



Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv





Contributors

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





















James R. Wilcox,







Doug Woos

UNIVERSITY of WASHINGTON

And Also

Anindya Benerjee Yotam Feldman Neil Immerman

















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 lvy: Safety Verification by Interactive Generalization

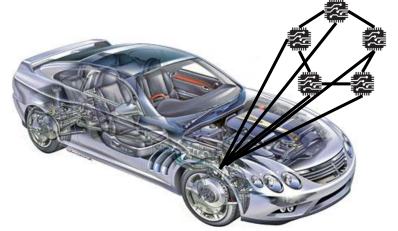
[POPL'16] Oded Padon, Neil Immerman, Aleksandr Karbyshev, Sharon Shoham, MS Decidability of Inferring Inductive Invariants

[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?

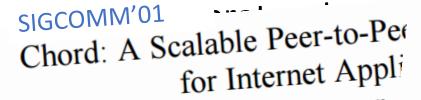
- Distributed systems are everywhere
 - Safety-critical systems
 - Cloud infrastructure
 - Blockchain
- Distributed systems are notoriously hard to get right
 - Even small protocols can be tricky
 - Bugs occur on rare scenarios
 - Testing is costly and not sufficient





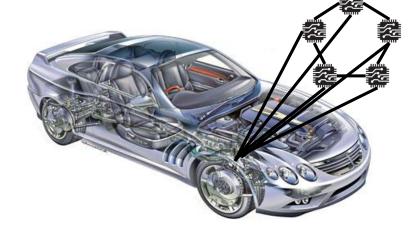
Why verify distributed protocols?

- Distributed systems are everywhere
 - Safety-critical systems
 - Cloud infrastructure
 - Blockchain
- Distributed systems are notoriously hard to get right



Ion Stoica, Robert Morris, David Liben-Nowell, David R. Kar

concurrent node arrivals and departure



Using Lightweight Modeling To Understand Chord

Pamela Zave AT&T Laboratories—Research Florham Park, New Jersey USA

pamela@research.att.com Attractive features of Chord include i the [SIGCOMM] version of the protocol is not correct, and actually invariant in [BOD and actually invariant in [BOD]] Attractive features of Chord papers, and provable performance actually invariantly true of it. The [PODC] is

actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. The

SOSP'07

Best Paper Award

Zyzzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong

Zvzzyva is a state machine replication protocol based on votocols: (1) agreement, (2) view change, and (3) ement protocol orders requests for exechange protocol coordinates CACM'08

ACM Transactions on Computer Systems '09

Zyzzyva: Speculative Byzantine

Fault Tolerance

LORENZO ALVISI, MIKE DAHLIN, ALLEN CLEMENT, and EDMUND WONG RAMAKRISHNA KOTLA Microsoft Research, Silicon Valley

The University of Texas at Austin

arXiv:1712.01367v1 [cs.DC] 4 Dec 2017

Revisiting Fast Practical Byzantine Fault Tolerance

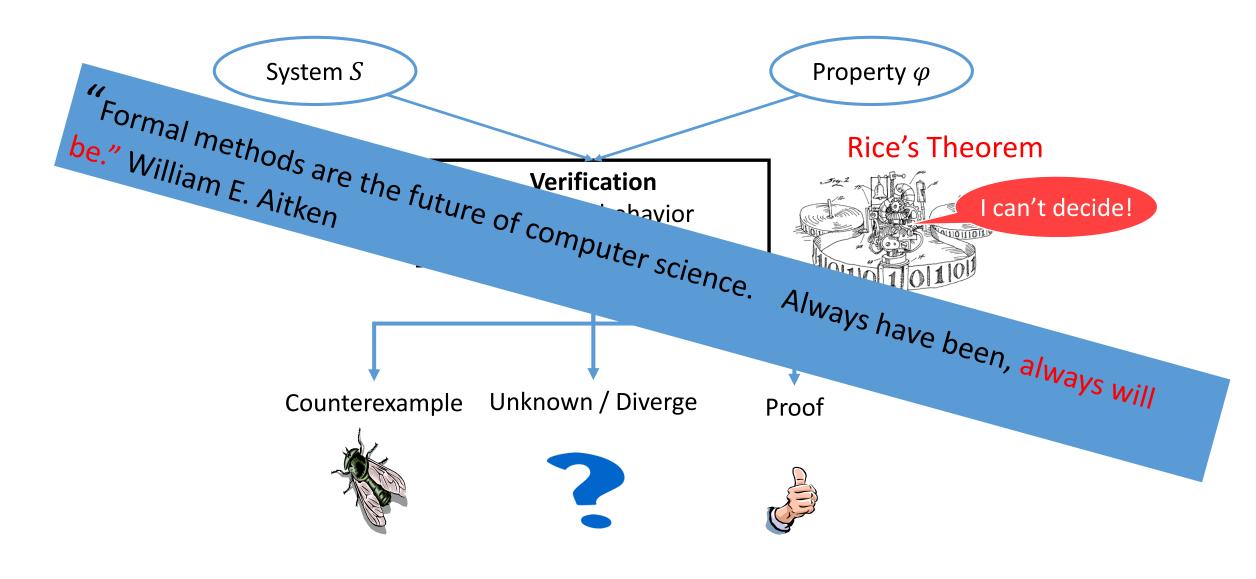
Ittai Abraham, Guy Gueta, Dahlia Malkhi VMware Research

> with: Lorenzo Alvisi (Cornell), Rama Kotla (Amazon), Jean-Philippe Martin (Verily)

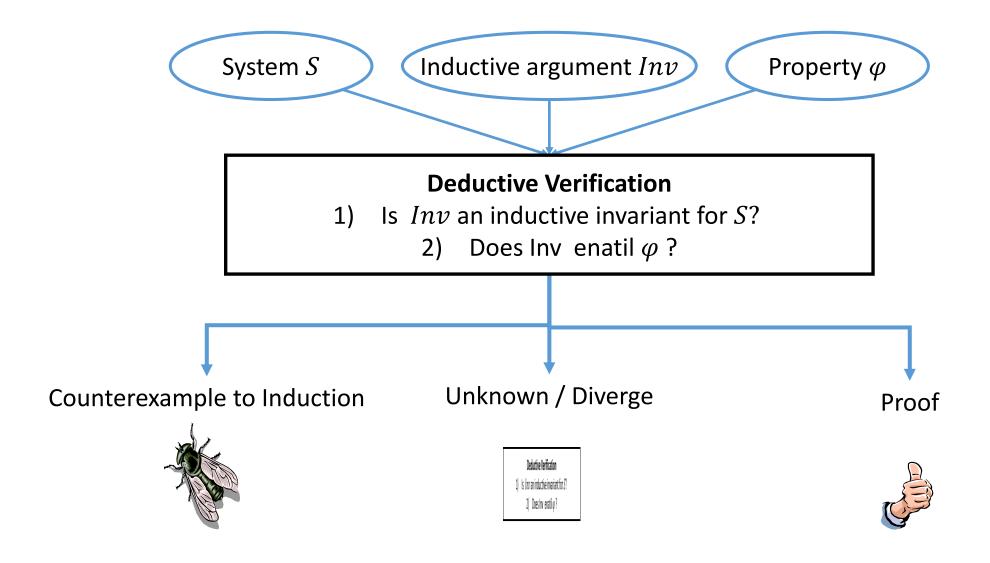
We now proceed to demonstrate that the view-change mechanism in Zyzzyva does not guarantee safety.

What about correctness of the low level implementation?

