PLDI Philadelphia 2018

deep spec

Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv

TEL AVIV UNIVERSITY

erc
European Research Council
Established by the European Commission

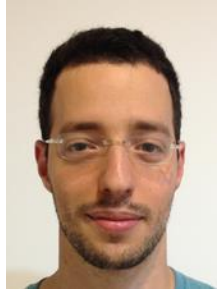
Contributors

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

TEL AVIV UNIVERSITY

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

Microsoft Research



TEL AVIV UNIVERSITY

TEL AVIV UNIVERSITY

James R. Wilcox, Doug Woos

UNIVERSITY of WASHINGTON

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

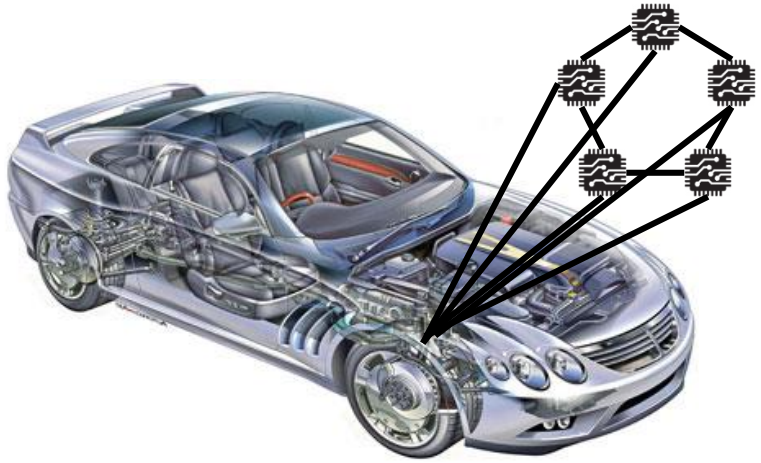
[PLDI'16] Oded Padon, Kenneth McMillan, Aurojit Panda, MS, Sharon Shoham
Ivy: Safety Verification by Interactive Generalization

[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

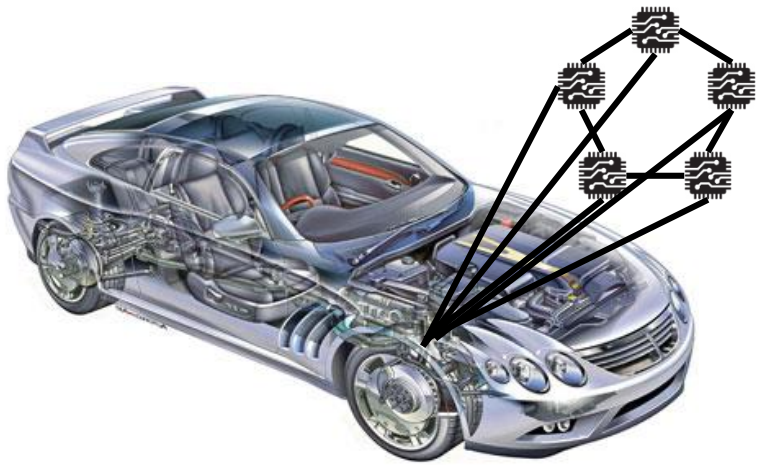
[PLDI'18] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, MS, Sharon Shoham, James R. Wilcox, Doug Woos: Modularity for Decidability of Deductive Verification with Applications to Distributed Systems

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

SIGCOMM'01

Chord: A Scalable Peer-to-Pe
for Internet Appli

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Kar
Hari Balakrishnan, *Memb*

Attractive features of Chord include
correctness, and provable performanc
concurrent node arrivals and departure

CCR'12

Using Lightweight Modeling To Understand Chord

Pamela Zave
AT&T Laboratories—Research
Florham Park, New Jersey USA
pamela@research.att.com

Under the same assumptions made in the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. Th
presented by means of

SOSP'07

Best Paper Award

Zyzzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong
Dept. of Computer Sciences
University of Texas at Austin

Zyzzyva is a state machine replication protocol based on
rotocols: (1) agreement, (2) view change, and (3)
ement protocol orders requests for exe-
change protocol coordinates

CACM'08

Zyzzyva: Sp
Fault

ACM Transactions on Computer Systems '09

Zyzzyva: Speculative Byzantine Fault Tolerance

RAMAKRISHNA KOTLA
Microsoft Research, Silicon Valley
and
LORENZO ALVISI, MIKE DAHLIN, ALLEN CLEMENT, and EDMUND WONG
The University of Texas at Austin

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

Revisiting Fast Practical Byzantine Fault Tolerance

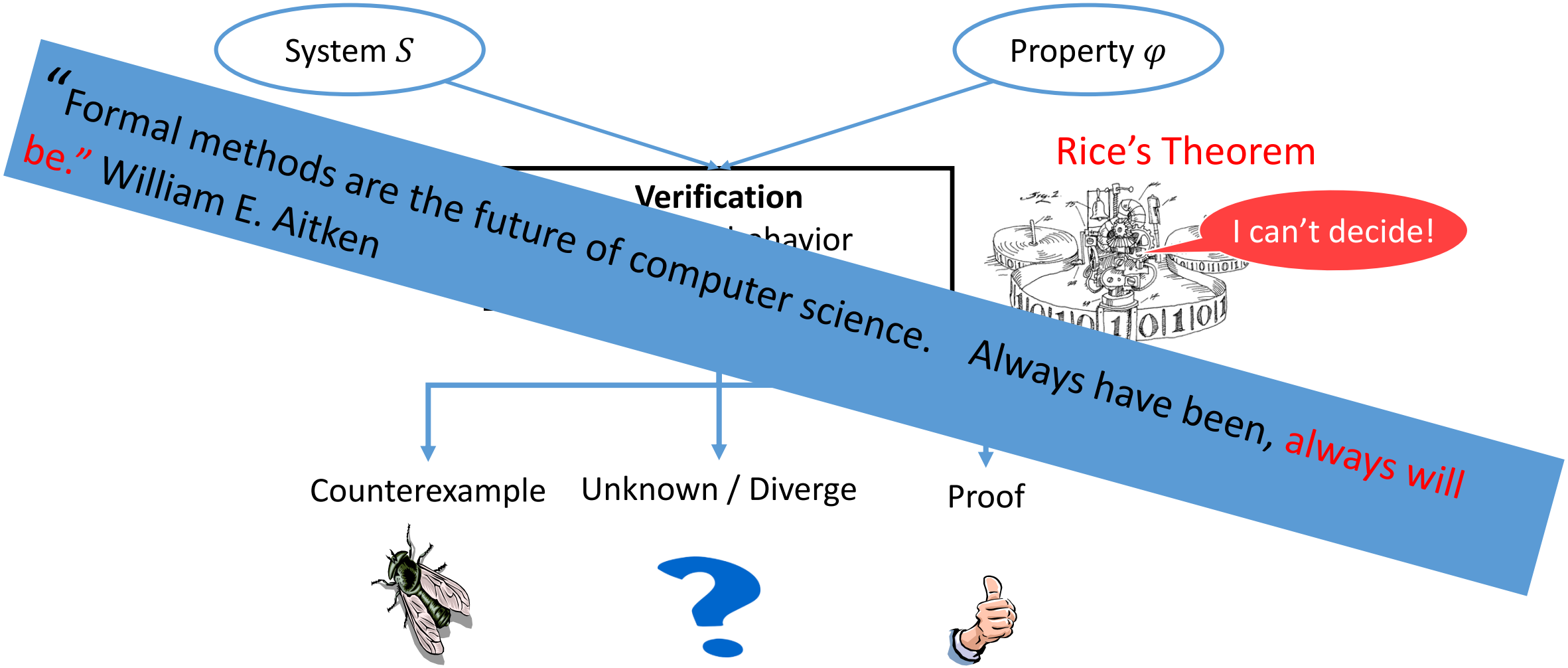
Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

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



"Formal methods are the future of computer science. Always have been, always will be." William E. Aitken

Deductive verification

System S

Inductive argument Inv

Property φ

Deductive Verification

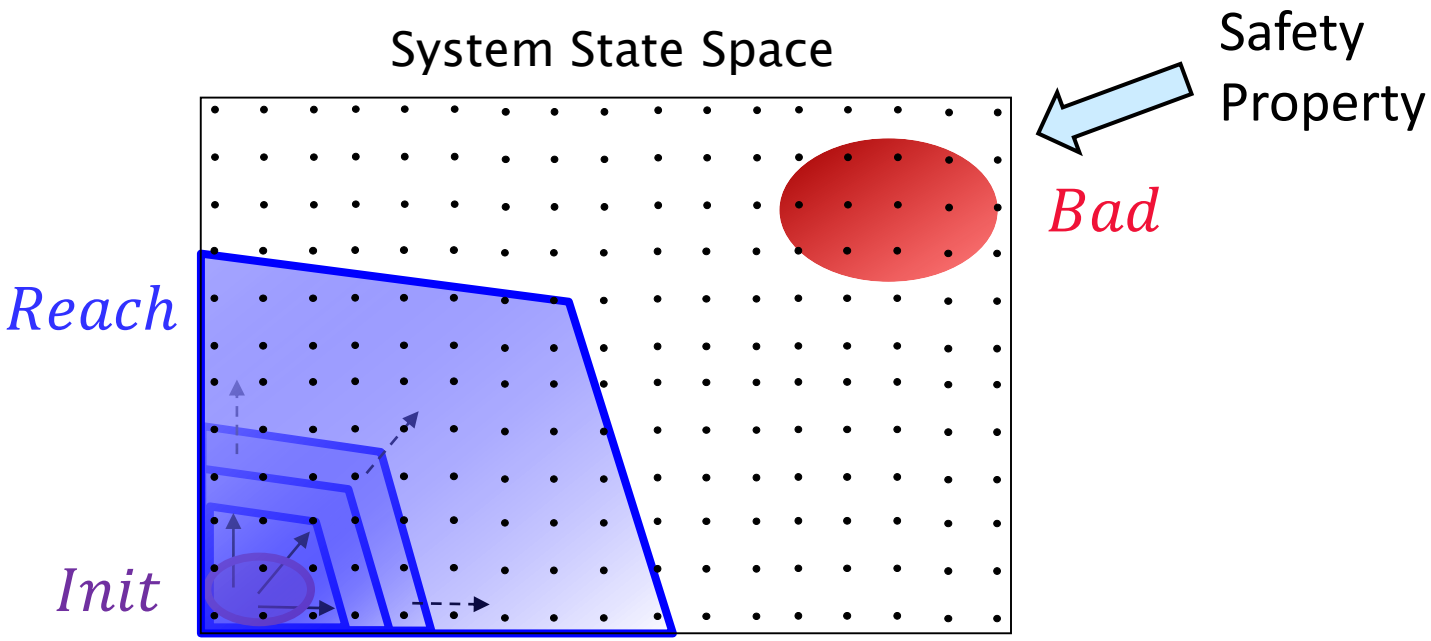
1) Is Inv an inductive invariant for S?
2) Does Inv enatil φ ?

