

# Modularity for decidability of deductive verification with applications to distributed systems

#### Mooly Sagiv





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

# Contributors

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

Research



















James R. Wilcox,

Doug Woos





UNIVERSITY of WASHINGTON

# And Also

Anindya Benerjee, Neil Immerman, Shachar Itzhaky, Aleks Nanevsky Aurojit Panda















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

# Virtual Machine

http://www.cs.tau.ac.il/~odedp/ivy-sri18.ova

# 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

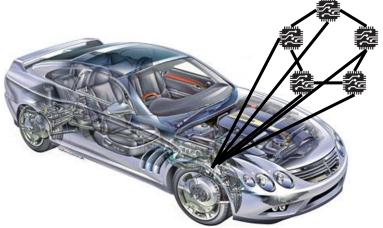
[PLDI'18] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, MS, Sharon Shoham, James R. Wilcox, Doug Woos: Modularity for Decidability of Deductive Verification with Applications to Distributed Systems

# Agenda

- Today
  - Motivation
  - Deductive Verification in Ivy
- Wednesday
  - Decidable logics
  - Case study
    - Reasoning about linked list
    - Modularity and decidability

# 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



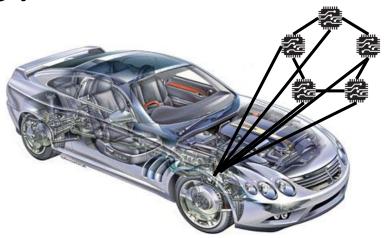


# Why verify distributed protocols?

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

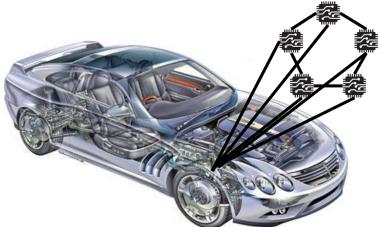
SIGCOMM'01
 Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications
 Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, Member, IEEE
 Attractive features of Chord include its simplicity, provable correctness, and provable performance even in the face of concurrent node arrivals and departures. It continues to func-

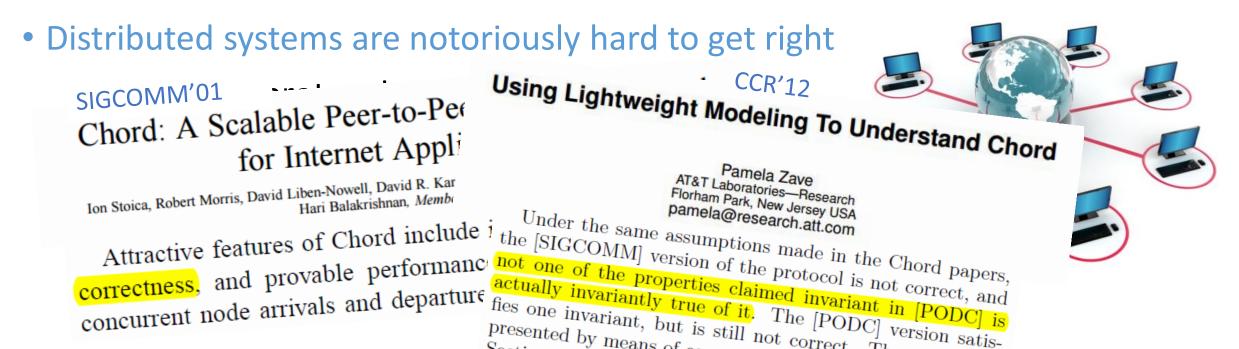




# Why verify distributed protocols?

- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchain





Best Paper Award Zyzzyva: Speculative Byzantine Fault Tolerance SOSP'07 arXiv:1712.01367v1 [cs.DC] 4 Dec 2017 Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong Revisiting Fast Practical Byzantine Fault Tolerance Zvzzyva is a state machine replication protocol based on rotocols: (1) agreement, (2) view change, and (3)Ittai Abraham, Guy Gueta, Dahlia Malkhi ment protocol orders requests for exe-VMware Research we change protocol coordinates CACM'08 ACM Transactions on Computer Systems '09 with: Lorenzo Alvisi (Cornell), Rama Kotla (Amazon), Zyzzyva: Speculative Byzantine Jean-Philippe Martin (Verily) We now proceed to demonstrate that the view-change mechanism in Zyzzyva does not guarantee safety. Fault Tolerance LORENZO ALVISI, MIKE DAHLIN, ALLEN CLEMENT, and EDMUND WONG RAMAKRISHNA KOTLA Microsoft Research, Silicon Valley The University of Texas at Austin