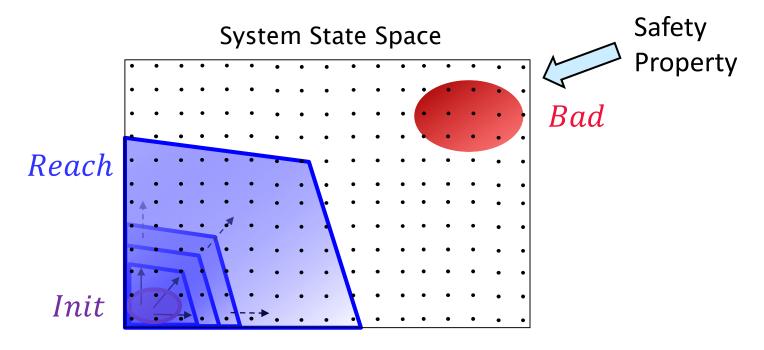
Automatic verification of infinite-state systems



Deductive verification

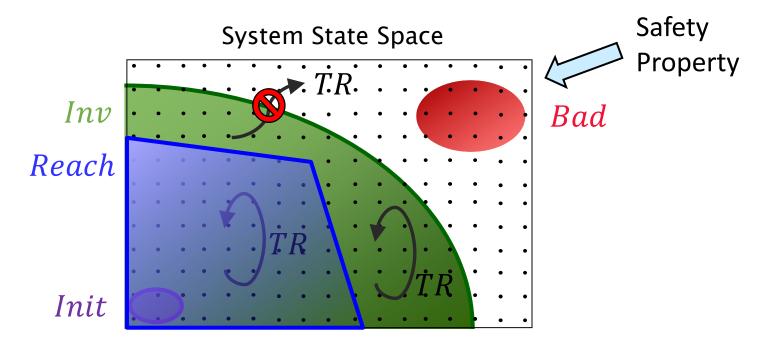


Inductive invariants



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

Inductive invariants



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:

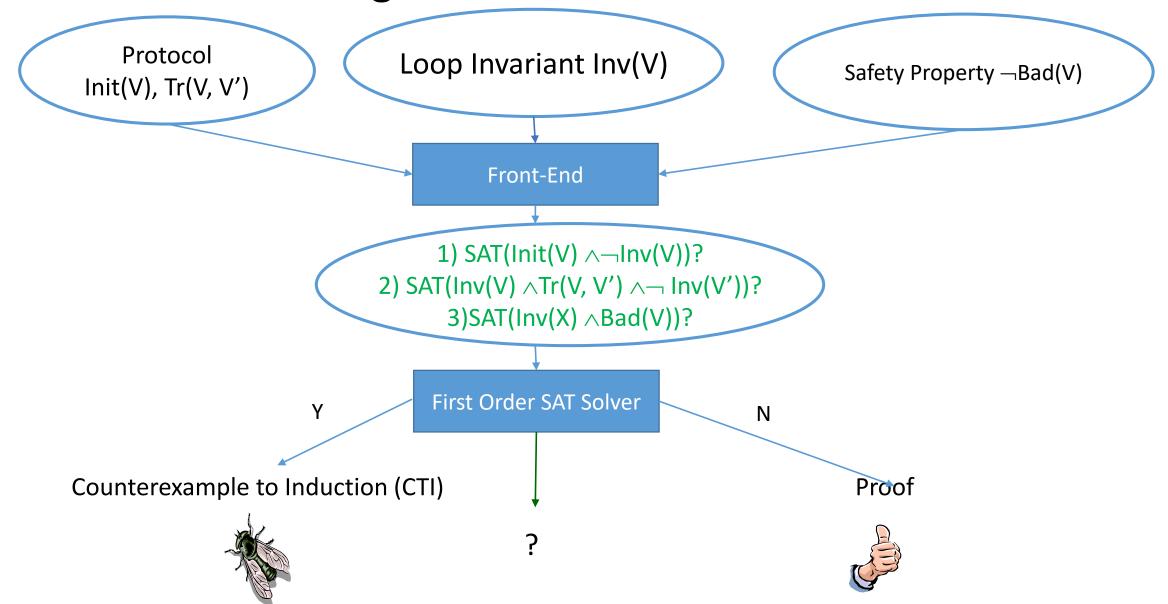
```
Init \subseteq Inv (Initiation) if \sigma \in Inv and \sigma \to \sigma' then \sigma' \in Inv (Consecution) Inv \cap Bad = \emptyset (Safety)
```

Logic-based deductive verification

- Represent Init, \rightarrow , Bad, Inv by logical formulas
 - Formula ⇔ 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



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

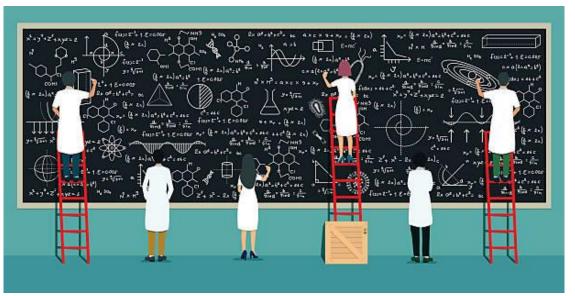
[POPL'16] Oded Padon, Neil Immerman, Aleksandr Karbyshev, Sharon Shoham, MS Decidability of Inferring Inductive Invariants

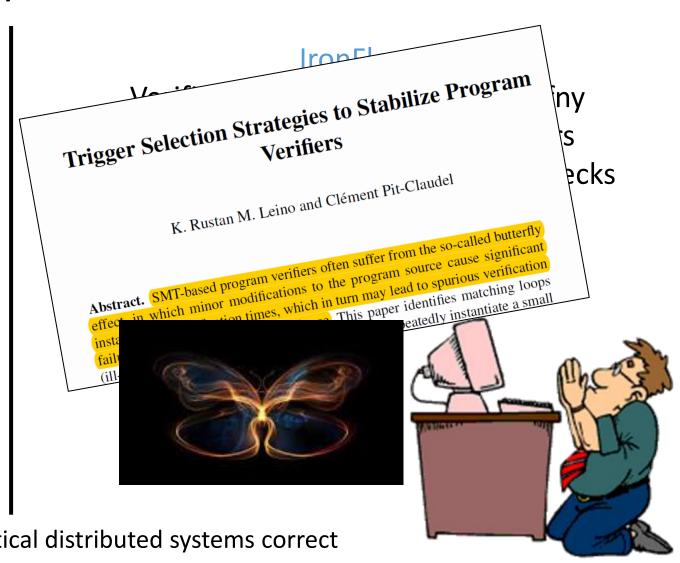
[JACM'17] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, Sharon Shoham: Property-Directed Inference of Universal Invariants or Proving Their Absence

Proving distributed systems is hard

Verdi

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





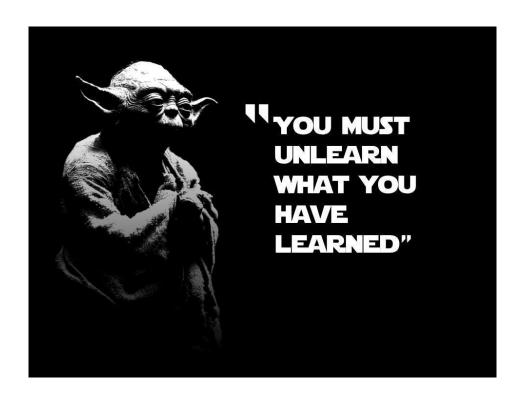
[SOSP'15] Hawblitzel et al. IronFleet: proving practical distributed systems correct

[PLDI'15] Wilcox et al. Verdi: a framework for implementing and formally verifying distributed systems

SAT Modulu 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

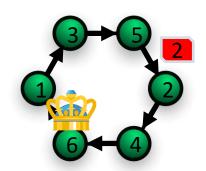
- Abstracts states as finite (uninterpreted) first order structures
 - Unbounded relations
 - No other data structures
 - Abstract integers, sets, cardinalities, ...

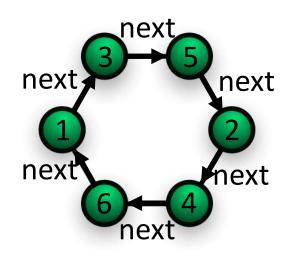
- Theories
- + Quantifiers

- Arbitrary loops and procedures
- Express program meaning as first order transition systems:
 - $r(X, Y) := \exists Z. p(X, Z) \land q(Z, Y) \equiv \forall X, Y. r'(X, Y) \Leftrightarrow \exists Z. p(X, Z) \land q(Z, Y)$
- "A step towards decidability"

Example: Leader election in a ring

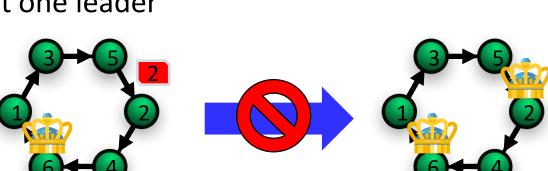
- Unidirectional ring of nodes, unique numeric ids
- Protocol:
 - Each node sends its id to the next
 - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node's own id
 - A node that receives its own id becomes a leader
- Theorem: The protocol selects at most one leader
 - Inductive?