Counterexample to Induction

Unknown / Diverge

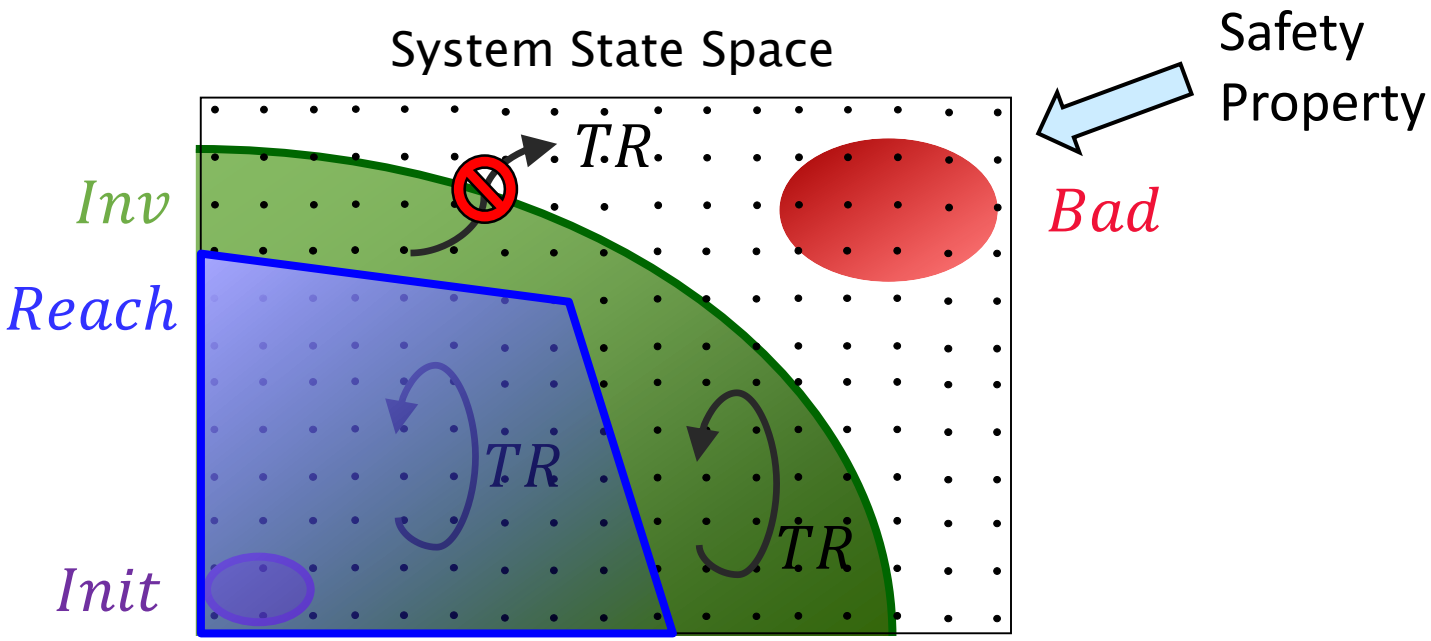
Proof

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 $\sigma \rightarrow \sigma'$ then $\sigma' \in Inv$ (**Consecution**)

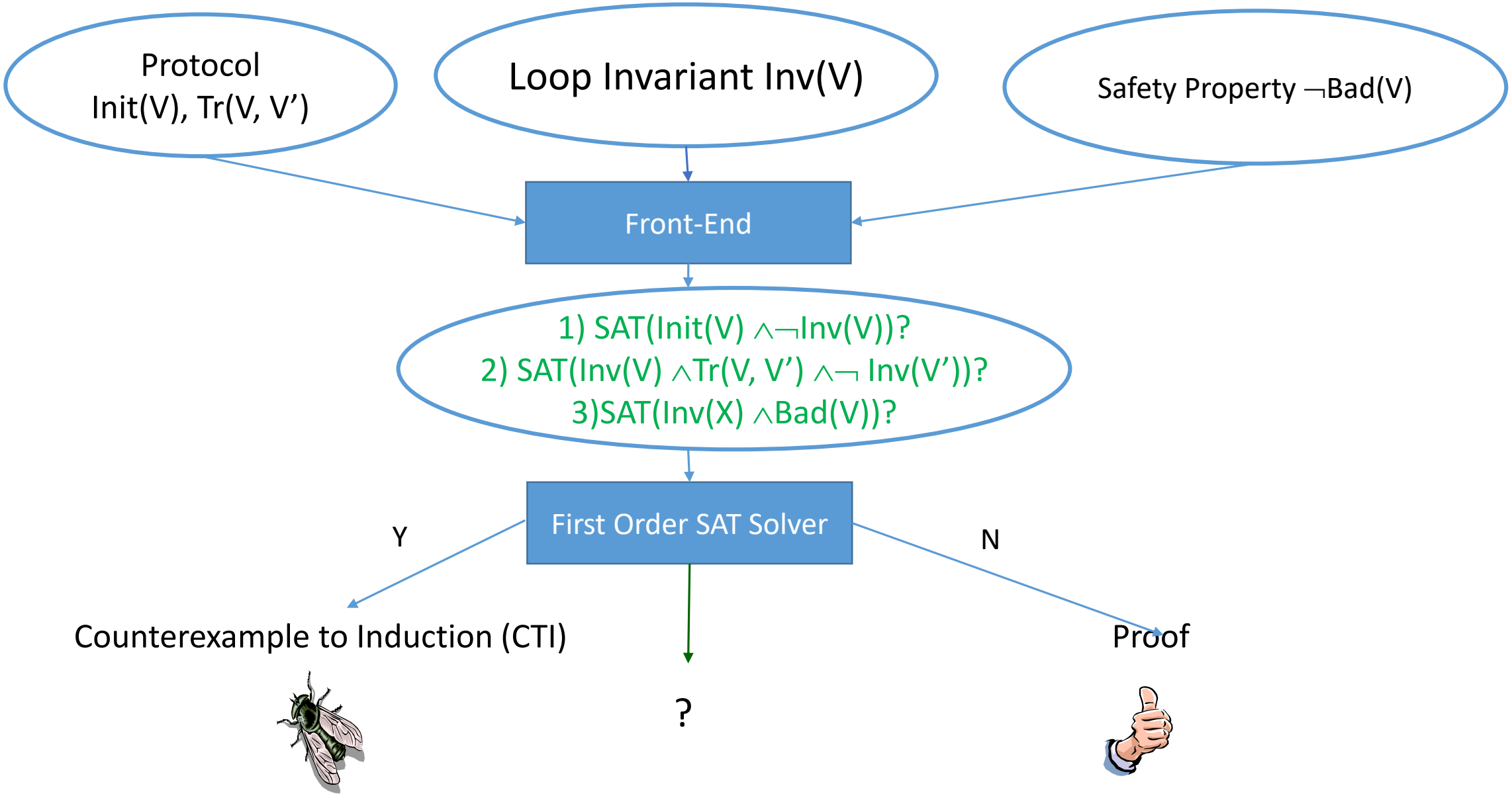
$Inv \cap Bad = \emptyset$ (**Safety**)

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

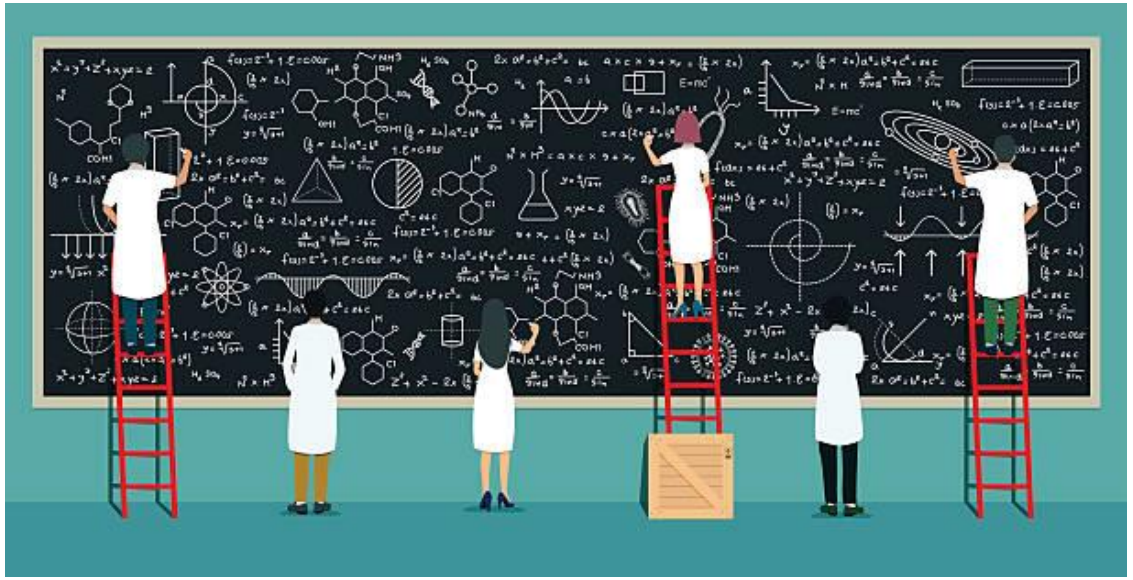
[JACM'17] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, Sharon Shoham:
Property-Directed Inference of Universal Invariants or Proving Their Absence