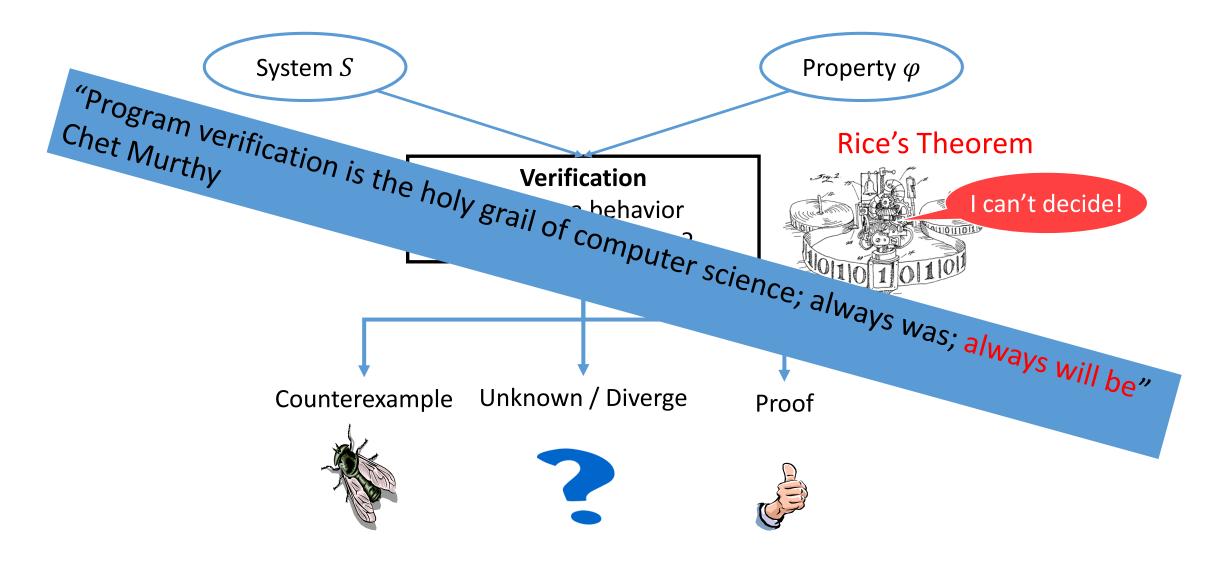
## Proving distributed systems is hard

- Amazon [CACM'15] uses TLA+ for testing protocols, but no proofs
- IronFleet [SOSP'15] verification of Multi-Paxos in Dafny (3.7 person-years)
- Verdi [PLDI'15] verification of Raft in Coq (50,000 lines of proofs)

Our goal: reduce human effort while maintaining flexibility Our approach: decompose verification into decidable problems

[CACM'15] Newcombe et al. How Amazon Web Services Uses Formal Methods
 [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

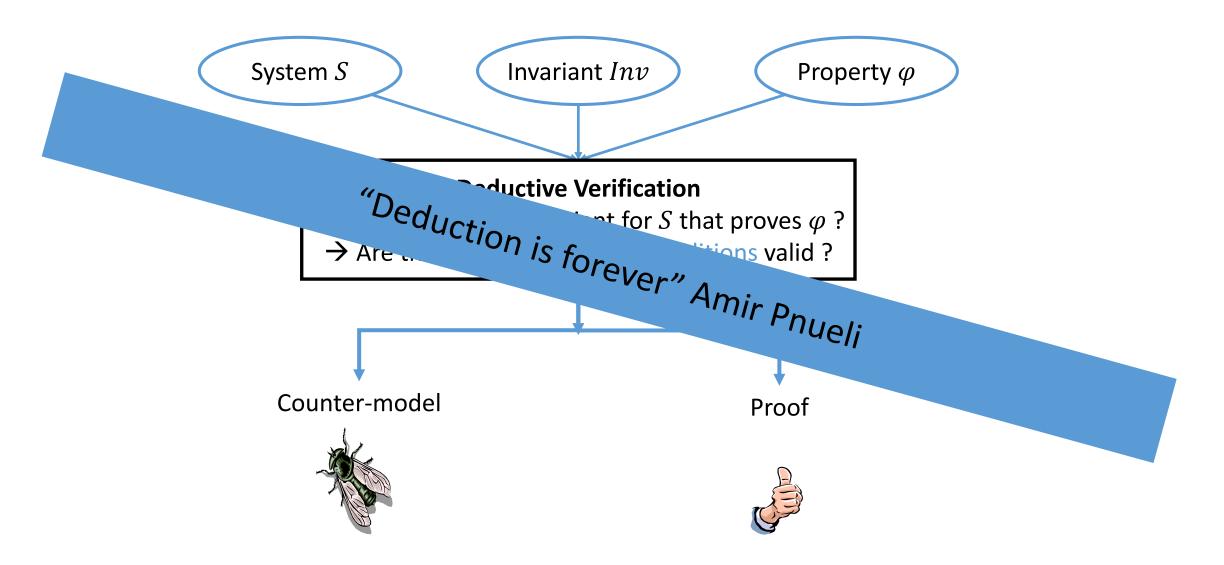
#### Automatic verification of infinite-state systems



