true;
}

procedure make_leader(n : pid, s : set) = {
    require majority(s);
    require ∀m. member(m, s) → voted(m, n);
    isleader(n) := true;
    quorum := s;
}

Invariant:

• one leader: ∀n, m. isleader(n) ∧ isleader(m) → n = m
• voted is a partial function: ∀p,n,m. voted(p,n) ∧ voted(p,m) → n = m
• leader has a quorum: ∀n, m. isleader(n) ∧ member(m, quorum) → voted(m, n)

Provable in EPR++
Implementation

- Uses real network vote messages
- Decorated with ghost calls to abstract model
- Uses abstract mode invariant in proof

```
relation already_voted(pid)
handle req(p:pid, n:pid) {
    if ¬already_voted(p) {
        already_voted(p) := true;
        send vote(p,n);
        ghost abs.vote(p,n);  \ call to abstract model must satisfy precondition
    }
}
```

In place of property $\text{maj}$, we use the $\textbf{one leader}$ invariant of the abstract model

\[
\forall p, n. \text{abs. voted}(p, n) \rightarrow \text{already_voted}(p)
\]

\[
\forall p, n. \text{network.vote}(p, n) \leftrightarrow \text{abs. voted}(p, n)
\]

\[
\forall n. \text{leader}(n) \leftrightarrow \text{abs. isleader}(n)
\]

...
Proof using Ivy/Z3

• For each module, we provide suitable inductive invariants
  • Reduces the verification to EPR++ verification conditions
    • the sub verification problems
• Each module’s VC’s in decidable fragment
  • Support from Z3
  • If not, Ivy gives us an explanation, for example a function cycle
• Z3 can quickly and reliably prove all the VC’s
## Proof Length

<table>
<thead>
<tr>
<th>Protocol</th>
<th>System/Project</th>
<th>LOC</th>
<th># manual proof</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAFT</td>
<td>Coq/Verdi</td>
<td>530</td>
<td>50,000</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Ivy</td>
<td>560</td>
<td>200</td>
<td>0.36</td>
</tr>
<tr>
<td>MULTIPAXOS</td>
<td>Dafny/IronFleet</td>
<td>3000</td>
<td>12,000</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Ivy</td>
<td>330</td>
<td>266</td>
<td>0.8</td>
</tr>
</tbody>
</table>
## Verification Effort

<table>
<thead>
<tr>
<th>Protocol</th>
<th>System/Project</th>
<th>Human Effort</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAFT</td>
<td>Coq/Verdi</td>
<td>3.7 years</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Ivy</td>
<td>3 months (from ground up)</td>
<td>Few min</td>
</tr>
<tr>
<td>MULTIPAXOS</td>
<td>Dafny/IronFleet</td>
<td>Several years</td>
<td>6 hr in cloud</td>
</tr>
<tr>
<td></td>
<td>Ivy</td>
<td>1 month (pre-verified model)</td>
<td>few minutes on laptop</td>
</tr>
</tbody>
</table>
Why do people hate First Order Logic?

<table>
<thead>
<tr>
<th>Rants</th>
<th>Ivy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard to understand and error prone</td>
<td>Finite model property</td>
</tr>
<tr>
<td></td>
<td>Display models graphically</td>
</tr>
<tr>
<td>Too weak: Cannot express</td>
<td>First order interface</td>
</tr>
<tr>
<td>Parity</td>
<td>Total orders</td>
</tr>
<tr>
<td>Numeric</td>
<td>Paths in deterministic graphs</td>
</tr>
<tr>
<td>Quorums</td>
<td>Majorities</td>
</tr>
<tr>
<td>Finiteness</td>
<td>Theories as adds-on</td>
</tr>
<tr>
<td>Paths in a graph</td>
<td>First order imperative updates</td>
</tr>
<tr>
<td>Hard for automation</td>
<td>Restrict to EPR++/FAU</td>
</tr>
<tr>
<td>Satisfiability is undecidable</td>
<td>Satisfiability is NEXPTIME complete/$\Sigma_2$</td>
</tr>
<tr>
<td>NP-complete for fixed size</td>
<td>Support from Yices, Z3, Iprover, Vampire</td>
</tr>
</tbody>
</table>
## Languages and Inductiveness

<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++/FAU)</td>
</tr>
</tbody>
</table>
State of the art in formal verification

Proof Assistants

- Verdi: ~10
- IronFleet: ~4

Decidable Models
- Model Checking
- Static Analysis

Ivy

- Decidable deduction
- Finite counterexamples
- proof/code: ~0.2

Ultimately limited by human

Ultimately limited by undecidability
Backup Slides
Inductive Invariant checking vs. inference

Invariant Checking
Is \( Inv \) an inductive invariant for \( \langle Init, TR, \neg Bad \rangle \)?

- Counterexample
- Unknown
- Proof

Invariant Inference
Is there \( Inv \in L \) inductive invariant for \( \langle Init, TR, Bad \rangle \)?

- No
- Unknown
- \( Inv + Proof \)
Relaxed error traces

• **Notation**: $\sigma \sqsubseteq \sigma'$ iff $\sigma$ is isomorphic to a substructure of $\sigma'$
• $\sigma \sqsubseteq \sigma'$ implies $\sigma$ satisfies more universal sentences than $\sigma'$
  • $\sigma \sqsubseteq \sigma'$, $\psi \in \forall^*$, $\sigma' \models \psi$ $\Rightarrow$ $\sigma \models \psi$
• **Relaxed error trace**: $\sigma_1, \sigma_2, ..., \sigma_N$ s.t.
  $\sigma_1 \models \text{Init}$ $\quad$ $\sigma_N \models \text{Bad}$ $\quad$ $\sigma_i, \sigma_{i+1} \models \text{TR}$ or $\sigma_{i+1} \sqsubseteq \sigma_i$

If there is a universal inductive invariant Inv $\in \forall^*$, then a relaxed error trace cannot exist

$\Rightarrow$ A relaxed error trace implies no universal inductive invariant exists
Key Idea: reduction to safety

Finite State

Liveness $\iff$ No Lasso
Key Idea: reduction to safety

Finite State

Liveness ⇔ No Lasso

Infinite State

Liveness ⇔ No Lasso
Problem: Spurious Lasso
Key Idea: reduction to safety

Finite State
Liveness ⇔ No Lasso

Infinite State
Liveness ⇔ No Lasso
Problem: Spurious Lasso

Dynamic Abstraction [POPL’18]
Defined using First-Order Logic