Proving distributed systems is hard

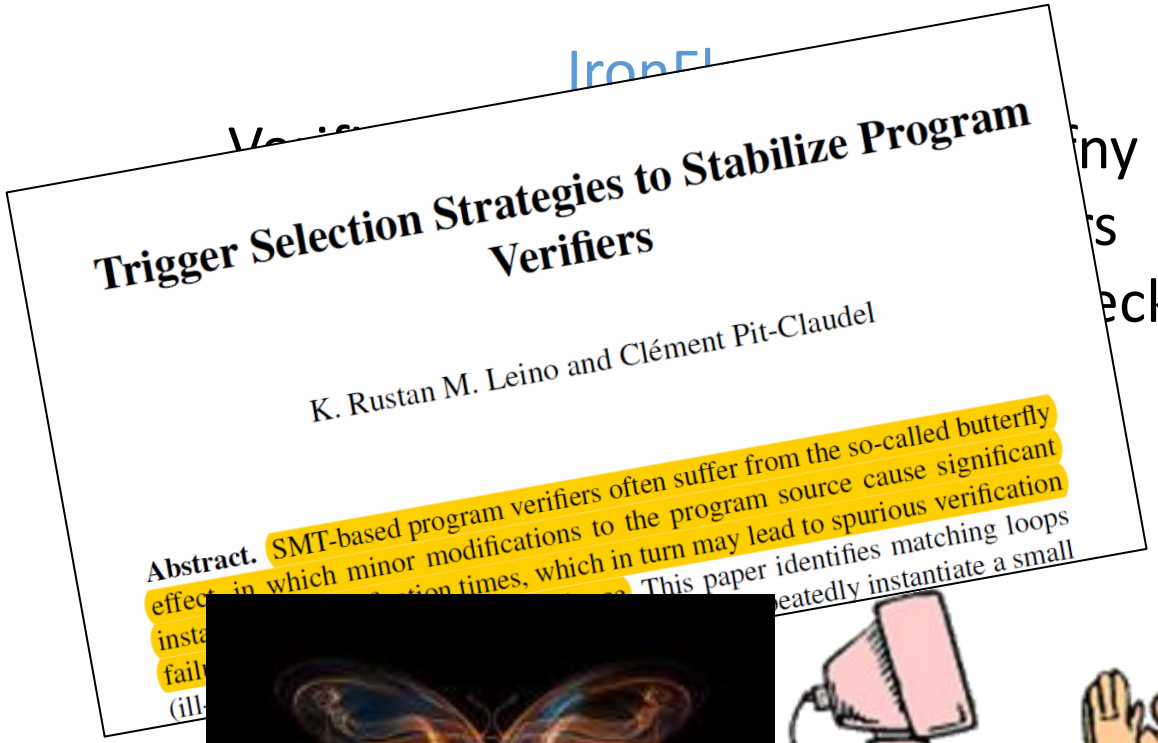
Verdi

Verification of Raft in Coq

50,000 lines of manual proof



IronFl



Trigger Selection Strategies to Stabilize Program Verifiers

K. Rustan M. Leino and Clément Pit-Claudel

Abstract. SMT-based program verifiers often suffer from the so-called butterfly effect, in which minor modifications to the program source cause significant ... times, which in turn may lead to spurious verification ... This paper identifies matching loops ... repeatedly instantiate a small

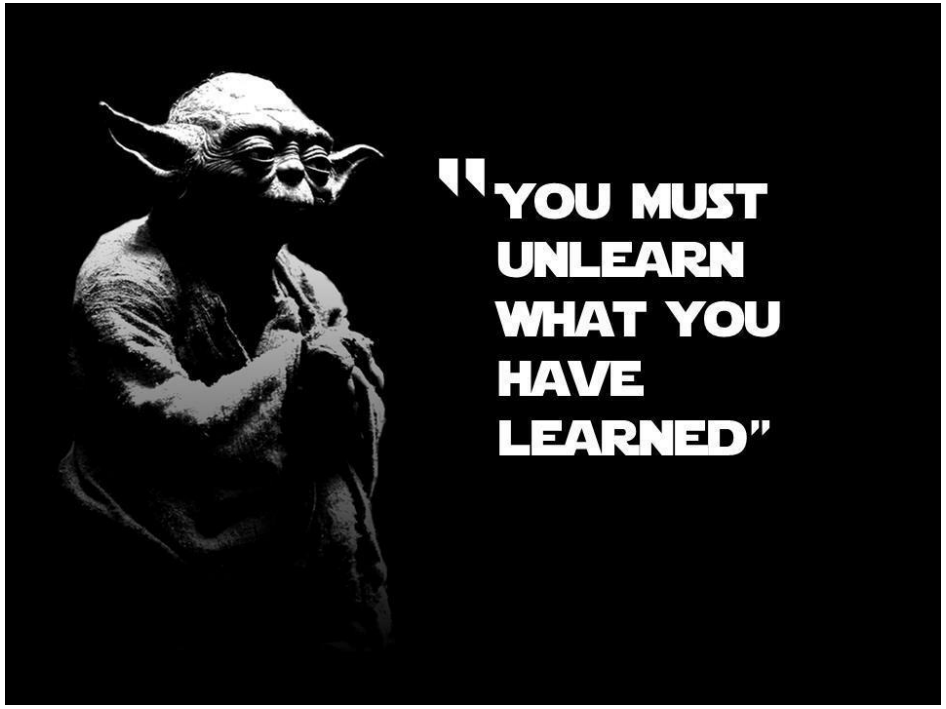
[SOSP'15] Hawblitzel et al. IronFleet: proving practical distributed systems correct

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

SAT Modulu Theory (SMT)

Used Sparingly in Ivy

- 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



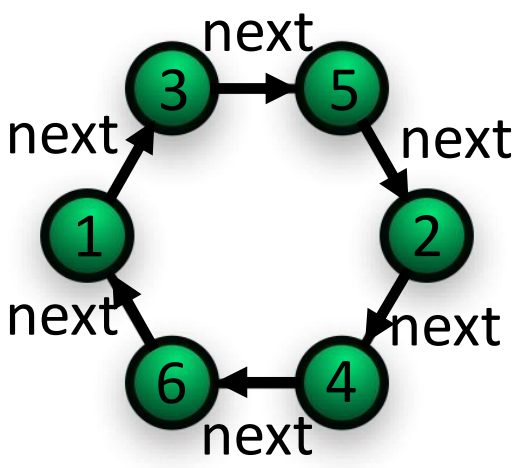
Ivy's 1st Principle: First Order Abstraction

- 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:
  - $r(X, Y) := \exists Z.\ p(X, Z) \wedge q(Z, Y) \equiv \forall X, Y.\ r'(X, Y) \Leftrightarrow \exists Z.\ p(X, Z) \wedge q(Z, Y)$
- "A step towards decidability"

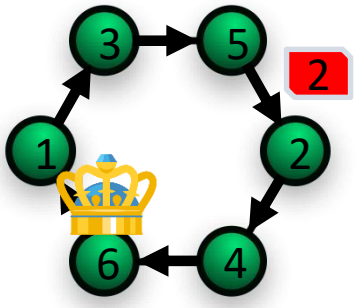
– Theories

+ Quantifiers

Example: Leader election in a ring

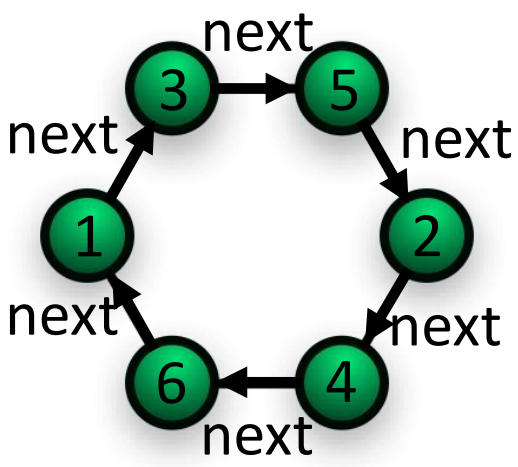


- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node's own id
  - A node that receives its own id becomes a leader
- Theorem: The protocol selects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

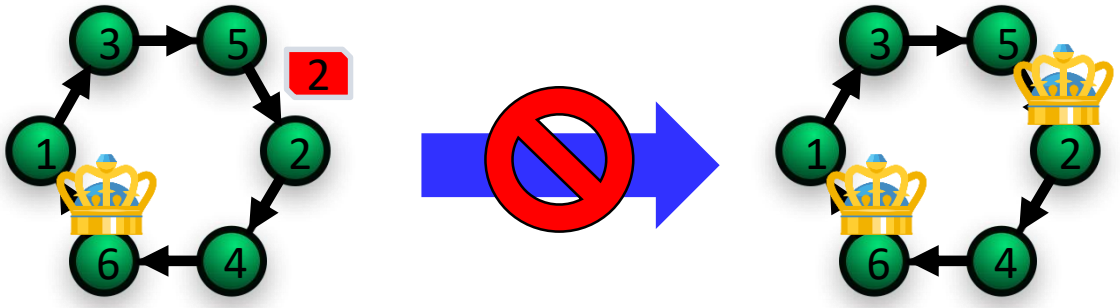


[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

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



[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

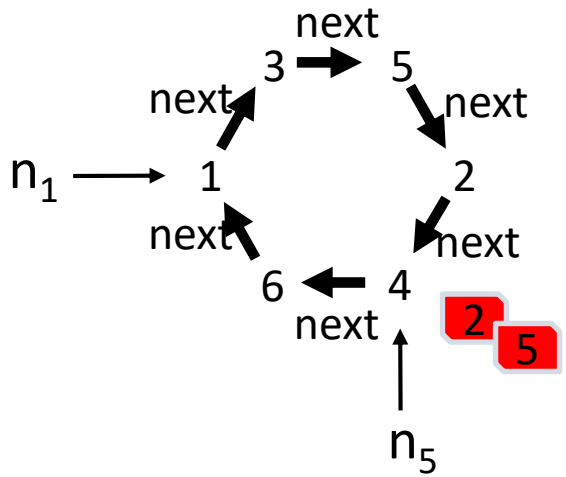
Modeling in first-order logic

State: finite first-order structure over vocabulary V :