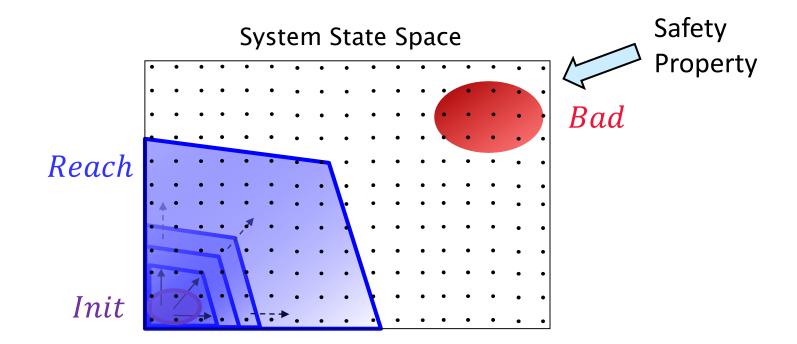
#### Semi-automatic deductive verification



#### Deductive verification

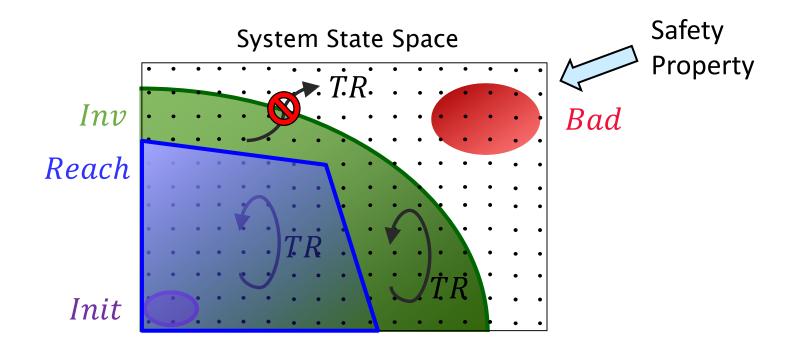


#### Inductive invariants



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

#### Inductive invariants



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

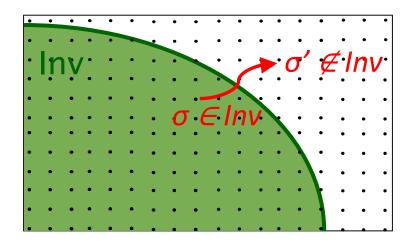
System S is safe iff there exists an **inductive invariant** Inv :

*Init*  $\subseteq$  *Inv* (Initiation) if  $\sigma \in$  *Inv* and  $\sigma \rightarrow \sigma'$  then  $\sigma' \in$  *Inv* (Consecution) *Inv*  $\cap$  *Bad* =  $\emptyset$  (Safety)

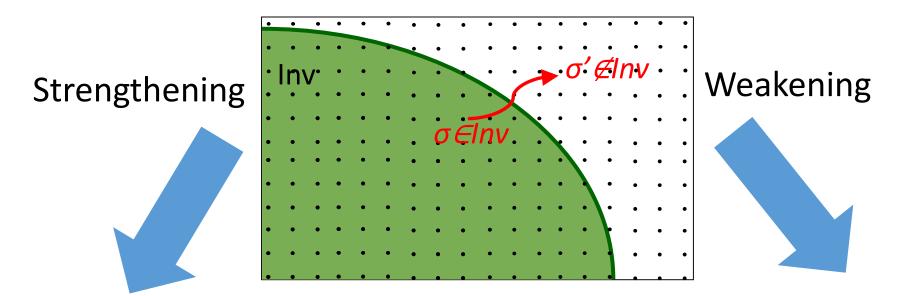
translated to VC's

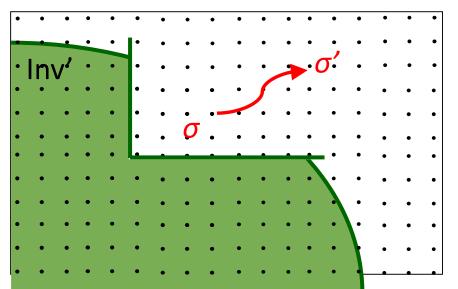
# Counterexample To Induction (CTI)