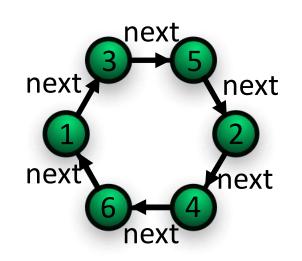




Example: Leader election in a ring

- 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



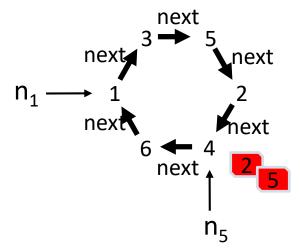


Modeling in first-order logic

State: finite first-order structure over vocabulary V:

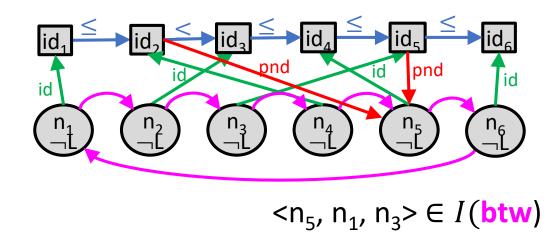
- ≤ (ID, ID) total order on node id's
- btw (Node, Node, Node) the ring topology
- id: Node → ID relate a node to its unique id
- pending(ID, Node) pending messages
- leader(Node) leader(n) means n is the leader

protocol state



first-order structure

Axiomatized in first-order logic

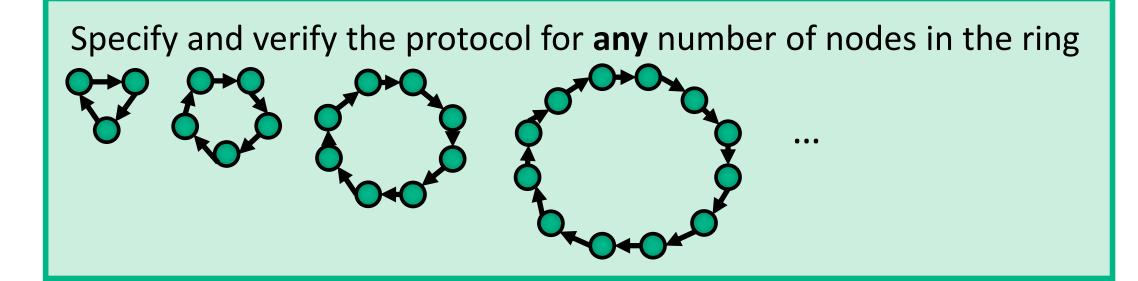


Modeling in first-order logic

State: finite first-order structure over vocabulary V:

- ≤ (ID, ID) total order on node id's
- btw (Node, Node, Node) the ring topology
- id: Node → ID relate a node to its unique id
- pending(ID, Node) pending messages
- leader(Node) leader(n) means n is the leader

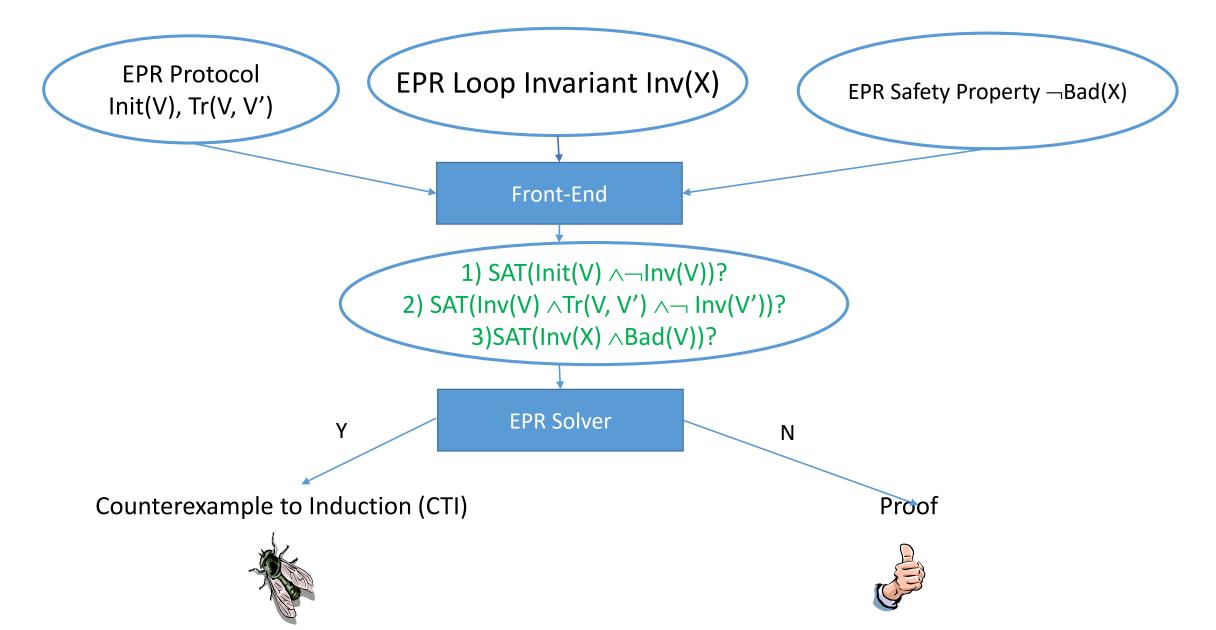
- Axiomatized in first-order logic



Modeling in first-order logic

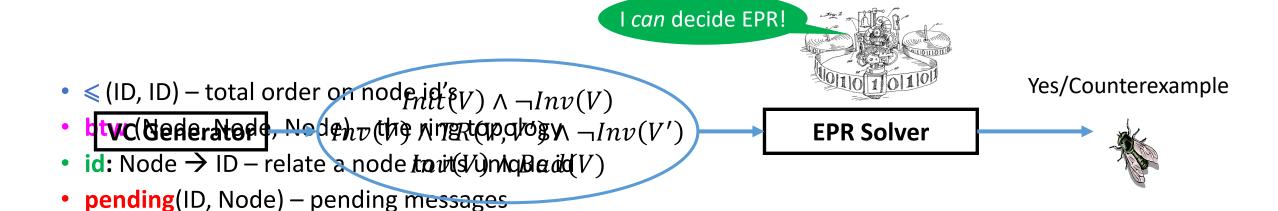
- State: finite first-order structure over vocabulary V (+ axioms)
- Initial states and safety property expressed as formulas:
 - Init(V) initial states, e.g., $\forall x,y . \neg pending(x,y)$
 - Bad(V) bad states, e.g., $\exists n_1, n_2$. leader(n_1) \land leader(n_2) $\land n_1 \neq n_2$
- Transition relation expressed as formula TR(V, V'), e.g.:
 - $\exists n,s.$ "s = next(n)" $\land \forall x,y. pending'(x,y) \leftrightarrow (pending(x,y) \lor (x=id[n] \land y=s))$
 - $\exists n. pending (id[n],n) \land \forall x. leader'(x) \longleftrightarrow (leader(x) \lor x=n)$

Deductive verification by reductions to EPR



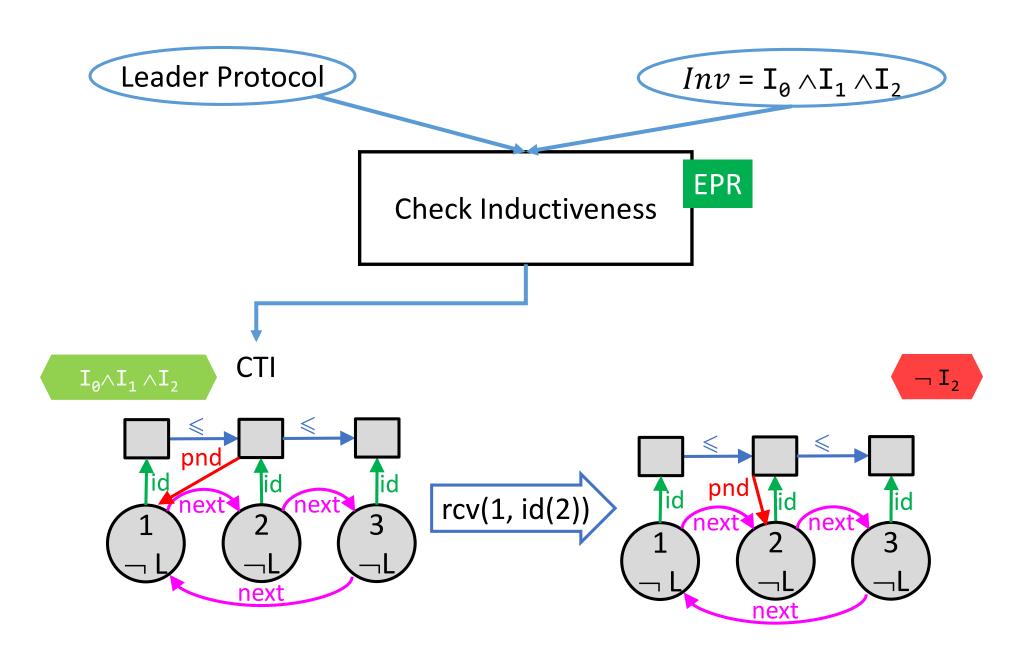
Leader election protocol – inductive invariant take 1

```
Inductive invariant: Inv = I_0 \wedge I_1 \wedge I_2
I_0 = \forall n_1, n_2 \colon \text{Node. leader}(n_1) \wedge \text{leader}(n_2) \rightarrow n_1 = n_2 \quad \text{Unique leader}
I_1 = \forall n_1, n_2 \colon \text{Node. leader}(n_2) \rightarrow \text{id}[n_1] \leqslant \text{id}[n_2] \quad \text{The leader has the highest ID}
I_2 = \forall n_1, n_2 \colon \text{Node. pending}(\text{id}[n_2], n_2) \rightarrow \text{id}[n_1] \leqslant \text{id}[n_2] \quad \text{Only the leader can be self-pending}
```