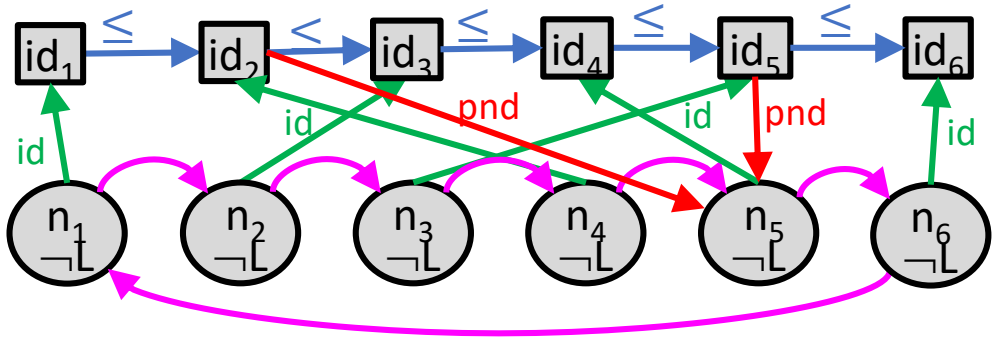
- $\leqslant$ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



first-order structure



$<n_5, n_1, n_3> \in I(\textbf{btw})$

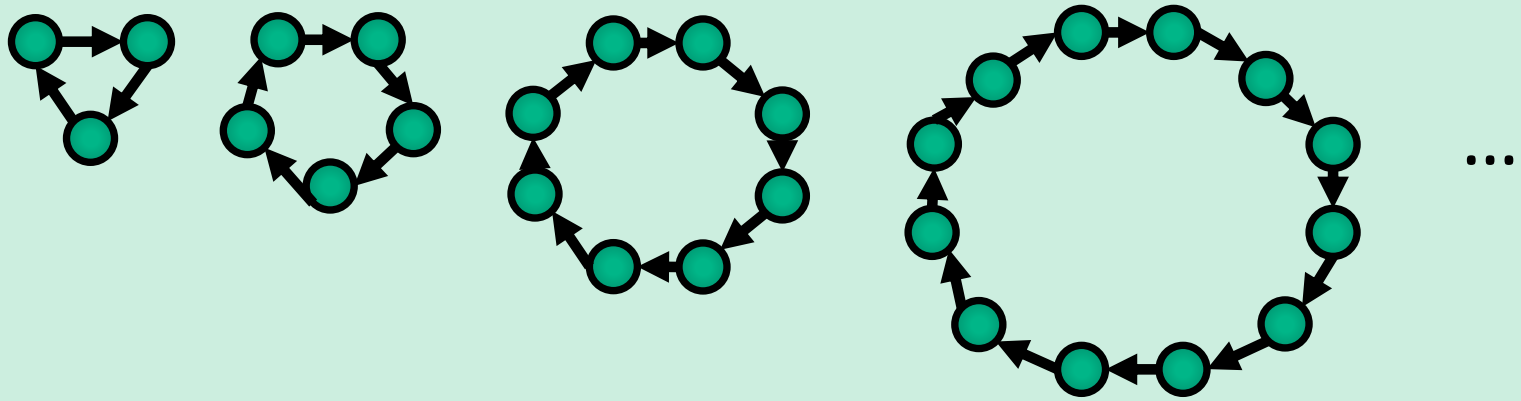
Modeling in first-order logic

State: finite first-order structure over vocabulary V :

- $\leqslant$ (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

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 (+ axioms)
- Initial states and safety property expressed as formulas:
  - Init(V) – initial states, e.g., $\forall x,y . \neg \mathbf{pending}(x,y)$
  - Bad(V) – bad states, e.g., $\exists n_1,n_2. \mathbf{leader}(n_1) \wedge \mathbf{leader}(n_2) \wedge n_1 \neq n_2$
- Transition relation expressed as formula TR(V, V'), e.g.:
  - $\exists n,s.$ "s = **next**(n)" $\wedge \forall x,y. \mathbf{pending'}(x,y) \leftrightarrow (\mathbf{pending}(x,y) \vee (x=\mathbf{id}[n] \wedge y=s))$
  - $\exists n. \mathbf{pending}(\mathbf{id}[n],n) \wedge \forall x. \mathbf{leader'}(x) \longleftrightarrow (\mathbf{leader}(x) \vee x=n)$

General Axioms

```
module total_order(le) = {
  axiom le(X,X)                          # Reflexivity
    axiom r(X, Y) & r(Y, Z) -> r(X, Z)  # Transitivity
    axiom r(X, Y) & r(Y, X) -> X = Y    # Anti-symmetry
    axiom r(X, Y) | r(Y, X)             # Totality
}
```

Ring Axioms

```
module ring_topology(carrier) = {
   individual head:carrier  # ring head
   individual tail:carrier  # ring tail
   relation le(X:carrier,Y:carrier)  # total order describing ring topology
   relation btw(X:carrier,Y:carrier, Z:carrier) # Y is on the acyclic path from X to Z
   instantiate total_order(le)  # total order
   axiom le(head, X)           # head is minimal
   axiom le(X, tail)           # tail is maximal
   # Axiom defining the btw relation
   axiom btw(X, Y, Z) <-> (
      (le(X, Y) & le(Y, Z)) |
      (le(Z, X) & le(X, Y) & X ~= Z) |
      (le(Y, Z) & le(Z, X) & X ~= Z)
   )
  action get_next(x:carrier) returns (y:carrier) = {
     assume (x = tail & y = head) | (le(x,y) & x ~= y & ((le(x, Z) & x ~= Z) -> le(y, Z)))
   }
   action get_prev(y:carrier) returns (x:carrier) = {
     assume (x = tail & y = head) | (le(x,y) & x ~= y & ((le(x, Z) & x ~= Z) -> le(y, Z)))
   }
}
```

Declarations

```
type node
type id
instantiate ring_topology(node)
relation le(X:id, Y:id)
instantiate total_order(le)
individual idn(X:node) : id
axiom idn(X)=idn(Y) -> X=Y  # the idn function is injective
relation leader(N:node)
init ~leader(N)
relation pending(V:id, N:node) # The identity V is pending at node N
init ~pending(V, N)
```

The send action

```
action send = {
    local n1:node, n2:node {
        # send my own id to the next node
        n2 := ring.get_next(n1);
        pending(idn(n1), n2) := true
    }
}
```

The Receive Action

```
action receive = {
    local n1:node, n2:node, m:id {
        # receive a message from the right neighbor
        assume pending(m, n1);
        pending(m, n1) := *; # abstract the number of pending messages
        if m = idn(n1) {  # Found a leader
            leader(n1) := true
        } else {
            if le(idn(n1), m) { # pass message to next node
                n2 := ring.get_next(n1);
                pending(m, n2) := true
            } # otherwise drop the message...
        }
    }
}
```

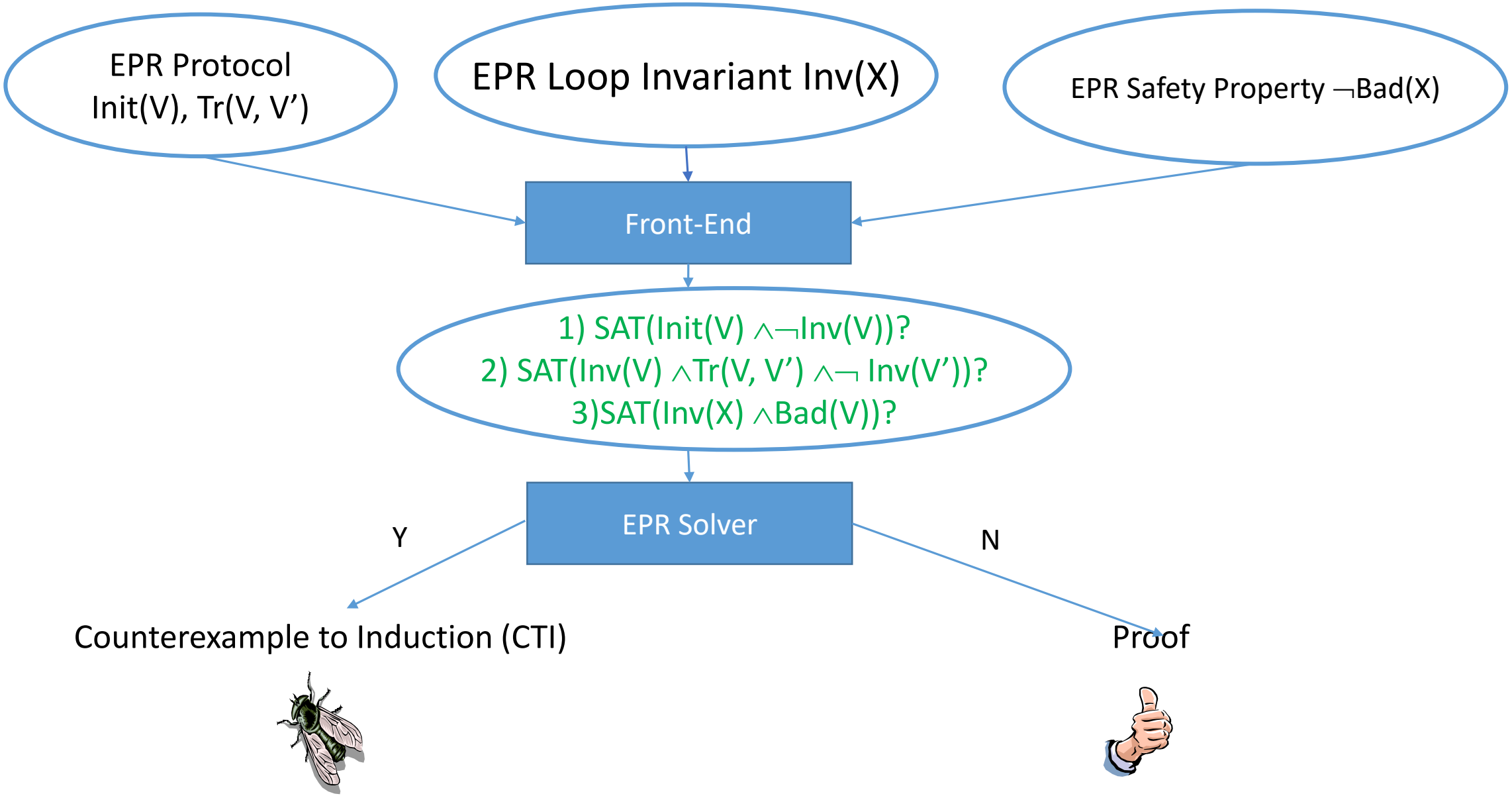
The Protocol

```
export send
export receive

# The safety property:
conjecture leader(X) & leader(Y) -> X = Y  # at most one leader
conjecture leader(X) -> le(idn(Y), idn(X)) # leader has highest id


# conjectures obtained via CTI's
# conjecture ~(le(idn(N1),idn(N0)) & pending(idn(N1),N1) & idn(N1) ~= idn(N0))
# conjecture ~(le(idn(N2),idn(N0)) & pending(idn(N2),N1) & ring.btw(N0,N1,N2) & N1 ~= N0)
```