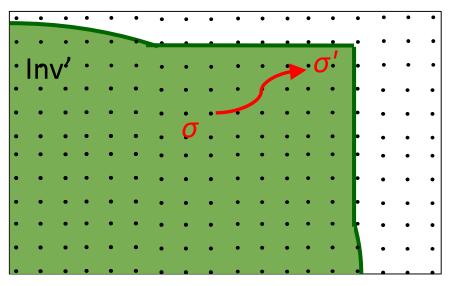
- States σ,σ' are a CTI of Inv if:
- $\sigma \in Inv$
- <mark>σ'</mark> ∉ Inv
- $\sigma \rightarrow \sigma'$
- A CTI may indicate:
  - A bug in the system
  - A bug in the safety property
  - A bug in the inductive invariant
    - Too weak
    - Too strong



## Strengthening & weakening from CTI

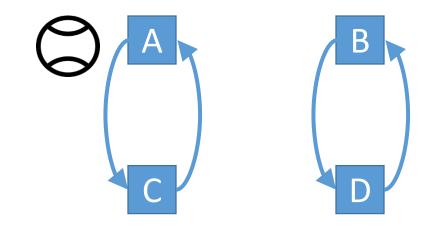






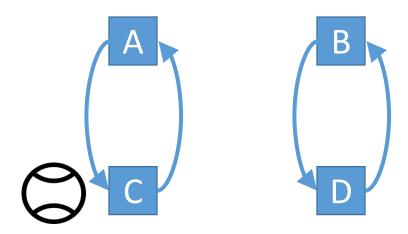
# Induction on a ball game

- Four players pass a ball:
  - A will pass to C
  - B will pas to D
  - C will pass to A
  - D will pass to B
- The ball starts at player A
- Can the ball get to D?



# Induction on a ball game

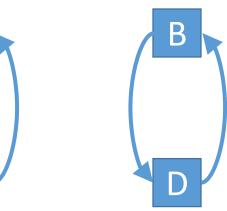
- Four players pass a ball:
  - A will pass to C
  - B will pas to D
  - C will pass to A
  - D will pass to B
- The ball starts at player A
- Can the ball get to D?



## Formalizing with induction

• 
$$x_0 = A$$
  
•  $x_{n+1} = \begin{cases} C \ if \ x_n = A \\ D \ if \ x_n = B \\ A \ if \ x_n = C \\ B \ if \ x_n = D \end{cases}$ 

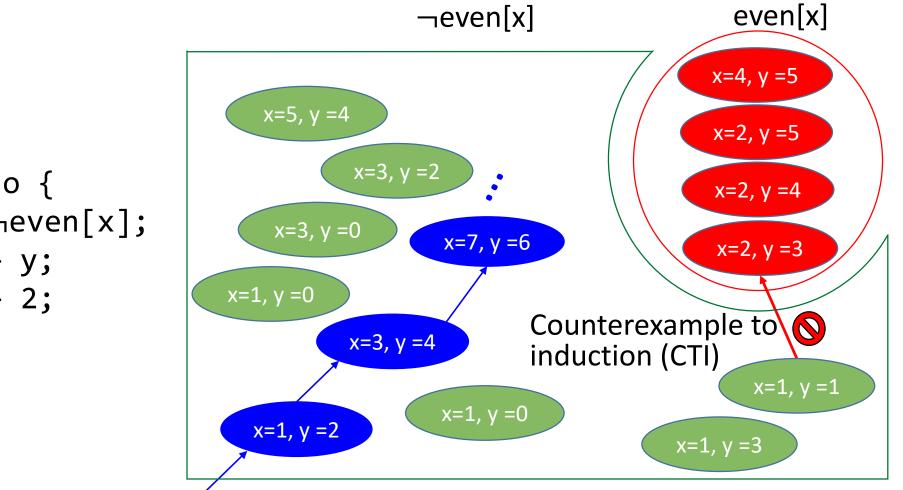
- Prove by induction  $\forall n. x_n \neq D$ 
  - $x_0 \neq D$  ?
  - $x_m \neq D \Rightarrow x_{m+1} \neq D$  ?