• leader(Node) – leader(n) means n is the leader

Ivy: check inductiveness



Leader election protocol – inductive invariant

```
Inductive invariant: Inv = I_0 \wedge I_1 \wedge I_2 \wedge I_3
                                                                                     Unique leader
 I_0 = \forall n_1, n_2: Node. leader(n_1) \land leader(n_2) \rightarrow n_1 = n_2
                                                                                      The leader has the highest ID
 I_1 = \forall n_1, n_2 : Node. leader(n_2) \rightarrow id[n_1] \leq id[n_2]
                                                                                         Only the leader can be self-pending
 I_2 = \forall n_1, n_2: Node. pending(id[n_2],n_2) \rightarrow id[n_1] \leq id[n_2]
 I_3 = \forall n_1, n_2, n_3: Node. btw(n_1, n_2, n_3) \land pending(id[n_2], n_1) \rightarrow id[n_3] \leqslant id[n_2]
                                                                                                    Cannot bypass higher nodes
                                                             I can decide EPR!
• \leq (ID, ID) – total order on node id'(V) \wedge \neg Inv(V)
                                                                                                                     Proof
   tvc(Sederated \phi, No d\phi) dv (the hingrapatory) -Inv(V')
                                                                                   EPR Solver
 id: Node \rightarrow ID – relate a node travit($\forall V)m\q\text{B}\equiv \( \delta \)
```

- pending(ID, Node) pending messages
- leader(Node) leader(n) means n is the leader

Skolemization

- Procedure that transforms a first order formula ϕ over vocabulary V=<S, C, R, F> into a universal formula Sk(ϕ) over vocabulary V'=<S, C U C', R, FU F'>
 - ϕ is satisfiable \Leftrightarrow Sk(ϕ) is satisfiable

- Example
 - $\forall X: S1. \exists y:S2. r(X, Y) \land q(Y)$

```
= _{SAT} 
\forall X: S1. r(X, f(X)) \land q(f(X))
```

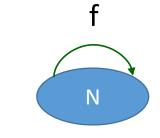
Why is SMT undecidable?

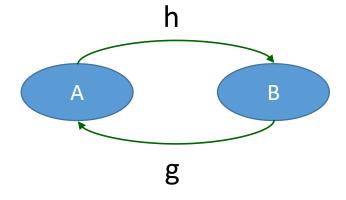
- Theories
 - $2 \times X^4 + 5 \times X^2 3 \times X + 2 = 0$
- Quantifier-alternation and function symbols (cycles)
 - ∀x: N. ∃y: N. x < y
 - $\forall x: N. x < f(x)$
 - $\forall x: A. \exists y: B. Q(x, y) \land \forall z: B. \exists w: A. P(z, w)$

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

- ∀x. le(x, x)
- $\forall x, y, z. le(x, y) \land le(y, x) \Rightarrow le(x, z)$
- $\forall x, y. le(x, y) \land le(y, x) \Rightarrow x=y$
- ∀x,y. le(x, y)∨le(y,x)
- ∀x. le(zero,x)
- ∀x. ∃y. le(x, y) ∧ x ≠y

For finite models validity is co-R.E.

Reflexive

Transitive

Antisymmetric

Total

Non-empty

Successor



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: $\exists^* \forall^* \varphi_{Q.F.}$
 - No ∀* ∃*







EPR Sat

Skolem

Herbrand

$$\exists x, y. \forall z. r(x, z) \leftrightarrow r(z, y)$$

$$=_{sat} \forall z . r(c_1, z) \leftrightarrow r(z, c_2)$$

$$=_{sat}(r(c_1, c_1) \leftrightarrow r(c_1, c_2)) \land (r(c_1, c_2) \leftrightarrow r(c_2, c_2))$$

$$=_{\text{sat}} (P_{11} \leftrightarrow P_{12}) \land (P_{12} \leftrightarrow P_{22})$$

SAT becomes undecidable

• ∀x. le(x, x)

• $\forall x, y, z. le(x, y) \land le(y, z) \Rightarrow le(x, z)$

• $\forall x, y. le(x, y) \land le(y, x) \Rightarrow x=y$

• $\forall x,y$. $le(x, y) \lor le(y, x)$

• ∀x. le(zero,x)

• ∀x. ∃y. le(x, y) ∧ x ≠y

Reflexive

Transitive

Antisymmetric

Total

Non-empty

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: $\exists^* \forall^* \varphi_{Q.F.}$
 - No ∀* ∃*
- 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

Designs

Low Level Low Level

EPR++ based verification

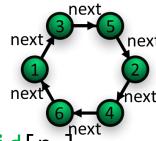
Predictiblity

- 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

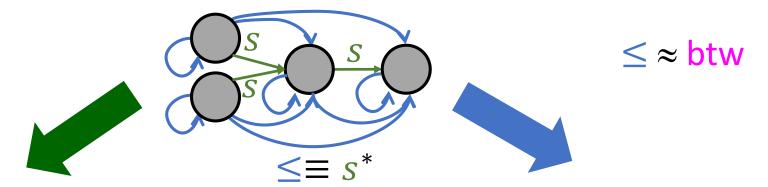


```
I_3 = \forall n_1, n_2, n_3: Node. btw(n_1, n_2, n_3) \land pending(id[n_2], n_1) \rightarrow id[n_3] \leq id[n_2]
```

Cannot bypass higher nodes

- Cannot express in first-order from "next" relation!
- Key enabler: use btw and not next

Key idea: representing deterministic paths



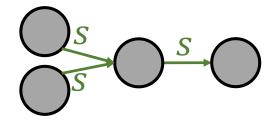
Alternative 1: maintain s

- defined by transitive closure of s
- not definable in first-order logic

Alternative 2: maintain ≤

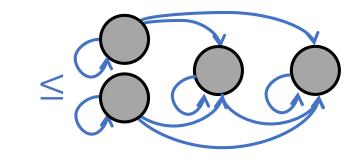
- s defined by transitive reduction of ≤
- Unique due to out degree 1
- Definable in first order logic

$$"s(x)=y" \equiv x < y \land \forall z. x < z \rightarrow y \le z$$
$$"x < y" \equiv x \le y \land x \ne y$$

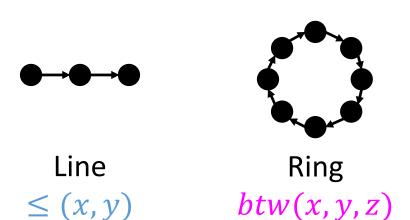


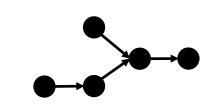
Not first order expressible

First order expressible

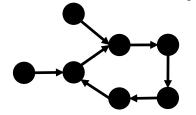


Sound and complete* axiomatization of deterministic paths





Forest, Tree, Acyclic partial function $\leq (x, y)$

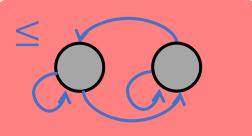


Graph with out degree 1, General partial function p(x, y, z)

For every class C of finite graphs above



Successor formula – 1 universal que

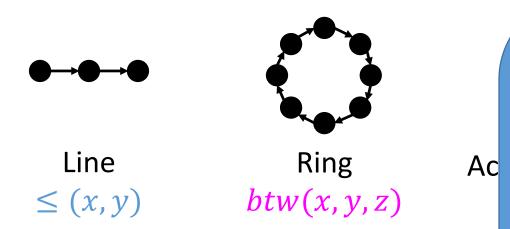


• Update formulas for node / edge addition and removal – universally quantified

• 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

Sound and complete* axiomatization of deterministic paths



EPR → finite model property+ Completeness Thm. for finite structures

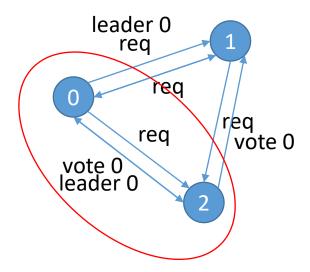
Sound and complete automatic deductive verification