Deductive verification by reductions to EPR



Leader election protocol – inductive invariant take 1

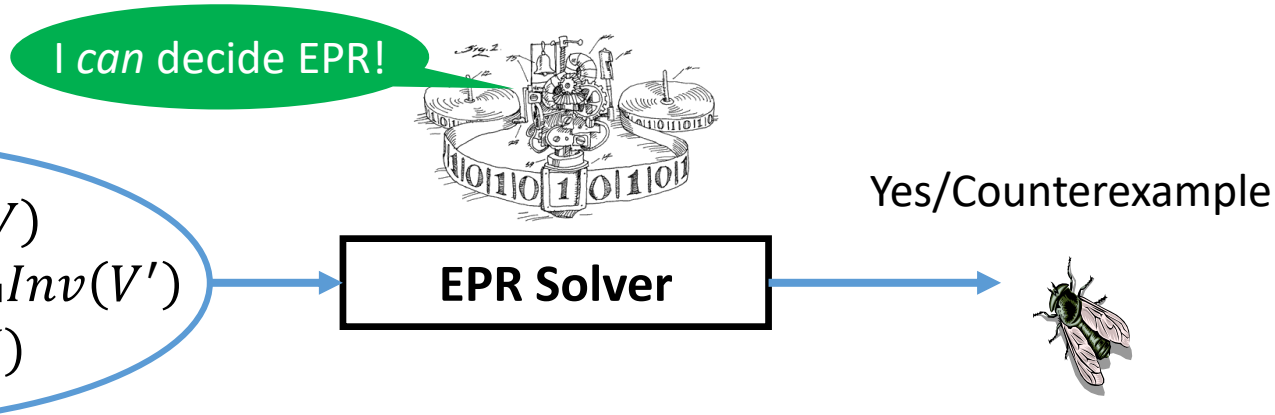
Inductive invariant: $Inv = I_0 \wedge I_1 \wedge I_2$

I_0 = ∀n_1,n_2: Node. **leader**(n_1) ∧**leader**(n_2) → n_1 = n_2 — Unique leader

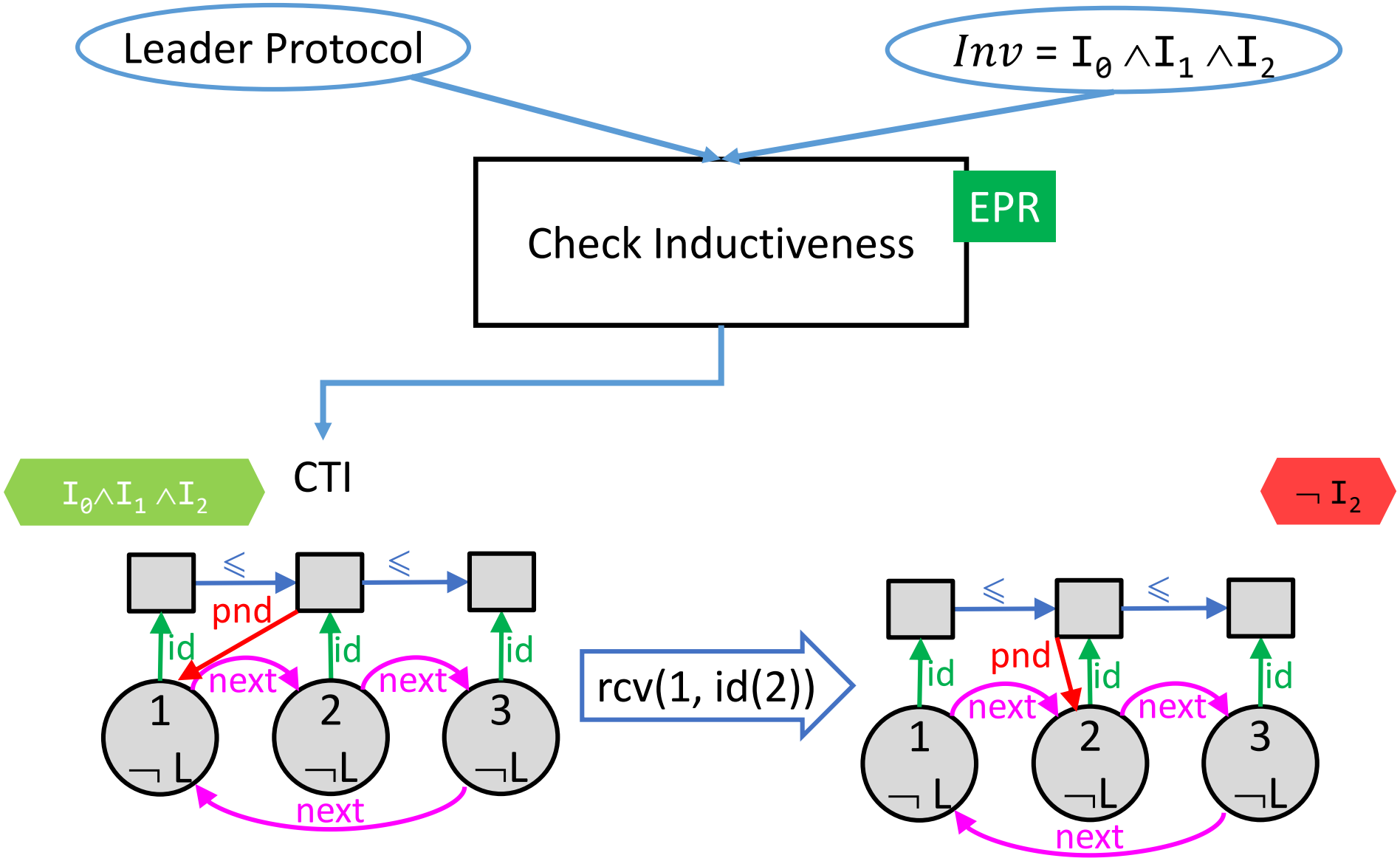
I_1 = ∀n_1,n_2: Node. **leader**(n_2) → id[n_1] ⩽ id[n_2] — The leader has the highest ID

I_2 = ∀n_1,n_2: Node. pending(id[n_2],n_2) → id[n_1] ⩽ id[n_2] — Only the leader can be self-pending

- ⩽ (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



Ivy: check inductiveness



Leader election protocol – inductive invariant

Inductive invariant: $Inv = I_0 \wedge I_1 \wedge I_2 \wedge I_3$

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

Unique leader

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

The leader has the highest ID

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

Only the leader can be self-pending

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

Cannot bypass higher nodes

- $\leqslant$ (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

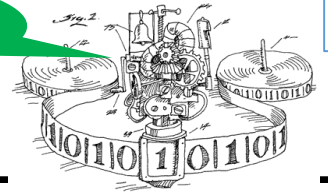
VC Generator

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

$Inv(V) \wedge TR(V, V') \wedge \neg Inv(V')$

$Inv(V) \wedge Bad(V)$

I *can* decide EPR!



EPR Solver

Proof

Skolemization

- Procedure that transforms
  a first order formula φ over vocabulary $V=\langle S, C, R, F\rangle$ into
  a universal formula $Sk(\varphi)$ over vocabulary $V'=\langle S, C \cup C', R, F \cup F'\rangle$
  - φ is satisfiable $\Leftrightarrow$ $Sk(\varphi)$ is satisfiable

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

    $=_{SAT}$

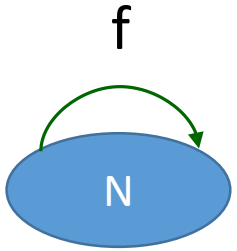
    $\forall X{:}\ S1.\ r(X, f(X)) \wedge q(f(X))$

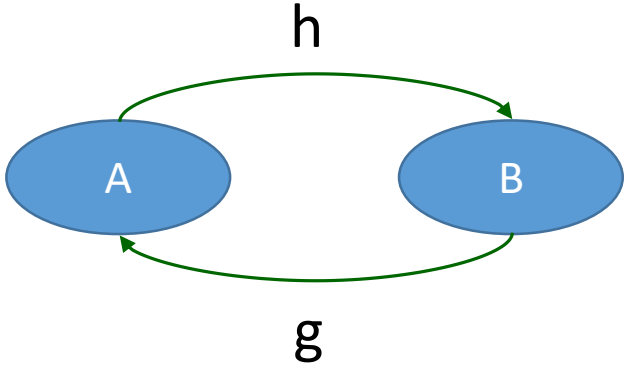
Why is SMT undecidable?