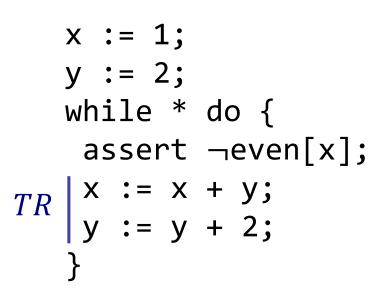
## Formalizing with induction

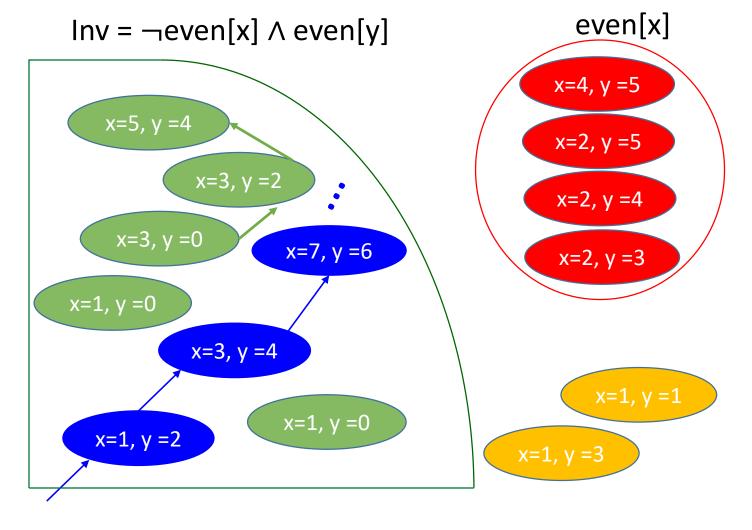
- Prove a stronger claim by induction  $\forall n. x_n \neq B \land x_n \neq D$ 
  - $x_0 \neq B \land x_0 \neq D$
  - $x_m \neq B \land x_m \neq D \Rightarrow x_{m+1} \neq B \land x_{m+1} \neq D$

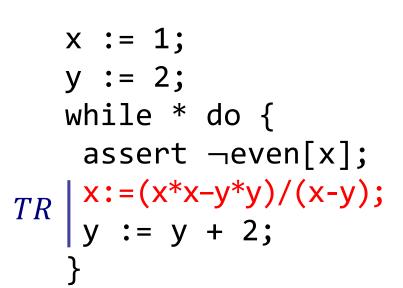
even[x] x=4, y =5 x := 1; x=2, y =5 y := 2; while \* do { x=2, y =4 assert ¬even[x]; x=3, y =0 x=7, y =6 x=2, y =3 x := x + y; y := y + 2; TRx=1, y =0 x=3, y =4 x=1, y =0 x=1, y =2