For every class C of finite graphs above

- Axioms for path relation universally \(\)
- 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

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

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

• Solution: axiomatize cardinalities in first-order logic

```
\forall s, t. majority(s) \land majority(t) \rightarrow \exists p. member(p, s) \land 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. \negmember(p, s)
  procedure add ensures \forall p. member(p,r) \leftrightarrow (\text{member}(p,s) \lor p = e)
  property [maj] \forall s, t. majority(s) \land majority(t) \rightarrow \exists p. member(p, s) \land 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
invariant decision(N1,R1,V1) & decision(N2,R2,V2) -> V1 = V2
# proposals are unique per round
invariant proposal(R,V1) & proposal(R,V2) -> V1 = V2
# only vote for proposed values
invariant vote(N,R,V) -> proposal(R,V)
# decisions come from quorums of votes:
invariant 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
invariant one_b_max_vote(N,R2,none,V1) & ~le(R2,R1) -> ~vote(N,R1,V2)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none -> ~le(R,RM) & vote(N,RM,V)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none & ~le(R,RO) & ~le(RO,RM) -> ~vote(N,RO,VO)
# property of choosable and proposal
invariant ~le(R2,R1) & proposal(R2,V2) & V1 ~= V2 -> exists N. member(N,Q) & left_rnd(N,R1) & ~vote(N,R1,V1)
# property of one b, left rnd
invariant one_b(N,R2) & ~le(R2,R1) -> left rnd(N,R1)
```

Paxos made EPR: Proof size and verification time

Pro	otocol	Model [LOC]	Invariants	Verification time [sec]
Paxos	Paxos		11	2.2
Multi-Pa	Multi-Paxos		12	2.6
Vertical Paxos*		123	18	2.2
Fast Paxos*		117	17	6.2
Flexible Paxos		88	11	2.2
Stoppak	ole Paxos*	132	16	5.4

Abstraction and transformation to EPR reusable across all variants!

^{*}first mechanized verification

Appendix: The Proof of Correctness

We now prove that Stoppable Paxos satisfies its safety and liveness r ties. For clarity and conciseness, we write simple temporal logic for with two temporal operators: \Box meaning always, and \Diamond meaning eally [13]. We use a linear-time logic, so \Diamond can be defined by $\Diamond F \stackrel{\triangle}{=}$ for any formula F. For a state predicate P, the formula $\Box P$ assert P is an invariant, meaning that it is true for every reachable state temporal formula $\bigcirc\Box P$ asserts that at some point in the execution, Ffrom that point onward.

We define a predicate P to be stable iff it satisfies the following con-if P is true in any reachable state s, then P is true in any state reafrom s by any action of the algorithm. We let stable P be the assertio state predicate P is stable. It is clear that a stable predicate is invarit is true in the initial state. Because stability is an assertion only reachable states s, we can assume that all invariants of the algorith

true in state s when proving stability.

Our proofs are informal, but careful. The two complicated, mult proofs are written with a hierarchical numbering scheme in which (the number of the yth step of the current level-x proof [9]. Although appear intimidating, this kind of proof is easy to check and helps to

A.1 The Proof of Safety

We now prove that Consistency and Stopping are invariants of Stop

NotChoosphie(i k s) A

 $(\exists Q : \forall a \in Q : (bal[a] > b) \land (vote_i[a][b] \neq$ \forall $(\exists j < i, w \in StopCmd : Done2a(j, b, w))$ $\forall ((v \in StopCmd) \land (\exists i > i, w : Done2a(i, b$

We next prove a number of simple invariance and st_i

Lemma 1 1. $\forall i, b, v : \Box (Chosen(i, b, v) \rightarrow Done2a(i, b, v)).$ $2. \forall i, b, v, w : \Box ((Done2a(i, b, v) \land Done2a(i, b, v)))$ 3. $\forall i, b, a, v : \Box((vote_i[a][b] = v) \Rightarrow Done2a(i, b,$

Stoppable Paxos

Dahlia Malkhi Leslie Lamport Lidong Zhou

April 28, 2008

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.

Inv(i, b, v), it suffices to prove it for a partic-emma 1.7 (the stability of NotCheosoble(...)) $efore(i, b) \land NoneChoosableAfter(i, b, v)$ hat can possibly make PropInv(i, b, v) false true. We can therefore assume $s \to t$ is a quorum Q. Formula E1(b, Q) holds because it ase2a(i, b, v, Q) action. ROVE" clause of (1)1 are proved as steps (1)5,

 $\Rightarrow Done2a(j, mbal2a(j, b, Q), val2a(j, b, Q))$

some more definitions, culminating in the key invariant

safety proof is the following proof that Proplaw is invariant.

∧ NaneChoosableAfter(i, h, v)

 $\stackrel{a}{=} \forall c < b, w \neq v : NotChoosable(i, c, w)$ $b, w \in StopCmd : NotChoosable(j, c, w)$

 $md) \Rightarrow \forall j > i, c < b, w : NotChoosable(j, c, w)$

 \triangle Done2a(i, b, v) \rightarrow SafeAt(i, b, v)

 $(Rer(i, b, v) \triangleq$

i, b, v : PropInv(i, b, v))

PROOF. Assume $mbal2a(j, b, Q) \neq -\infty$. By definition of mbal2a, this implies val2a(j, b, Q) is a command (and not T). Since E1(b, Q) holds by assumption (1)1.4, the definitions of mbal2a and val2a imply that some acceptor ain Q has sent a ("1b", a, b, $(mbal^2a(j,b,Q),val^2a(j,b,Q)))_j$ message, which implies $vot_{i}[a|[mbal^2a(j,b,Q)] = val^2a(j,b,Q)$ when the message was sent. Lemma 1.3 then implies $Done^2a(j,mbal^2a(j,b,Q),val^2a(j,b,Q))$ was true when the message was sent, and is still true because $Done^2a(...)$ is stable.

(1)3. $\forall j, c < b, w : (c \le mbal2a(j, b, Q)) \land (w \ne val2a(j, b, Q)) \Rightarrow$ NotChoosable(1, c, w)

PROOF: We assume $c \leq \operatorname{molecule}_i(s, w)$ of $w \neq \operatorname{vol}2a(j, b, Q)$ and w prove NetChouske(j, i, w). Since $-\infty < c \leq \operatorname{molecule}_i(s, Q)$, step (1/2 implies $\operatorname{Decot}_2(s)$, $\operatorname{holecule}_i(s)$, $\operatorname{holecule}_i(s)$, $\operatorname{holecule}_i(s)$, $\operatorname{holecule}_i(s)$, which is implies $\operatorname{Sigle}_i(s)$, $\operatorname{holecule}_i(s)$, $\operatorname{holecule}_i($

instancing, monacouply, $u_i(y_i)$ volume $u_i(y_i)$, then unique i manifolds $u_i(y_i)$. ((), k, $y_i < c$, k w: $\{$ small (u_i) , $(u_i) = T\}$ so NotChoosuble(j, c, w). PRIOD: We assume c < b and small 2a(j, b, Q) = T and prove NotChoosuble(j, c, w). We split the proof into two cases. (2).1. Class: $mbol^2 2a(j, b, Q) = -\infty$

PROOF: The case assumption implies mbal2a(j, b, Q) < c, so assumption (1)1.4 and Lemma 3 imply NotChoosable(j, c, w). (2)2. Case: $mbal2a(j, b, Q) \neq -\infty$

(3) Constant measure (1), q(γ) = 1.00 PROOPS Since c > b, we can split the proof into the following three cases, (3)1. CASI: mbal2a(t), b, Q) < c < b PROOPS By assumption (1)1.4, the case assumption and Lemma 3 imply NotChoousble(t, c, w).</p>

(3)2. Case: $c \le mbal2a(z, b, Q)$ and $w \ne val2a(z, b, Q)$

PROOF: By (1)3. (3)3. CASE: $e = \operatorname{bind}(2a(j,b,Q)$ and $w = \operatorname{vol}(2a(j,b,Q))$ (4)1. $\operatorname{vol}(2a(j,b,Q)) \in \operatorname{StopCond}$ and $w \in \operatorname{can}$ choose k > j such that $\operatorname{mind}(2a(k,b,Q)) \geq \operatorname{mind}(2a(j,b,Q))$. PROOF: We deduce that $\operatorname{vol}(2a(j,b,Q)) \in \operatorname{StopCond}$ and such a k exists

by the (2)2 case assumption, the assumption $sval2a(i, b, O) = \top$, and

(4)2. Done2a(k, mbal2a(k, b, O), val2a(k, b, O))

(4)2. Dom2a(k, mbd2a(k, Q), mb2a(k, Q)) $Phoor: The (33 cose assumption and (4)1 imply <math>mba2a(k, k, Q) \neq -\infty$. Step (1)2 then proves (4)2. Q(3). Nac(Dovabed(x), c. w) $Phoor: Assumption (1)1.1 (with <math>j \leftarrow k, c \leftarrow mba2a(k, k, Q)$, and $w \leftarrow ua2a(k, k, Q)$) and (4)2 imply NoReconfigHere(x, mba2a(k, k, Q)). Step (4)1 asserts j < k; case assumption (3)3 and (4)1 imply.