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

Also happens without theories

  - $\forall x: A.\ Q(x, h(x)) \wedge \forall z: B.\ P(z, g(z))$
    $h: A \rightarrow B$ and $g: B \rightarrow A$





Infinite Structures

- $\forall x.\ le(x, x)$ Reflexive
- $\forall x, y, z.\ le(x, y) \wedge le(y, x) \Rightarrow le(x,z)$ Transitive
- $\forall x, y.\ le(x, y) \wedge le(y, x) \Rightarrow x=y$ Antisymmetric
- $\forall x,y.\ le(x, y) \vee le(y,x)$ Total
- $\forall x.\ le(zero,x)$ Non-empty
- $\forall x.\ \exists y.\ le(x, y) \wedge x \neq y$ 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

Skolem

Herbrand

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

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

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

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

SAT becomes ~~un~~decidable

- $\forall x.\ le(x, x)$ Reflexive
- $\forall x, y, z.\ le(x, y) \wedge le(y, z) \Rightarrow le(x, z)$ Transitive
- $\forall x, y.\ le(x, y) \wedge le(y, x) \Rightarrow x=y$ Antisymmetric
- $\forall x,y.\ le(x, y) \vee le(y, x)$ Total
- $\forall x.\ le(zero,x)$ Non-empty
- ~~$\forall x.\ \exists y.\ le(x, y) \wedge x \neq y$ 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

Designs

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

Low Level Implementations

[CAV'07] Ge & de Moura: *Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories*

EPR++ based verification

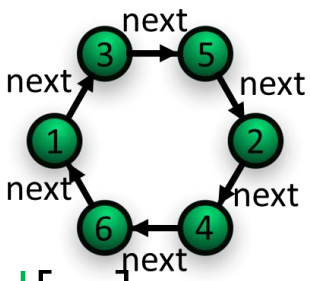
Predictiblity

- 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



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

Cannot bypass higher nodes

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

relation btw (Node, Node, Node)

axiom $\forall x, y, z$: Node. $\text{btw}(x, y, z) \rightarrow \text{btw}(y, z, x)$ *circular*

axiom $\forall x, y, z, w$: Node. $\text{btw}(w, x, y) \wedge \text{btw}(w, y, z) \rightarrow \text{btw}(w, x, z)$ *transitive*

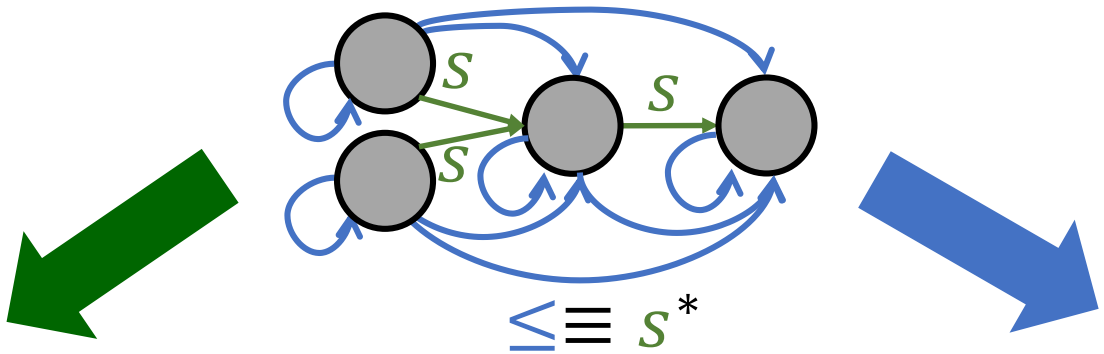
axiom $\forall x, y, w$: Node. $\text{btw}(w, x, y) \rightarrow \neg\text{btw}(w, y, x)$ *anti-symmetric*

axiom $\forall x, y, w$: Node. $\neq(w, x, y) \rightarrow \text{btw}(w, x, y) \vee \text{btw}(w, y, x)$ *total*

macro "$\text{next}(a)=b$" $\equiv \forall x$: Node. $x=a \vee x=b \vee \text{btw}(a,b,x)$ *edges*

[Itzhaky SIGPLAN Dissertation Award 2016]

Key idea: representing deterministic paths



$\leq \approx$ btw

Alternative 1: maintain s

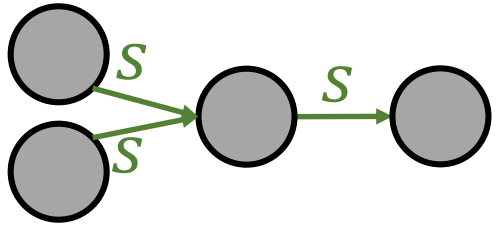
- $\leq$ defined by transitive closure of s
- not definable in first-order logic

Alternative 2: maintain $\leq$

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

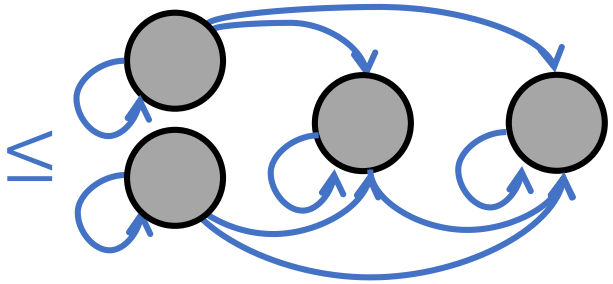
"s(x)=y" $\equiv x < y \wedge \forall z. x < z \rightarrow y \leq z$

"x<y" $\equiv x \leq y \wedge x \neq y$



Not first order expressible

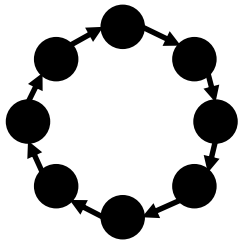
First order expressible



Sound and complete* axiomatization of deterministic paths

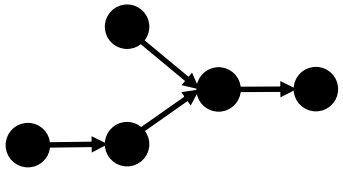
Line

$\leq (x, y)$



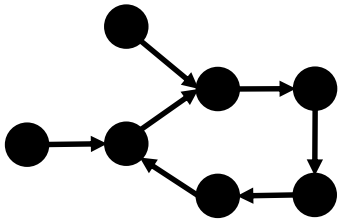
Ring

$btw(x, y, z)$



Forest, Tree,
Acyclic partial function

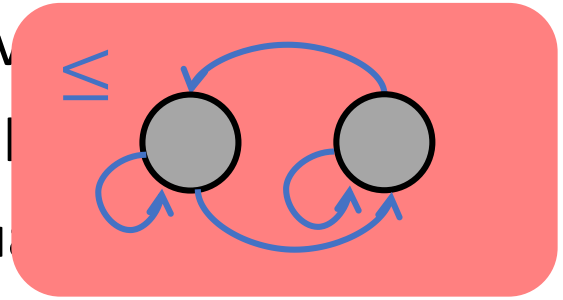
$\leq (x, y)$



Graph with out degree 1,
General partial function

$p(x, y, z)$

For every class C of finite graphs abov

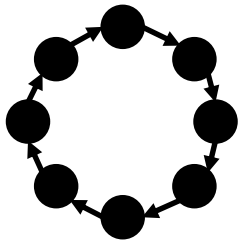


- Axioms for path relation – universal
- Successor formula – 1 universal qu
- Update formulas for node / edge addition and removal – universally quantified
- Soundness Theorem — *Every graph of class C satisfies the axioms of C*
  Edges agree with successor formula
- Completeness Theorem — *Every finite structure satisfying the axioms of C is isomorphic (paths and edges) to a graph of class C*

Sound and complete* axiomatization of deterministic paths

Line

$\leq (x, y)$



Ring

$btw(x, y, z)$

Ac

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

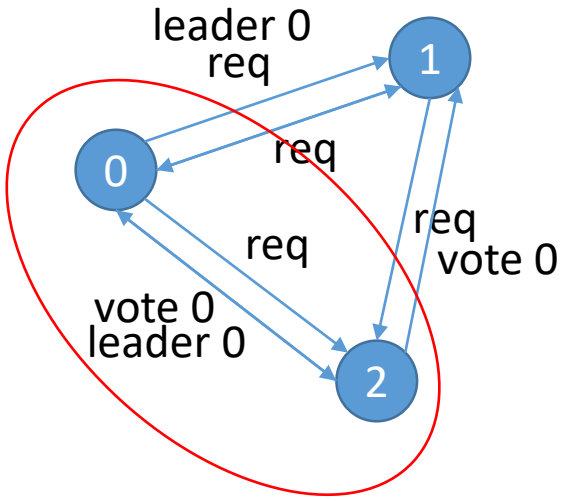
Sound and complete automatic deductive verification

For every class C of finite graphs above

- Axioms for path relation – universally q
- 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 $|\mathrm{votes}(p)| > \frac{|\mathrm{all}|}{2}$ then send leader(p)

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

- Solution: axiomatize cardinalities in first-order logic

$$\forall s, t. majority(s) \wedge majority(t) \rightarrow \exists p.\ \mathrm{member}(p, s) \wedge \mathrm{member}(p, t)$$

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 ∀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

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

Protocol	Model [LOC]	Invariants	Verification time [sec]
Paxos	85	11	2.2
Multi-Paxos	98	12	2.6
Vertical Paxos*	123	18	2.2
Fast Paxos*	117	17	6.2
Flexible Paxos	88	11	2.2
Stoppable Paxos*	132	16	5.4

*first mechanized verification

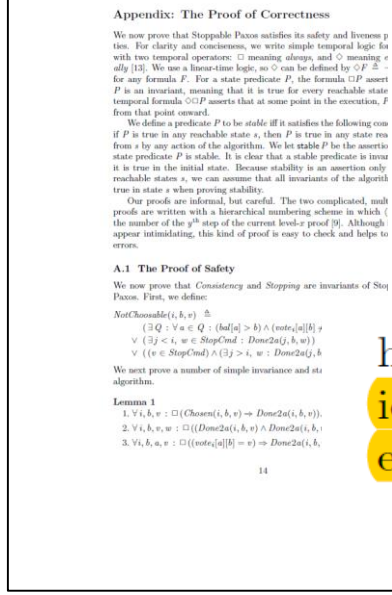
Abstraction and transformation to EPR reusable across all variants!