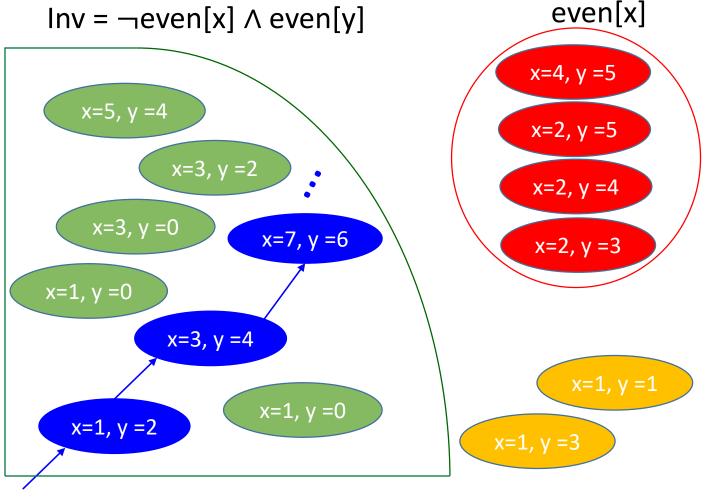


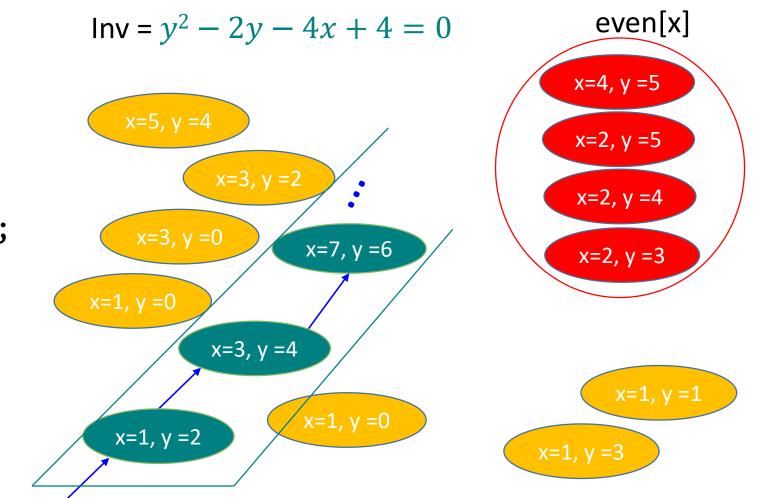
x := 1; y := 2; while \* do { assert ¬even[x]; TR | x := x + y; y := y + 2; }



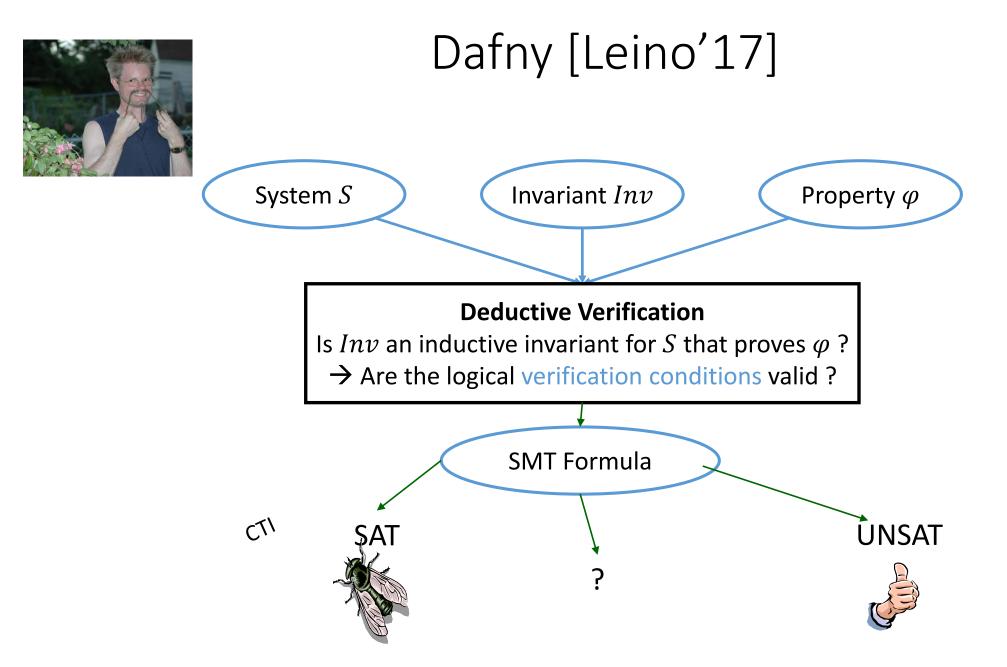






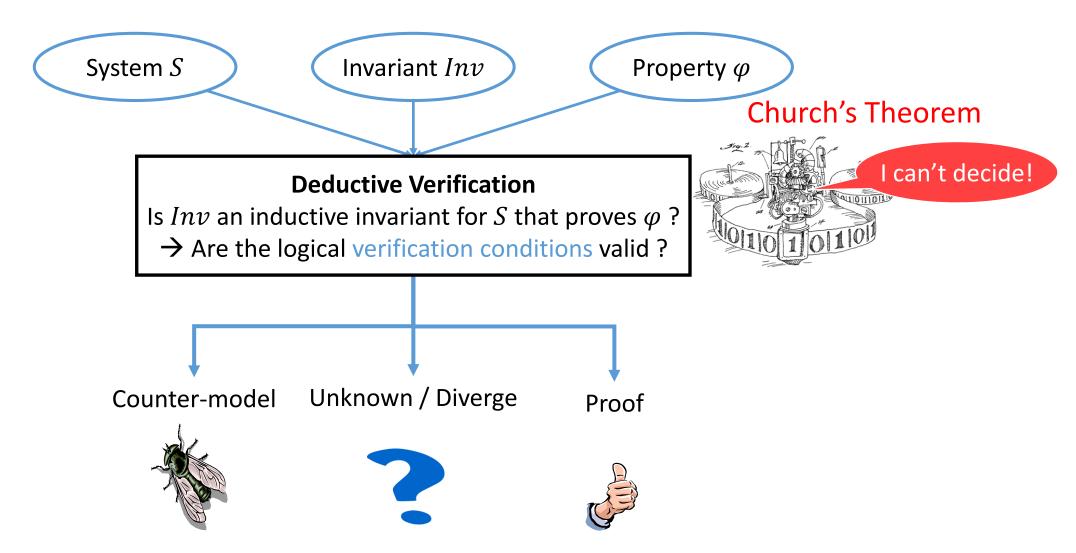


x := 1; y := 2; while \* do { assert ¬even[x];  $TR \begin{vmatrix} x & := x + y; \\ y & := y + 2; \end{vmatrix}$ 