NotChoosable(j, c, w).

(1)5. SafeAt(i,b,v)'PROOF: We assume c < b and $w \neq v$ and prove NotChoosable(i,c,w)'. By Lemma 1.7, it suffices to prove NotChoosable(i,c,w). We split the proof into two

(9)1 Cast: mul9a(i h O) - T

PROOF: (1)4 (substituting $j \leftarrow i$) implies NotChowable(i, c, w)

1959 Cast: and Pa(i h O) + T

2(2. Cose. seu 20(i, b, Q) ≠ i PROOF. Since c < b, we can break the proof into two sub-cases.</p>
(3)1. Cose: mba?2a(i, b, Q) < c < b</p>
PROOF. Assumption (1)1.4 and Lemma 3 imply NotChoosable (i, c, w)

Follow: Assumption (i.), i. a distribution is simply restribution (i.e., w) for v and v

(1)6. NoReconfigBefore(i, b)' (1)0. Noncompactor(i, i) : PROOF: We assume j < i, w ∈ StopCmd, and c ≤ b and we prove NotChoosable(j, c, w)'. By Lemma 1.7, it suffices to prove NotChoosable(j, c, w). Since c ≤ b, we need consider only the following two cases.

PROOF: Assumption (1)1.3 implies $Done2a(i, b, v)^i$. Since i > j and $w \in StopCmd$, this implies the third disjunct of NotChoosable(j, b, w)' (substituting i and v for the existentially quantified variables), which by the case assumption proves NotChoosable(j, c, w)'.

(2)2. CASE: c < b

(2)2. OASE: C< 0 PROOFE We consider two sub-cases.</p>
(3)1. CASE: swa22c(f, b, Q) = T PROOFE (M and case assumption (2)2 imply NotCheonable(f, c, w).
(3)2. CASE: swa22c(f, b, Q) ≠ T PROOFE (S) case assumption (2)2, we have the following two sub-cases.

(4)1. CASE: meal2a(j, k, Q) < ε < b PROOF: Assumption (1)1.4, the case assumption, and Lemma 3 imply NotChoosable(j, ε, ω).

NotiChosouble(j, c, w). (Q2 Casue: c_2 mod 2e(j, b, Q) PinOri: Assumption (1)1.3 implies $E\theta(i, b, Q)$. The (3)2 case assumption, the assumption j < i, and $E\theta(i, b, Q)$ imply surl2 $a(j, b, Q) \notin StopCnd$. The assumption $w \in StopCnd$ then insplies $w \neq sur2a(1, b, Q)$ by the (3)2 case cosumption and the definition of the c_2 case c_3 and c_4 are c_4 and c_4 are c_4 and c_4 are c_4 and c_4 are c_4 are c_4 and c_4 are c_4 are c_4 and c_4 are c_4 are c_4 and c_4 and c_4 are c_4 are c_4 and c_4 are c_4 and c_4 are c_4 and c_4 are c_4 are c_4 and c_4 are c_4 are c_4 and c_4 are c_4 and c_4 are c_4 are c_4 are c_4 and c_4 are c_4 and c_4 are c_4 are c_4 are c_4 are c_4 and c_4 and c_4 and c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 and c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 are c_4 are c_4 and c_4 are c_4 are c_4 are c_4 are c_4 and c_4 are c_4

nition of sval2a, we then have $w \neq val2a(j, b, Q)$. The (4)2 case as NotChoosable(j, c, w).

(1)7. NoneChosableAfter(i, b, v)'
PROOF: We assume $v \in StopCmd$, j > i, c < b, and w any command and we prove NotChoosable(j, c, w)'. By Lemma 1.7, it suffices to prove NotChosable(j, c, w). (2)1. Case: sval2a(i, b, Q) = ⊤

 $A(1, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot) = 1$ PROOF: Assumption (1)1.3 implies EA(i, b, Q, v), so the assumption $v \in StopCmd$ implies EAb(i, b, Q, v). The case assumption, the assumption j > i, and EAb(i, b, Q, v) imply sunPLa(j, b, Q, v) = T. The assumption c < b and step (1)4 then imply NotChoosable(j, c, w).

and step (1)4 then imply $NotLnoessee(j, \epsilon, w)$. (2)2. Case: sea2(i, k, Q) = v(3)1. sea22a(i, k, Q) = vel2a(i, k, Q) = v PROOF. Assumption (1)1.3 implies E3(i, k, Q, v), which implies sea22a(i, k, Q) = v. The case assumption and the definition of seal2a then implies a2a(i, k, Q) = v.

impass var2a(t, b, Q | v = v.) Q(3). Done 2a(t, nbal2a(t, b, Q), v). PROOP: (3)1, aspino (1), aspino

By the assumption $\epsilon < b$, it at finites to consider the belowing two cases, (b, b, c) = (b, c), and (b, c) =

PROOF: By definition of Consistency, it suffices to assume Chosen(i, b, v) and Chosen(i, c, w) and to prove v = w. Without loss of generality, we can assume $b \le c$. We then have two cases.

PROOF: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, c, w) imply Done2a(i, c, w). By Lemma 4, this implies

Chosen(i, b, v).

PROOF: Lemma 1.1 implies $Done2a(s, b, v) \wedge Done2a(s, c, w)$, which by Lemma 1.2 implies b = c.

PROOF: By definition of Stopping, it suffices to assume Chosen(i, b, v), Chosen(j, c, w), $v \in StopCmd$, and j > i and to obtain a contradiction. We split

the proof into two cases. I. Castic < c > b. It casts: < c > b. PROOF: Chosen(i, b, v) and Lemma 1.1 imply Done 2a(i, b, v). This and Lemma 1 imply Pone 2a(i, b, v), which by the case assumption and the assumptions $v \in StopCond$ and j > i implies NotChososhle(j, c, w). The assumption Chososhle(j, c, w) and Lemma 2 then provide the required contradiction.

CASE: $c \geq 0$ PROOF: Chosen(j, c, w) and Lemma 1.1 imply Donc2a(j, c, w). Lemma 4 then implies NoReconfigBefore(j, c). The case assumption, the as-sumptions $v \in SopCmd$ and j > i, and NoReconfigBefore(j, c) imply NotChocoable(i, b, w). The assumption Chosen(i, b, v) and Lemma 2 then pre-

Theorem 3 $\forall b, Q$: Progress(b, Q)PROOF: We assume P1(b, Q), P2(b, Q) and P3(b) and we must prove that then exists a v such that either $\lozenge Chosen(i, b, v)$ or $(v \in StopCmd) \land \lozenge Chosen(j, b, v)$,

(1): "OLD (1): (1) (1) implies that the balls b looke constants concurs a Proper P(1)(2) implies that the balls b looke constants concurs a Proper proper in (2) constantly receive the Plancia messages. Because fall[4] is set to a value c only by receiving a ballot c message, somewhore P(2) implies half [6] S. Hence, a must eventually receive the Plancia message statement of Plancia (4), b). By P2(b, Q), the Plancia message is sense to eventually received by the looker.

(1)2. ∀i. w : □(Bone2a(i.h.w) = ○(Bosen(i.h.w))

 $2(2, \forall 1, w : \Box(Dime2a(1, b, w) \Rightarrow \bigcirc Choosen(1, b, w))$ action has been executed Phicory Dime2a(1, b, w) means that a Phow2a(1, b, w) action has been executed sending a $(^{+}2a^{+}, b, w)$, message to every acceptor a. If a is in Q, then assumption $P_{2}(b, Q)$ implies that it eventually receives that message. Assumption: $P_{2}(b, Q)$ implies $bal[a] \subseteq b$, so P(b, Q) implies that every a in Q eventually executes $bal[a] \subseteq b$, so P(b, Q) implies that every a in Q eventually executes $bal[a] \subseteq b$, so P(b, Q) implies $bal[a] \in b$. So P(b, Q) implies $bal[a] \in b$. So P(b, Q) implies $bal[a] \in b$. So P(b, Q) implies $bal[a] \in b$.