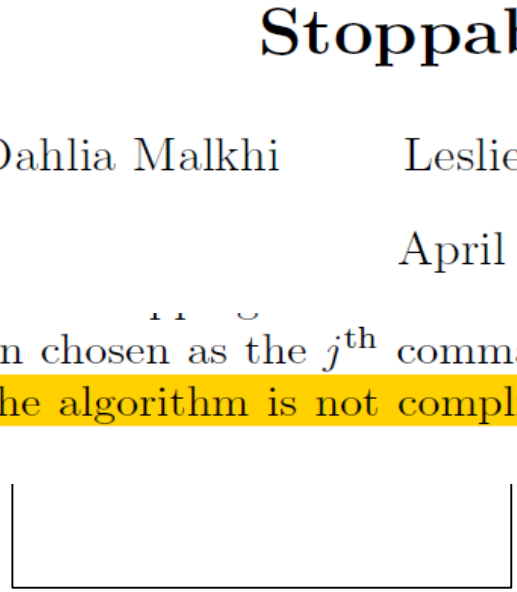
Stoppable Paxos

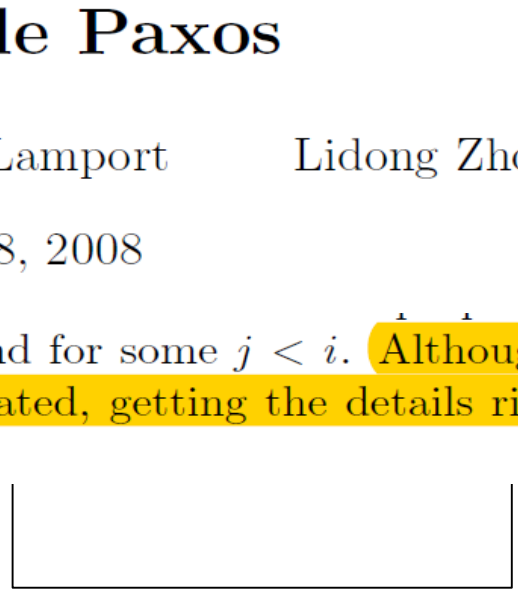
Dahlia Malkhi Leslie Lamport Lidong Zhou

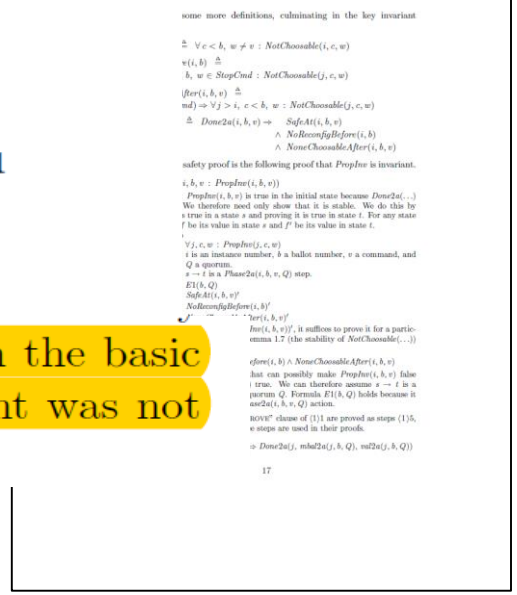
April 28, 2008

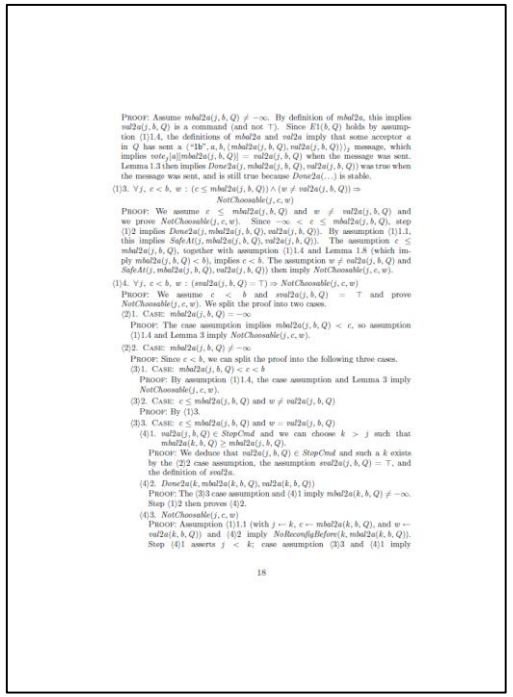
have been chosen as the jth command for some $j < i$. Although the basic idea of the algorithm is not complicated, getting the details right was not easy.

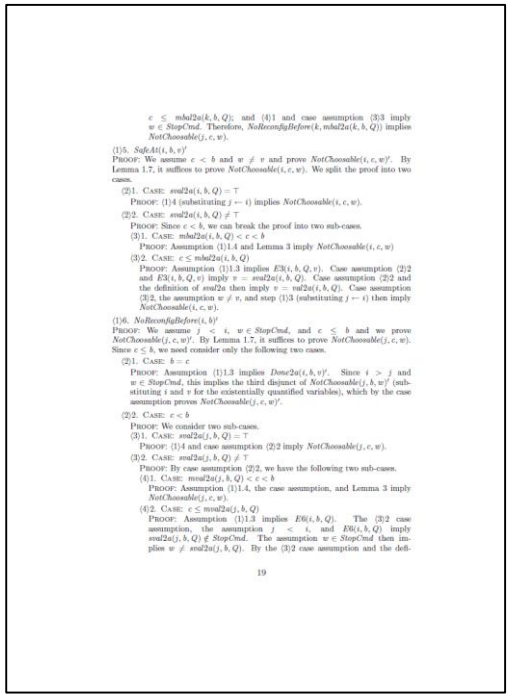


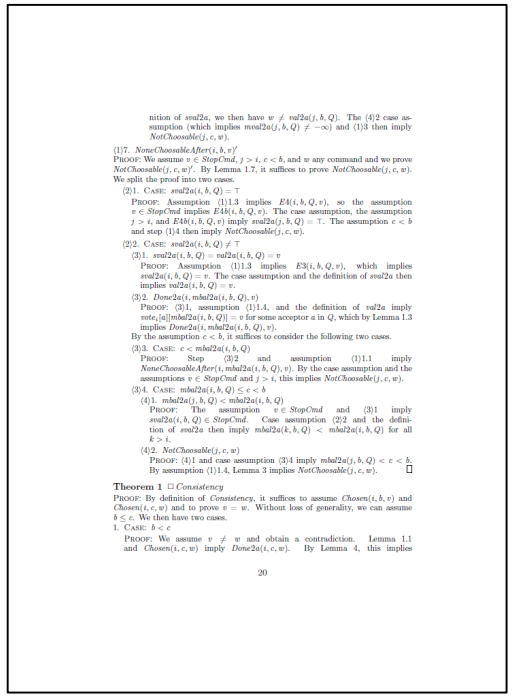


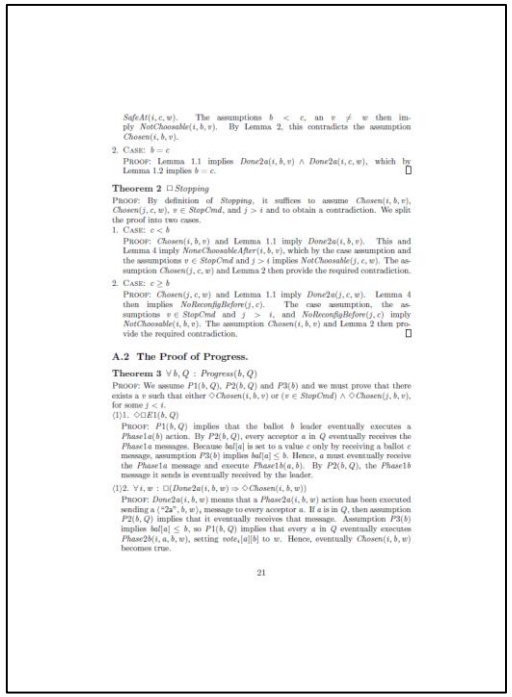












$\langle 1 \rangle 7.$ $NoneChoosableAfter(i, b, v)'$

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.

$\langle 2 \rangle 1.$ CASE: $sval2a(i, b, Q) = \top$

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: $\langle 3 \rangle 1$, assumption $\langle 1 \rangle 1.4$, 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: $mbal2a(i, b, Q) \leq c < b$

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

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 $NotChoosable(j, c, w)$. $\square$

Impact First Order Abstraction

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

pirapira commented on Sep 28, 2017 Owner

This is beautiful, thank you! I'll update README to direct people to your script.

pirapira merged commit `1aaea07` into `pirapira:master` on Sep 28, 2017 Revert

ethereum

Closing the gap

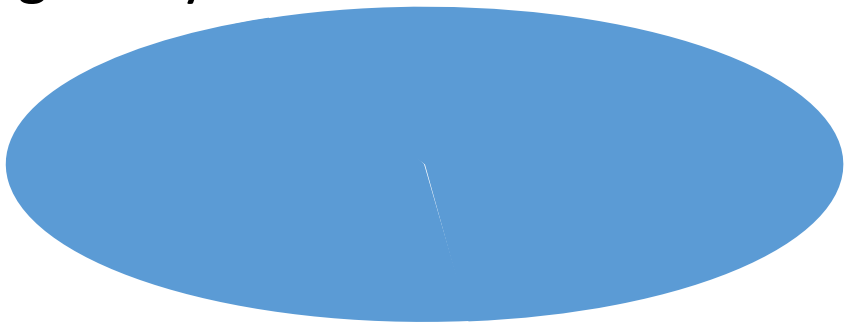
- Reasoning about abstract protocols (designs)
  - User provides axioms expressed in first order logic
  - Not checked by the system
  - Missing axioms can lead to false alarms
- Reasoning about implementations
  - Abstract total order → concreter domain, e.g., integers
  - Abstract sets with majorities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