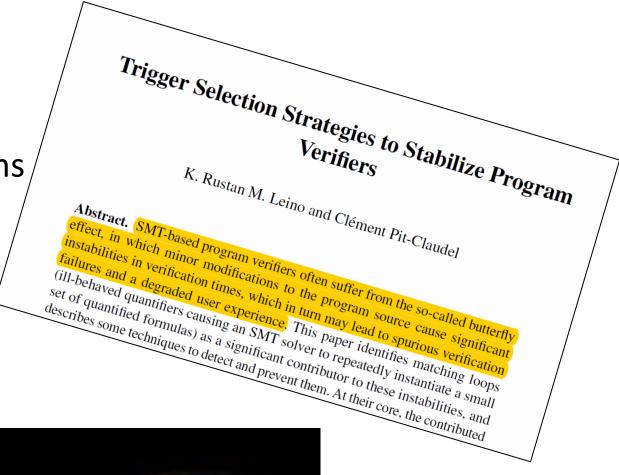
K. Rustan M. Leino: Accessible Software Verification with Dafny. IEEE Software 34(6): 94-97 (2017)

#### Deductive verification



# Effects of undecidability

- The verifier may fail on tiny programs
- No explanation when tactics fails
  - Counterproofs
- The butterfly effect
- Observed in the IronFleet Project





# Challenges in deductive verification

- 1. Formal specification: formalizing infinite-state systems and their properties
- 2. Deduction: checking inductiveness
  - Undecidability of implication checking
    - Unbounded state (threads, messages), arithmetic, quantifier alternation
- 3. Inference: finding inductive invariants (Inv)
  - Hard to specify
  - Hard to maintain
  - Hard to infer
    - Undecidable even when deduction is decidable