 $\langle 1 \rangle 7$. NoneChoosableAfter(i, b, v)'

PROOF: We assume $v \in StopCmd$, 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) = ⊤

Proof: Assumption $\langle 1 \rangle 1.3$ implies E4(i, b, Q, v), so the assumption

Protocol	Model [LOC]	Invariants	Verification time [sec]
Stoppable Paxos*	132	16	5.4

sval2a(i, b, Q) = v. The case assumption and the definition of sval2a then implies val2a(i, b, Q) = v.

 $\langle 3 \rangle 2$. Done2a(i, mbal2a(i, b, Q), v)

PROOF: (3)1, assumption (1)1.4, and the definition of val2a imply $vote_i[a][mbal2a(i, b, Q)] = v$ 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.

 $\langle 3 \rangle 3$. Case: c < mbal2a(i, b, Q)

PROOF: Step $\langle 3 \rangle 2$ and assumption $\langle 1 \rangle 1.1$ imply NoneChoosableAfter(i, mbal2a(i, b, Q), v). By the case assumption and the assumptions $v \in StopCmd$ and j > i, this implies NotChoosable(j, c, w).

 $\langle 3 \rangle 4$. Case: mbal2a(i, b, Q) < c < b

 $\langle 4 \rangle 1$. mbal2a(j, b, Q) < mbal2a(i, b, Q)

PROOF: The assumption $v \in StopCmd$ and $\langle 3 \rangle 1$ imply $sval2a(i, b, Q) \in StopCmd$. Case assumption $\langle 2 \rangle 2$ and the definition of sval2a then imply mbal2a(k, b, Q) < mbal2a(i, b, Q) for all k > i.

 $\langle 4 \rangle 2$. NotChoosable(j, c, w)

Proof: $\langle 4 \rangle$ 1 and case assumption $\langle 3 \rangle$ 4 imply mbal2a(j, b, Q) < c < b. By assumption $\langle 1 \rangle$ 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

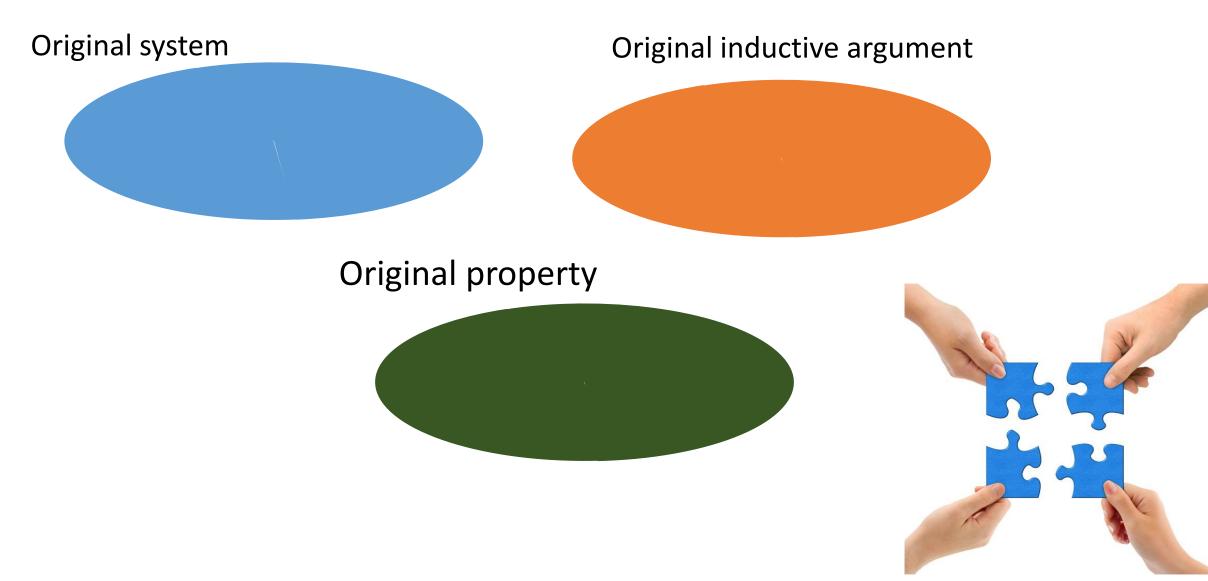


- Reasoning about abstract protocols (designs)
 - User provides axioms expressed in first order logic
 - Not checked by the system
 - Missing axioms can lead to false alarms
- Reasoning about implementations
 - Abstract total order \rightarrow concreter domain, e.g., integers
 - Abstract sets with majorities → 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

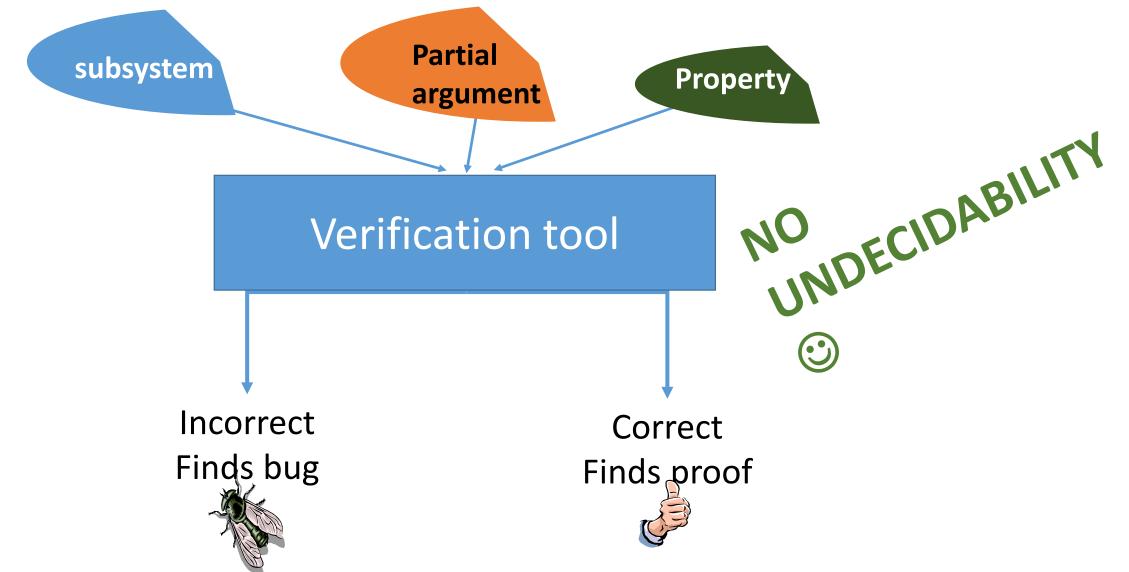
Ivy 2rd Principle: Scope Verification Conditions

- 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



Separate Verification of each module



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. \negmember(p, s)
  procedure add ensures \forall p. member(p,r) \leftrightarrow (\text{member}(p,s) \lor p = e)
  property [maj] \forall s, t. majority(s) \land majority(t) \rightarrow \exists p. member(p, s) \land 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 member(p, s)≡ ∃i. repr(s)[i] = p
 - Define cardinality of sets as a recursive function | |: set → int
 - majority(s)≡ |s| + |s| > |all|
 - Prove lemma by induction on |all|

```
\forall s, t. |s| + |t| > |all| \rightarrow \exists p. member(p, s) \land member(p, t)
```

- The lemma implies property maj
- All the verification conditions are in EPR+++limited arithmetic (FAU)

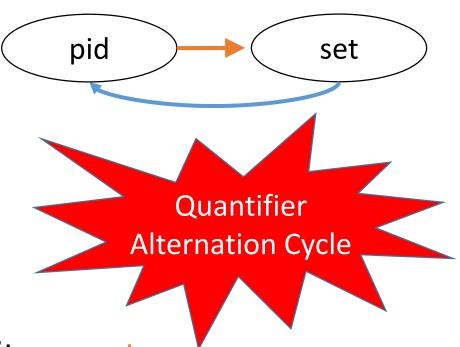
Quantifier alternation cycles

Protocol state

voters: pid → set

Property maj

 \forall s, t: set. \exists p: pid. majority(s) \land majority(t) \Rightarrow member(p, s) \land 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 \forall m. \neg voted(v, m);
    voted(v,n) := true;
}

procedure make_leader(n : pid, s : set) = {
    require majority(s);
    require \forall m. member(m, s) \rightarrow 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)
```

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 maj, we use the one leader invariant of the abstract model

```
\forall p, n. abs. voted(p, n) \rightarrow already\_voted(p)

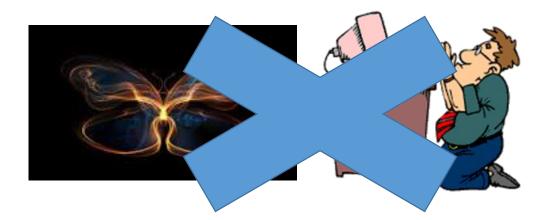
\forall p, n. network. vote(p, n) \leftrightarrow abs. voted(p, n)