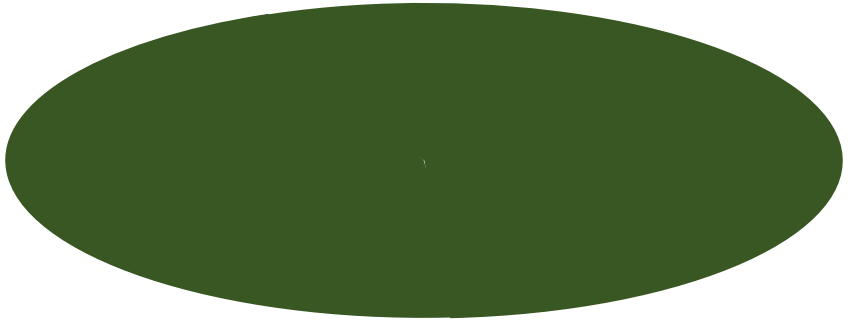
Modularity

Original system

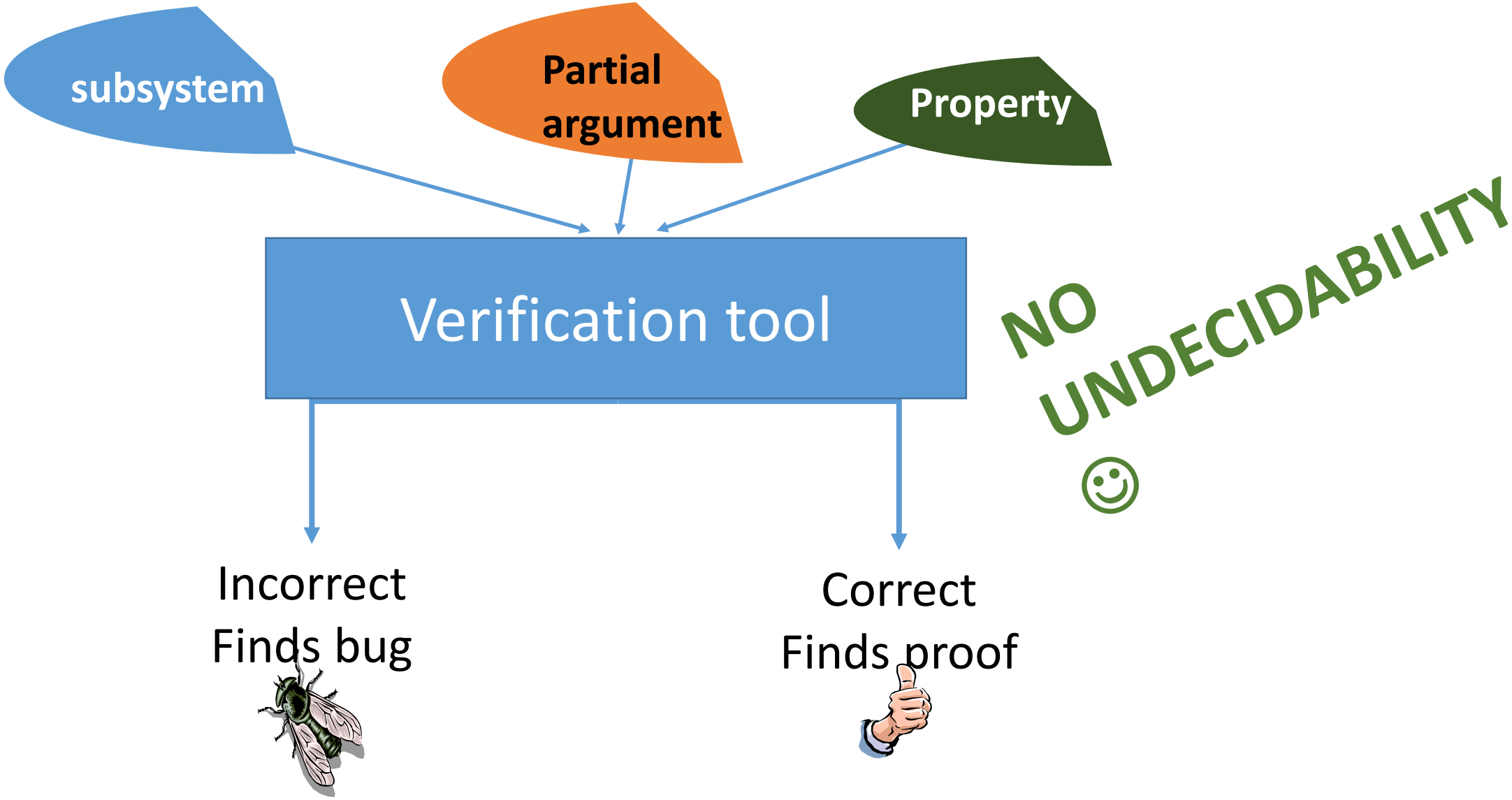


Original inductive argument

Original property



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 ∀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
    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|

$$\forall s, t. |s| + |t| > |\text{all}| \rightarrow \exists p. member(p, s) \wedge member(p, t)$$

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

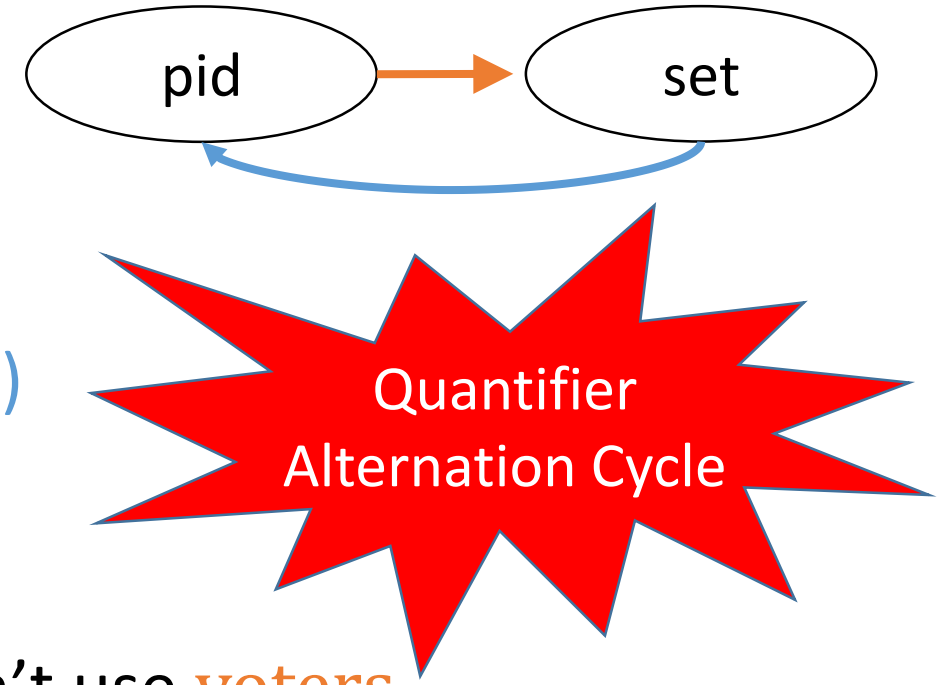
Quantifier alternation cycles

- Protocol state

  voters: pid → set

- Property maj

  $\forall s, t: set. \exists p: pid.$ $majority(s) \wedge majority(t) \Rightarrow member(p, s) \wedge member(p, t)$



- 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 $\forall\, m. \neg \mathrm{voted}(\mathrm{v}, m)$;
  voted(v,n) := true;
}

procedure make_leader(n : pid, s : set) = {
  require $\mathrm{majority}(\mathrm{s})$;
  require $\forall m. \mathrm{member}(m, \mathrm{s}) \rightarrow \mathrm{voted}(m, \mathrm{n})$;
  isleader(n) := true;
  quorum := s;
}

Invariant:

- one leader: $\forall n, m. isleader(n) \wedge isleader(m) \rightarrow n = m$
- voted is a partial function: $\forall$p,n,m. voted(p,n) $\wedge$ voted(p,m)$\rightarrow n$=m
- leader has a quorum: $\forall n, m. isleader(n) \wedge member(m, quorum) \rightarrow voted(m, n)$

Provable in $\mathrm{EPR{+}{+}}$

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);
    }
}
```

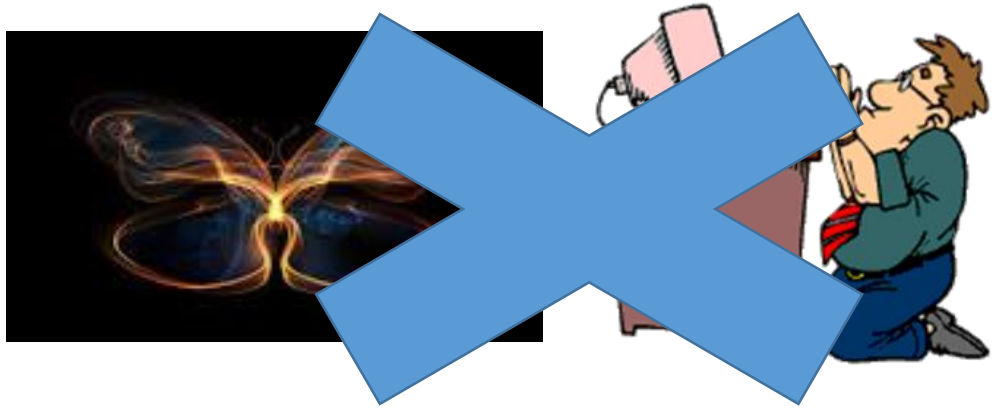
call to abstract model must satisfy precondition

In place of property maj, we use the *one leader* invariant of the abstract model

$$\forall p, n. abs.voted(p, n) \rightarrow already\_voted(p)$$
$$\forall p, n. network.vote(p, n) \leftrightarrow abs.voted(p, n)$$
$$\forall n. leader(n) \leftrightarrow abs.isleader(n)$$
$$\ldots$$

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
RAFT	Coq/Verdi	530	50,000	94
	Ivy	560	200	0.36
MULTIPAXOS	Dafny/IronFleet	3000	12,000	4
	Ivy	330	266	0.8

Verification Effort

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

Why do people hate First Order Logic?

Rants	Ivy
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…	☑	Turing-Complete	Undecidable
SMV	☒	Finite-state	Temporal Properties
TLA+	☒	Turing-Complete	Manual
Coq, Isabelle/HOL	☑	“Turing-Complete”	Manual with tactics
Dafny	☑	Turing-Complete	Undecidable with lemmas
Ivy	☑	Turing-Complete	Decidable(EPR++/FAU)

State of the art in formal verification

Expressiveness

Proof Assistants

Ultimately limited by human

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

Ivy

Decidable deduction
Finite counterexamples
proof/code: ~0.2

Ultimately limited by undecidability

Decidable Models
Model Checking
Static Analysis

Automation