\forall n. leader(n) \leftrightarrow 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

Protocol	System/Project	LOC	# manual proof	Ratio
	Coq/Verdi	530	50,000	94
RAFT	lvy	560	200	0.36
MULTIPAXOS	Dafny/IronFleet	3000	12,000	4
	lvy	330	266	0.8

Verification Effort

Protocol	System/Project	Human Effort	Verification Time
	Coq/Verdi	3.7 years	-
RAFT	lvy	3 months (from ground up)	Few min
	Dafny/IronFleet	Several years	6hr in cloud
MULTIPAXOS	lvy	1 month (pre-verified model)	few minutes on laptop



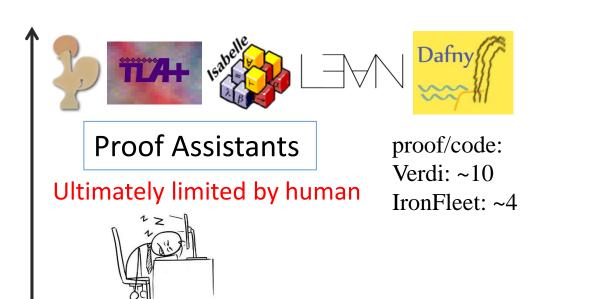
Why do people hate First Order Logic?

Rants	lvy
Hard to understand and error prone	Finite model property Display models graphically
Too weak: Cannot express Parity Numeric Quorums Finiteness Paths in a graph	First order interface Total orders Paths in deterministic graphs Majorities Theories as adds-on First order imperative updates
Hard for automation Satisfiability is undecidable NP-complete for fixed size	Restrict to EPR++/FAU Satisfiability is NEXPTIME complete/ Σ_2 Support from Yices, Z3, Iprover, Vampire

Languages and Inductiveness

Language	Executable	Expressiveness	Inductiveness
C, Java, Python	$\overline{\checkmark}$	Turing-Complete	Undecidable
SMV	X	Finite-state	Temporal Properties
TLA+	X	Turing-Complete	Manual
Coq, Isabelle/HOL	$\overline{\checkmark}$	"Turing-Complete"	Manual with tactics
Dafny	$\overline{\checkmark}$	Turing-Complete	Undecidable with lemmas
lvy	✓	Turing-Complete	Decidable(EPR++/FAU)

State of the art in formal verification





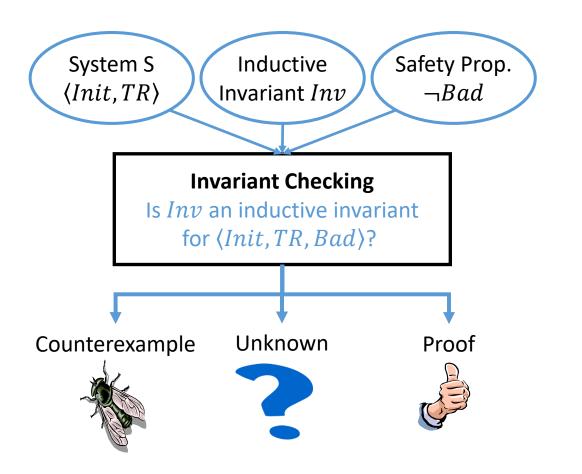
Decidable deduction Finite counterexamples proof/code: ~0.2

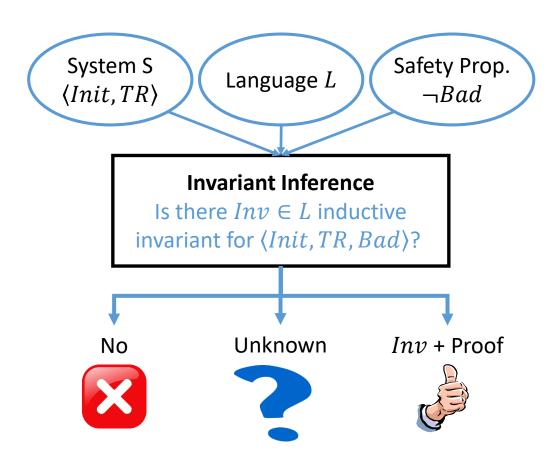
Ultimately limited by undecidability

Decidable Models Model Checking Static Analysis

Backup Slides

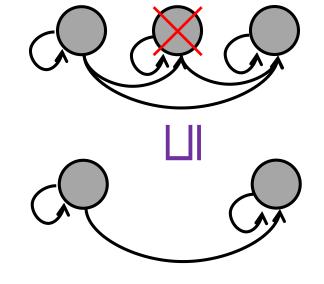
Inductive Invariant checking vs. inference

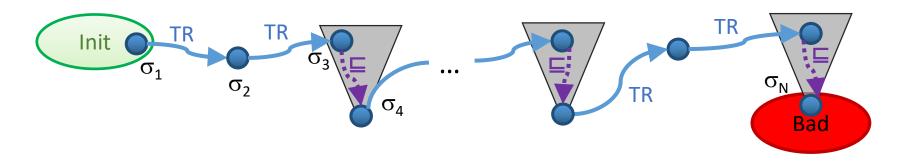




Relaxed error traces

- Notation: $\sigma \sqsubseteq \sigma'$ iff σ is isomorphic to a substructure of σ'
- $\sigma \sqsubseteq \sigma'$ implies σ satisfies more universal sentences than σ'
 - $\sigma \sqsubseteq \sigma', \psi \in \forall^*, \sigma' \models \psi \Rightarrow \sigma \models \psi$
- Relaxed error trace: σ_1 , σ_2 ,..., σ_N s.t. $\sigma_1 \models Init \quad \sigma_N \models Bad \quad \sigma_i$, $\sigma_{i+1} \models TR \text{ or } \sigma_{i+1} \sqsubseteq \sigma_i$



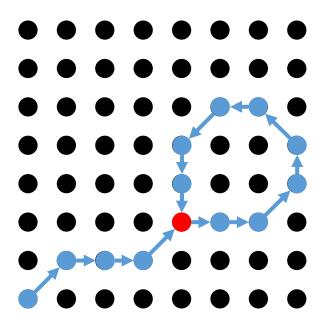


If there is a universal inductive invariant Inv $\in \forall^*$, then a relaxed error trace cannot exist

→ A relaxed error trace implies no universal inductive invariant exists

Key Idea: reduction to safety

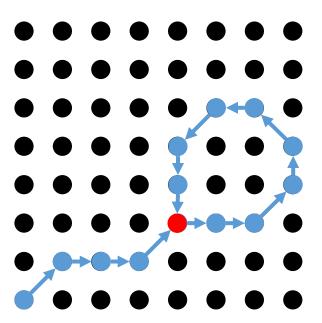
Finite State



Liveness ⇔ No Lasso

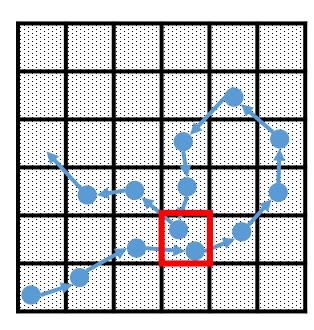
Key Idea: reduction to safety

Finite State



Liveness ⇔ No Lasso

Infinite State

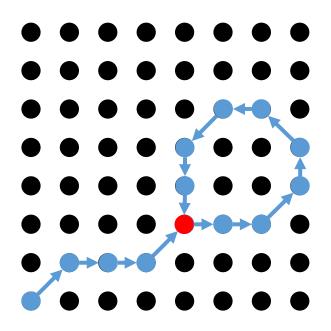


Liveness ← No Lasso

Problem: Spurious Lasso

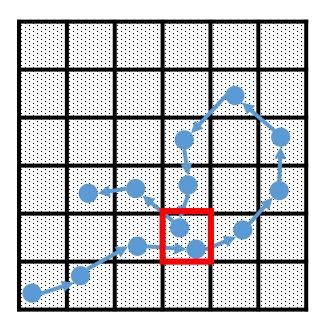
Key Idea: reduction to safety

Finite State



Liveness ⇔ No Lasso

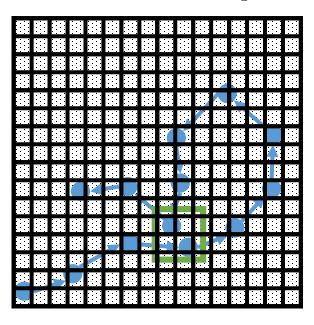
Infinite State



Liveness ← No Lasso

Problem: Spurious Lasso

Dynamic Abstraction [POPL'18]



Defined using First-Order Logic