# State of the art in formal verification





Proof Assistants

Ultimately limited by human



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



Decidable deduction Finite counterexamples proof/code: ~0.2

#### Ultimately limited by undecidability

Decidable Models Model Checking Static Analysis

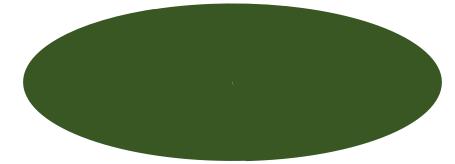
Automation

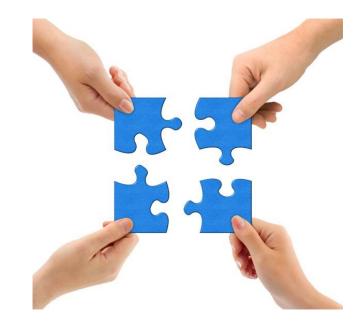
# Modularity

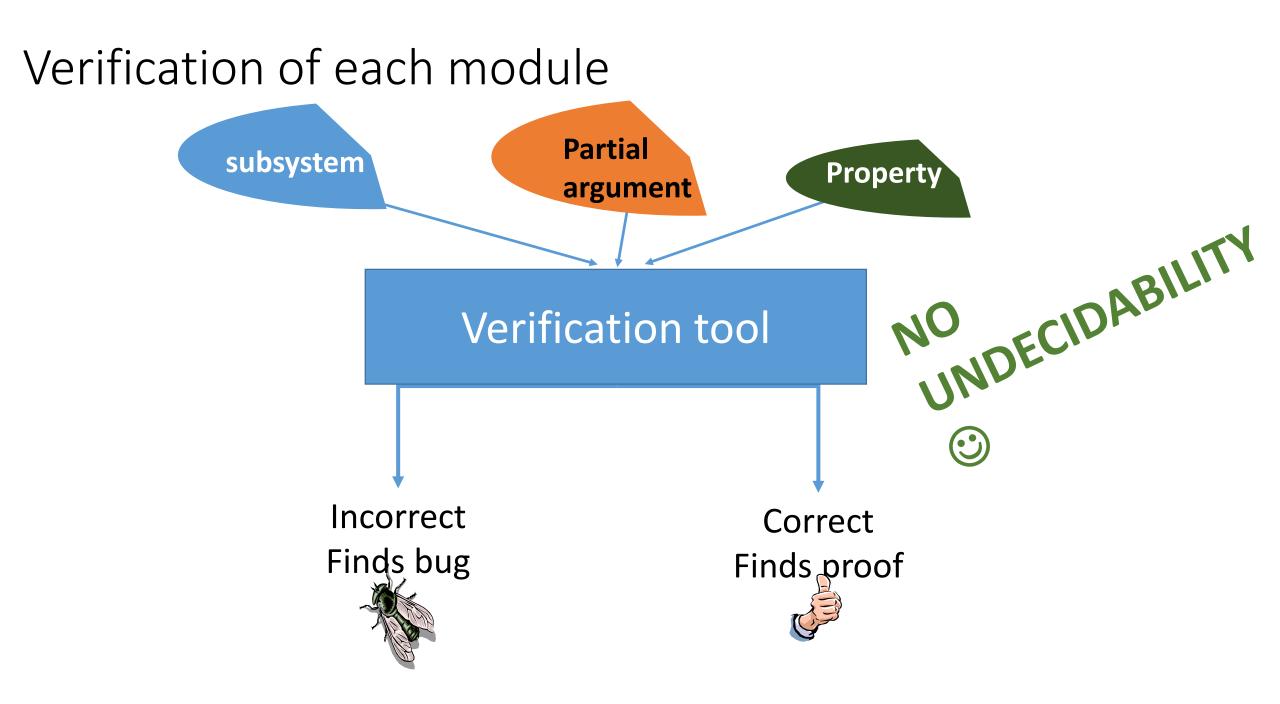
Original system

#### Original inductive argument

#### Original property







# lvy's principles

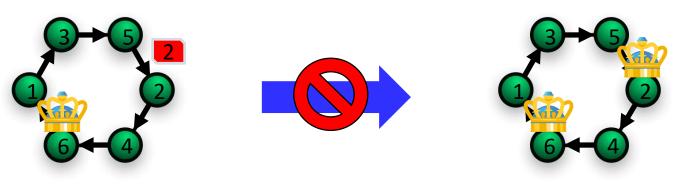
- Modularity
  - The user breaks the verification system into small problems expressed in decidable logics
  - The system explores circular assume/guarantee reasoning to prove correctness
- Inductive invariants and transition systems are expressed in decidable logics
  - Turing complete imperative programs over unbounded relations
  - Allows quantifiers to reason about unbounded sets
    - But no arbitrary quantifier alternations and theories
  - Checking inductiveness is decidable
  - Display CTIs as graphs (similar to Alloy)

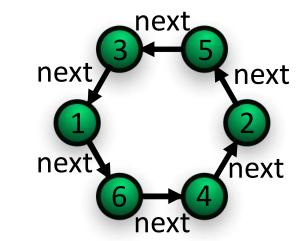
### Languages and verification

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

# 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

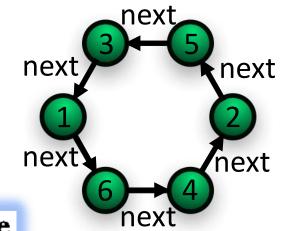




[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

# 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 ic *Proposition:* This algorithm detects one and only one A not highest number.
- Theorem Argument: By the circular nature of the configuration and the consistent direction of messages, any message must meet all other processes before it comes back to its initiator. Only one message, that with the highest number, will not encounter a higher number on its way around. Thus, the only process getting its own message back is the one with the highest number.



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

## Leader election protocol – first-order logic

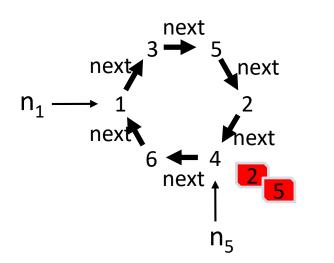
• ≤ (ID, ID) – total order on node id's

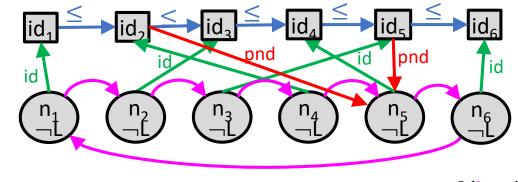
protocol state

- **btw** (Node, Node, Node) the ring topology
- id: Node  $\rightarrow$  ID relate a node to its unique id
- **pending**(ID, Node) pending messages
- leader(Node) leader(n) means n is the leader



first-order structure





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

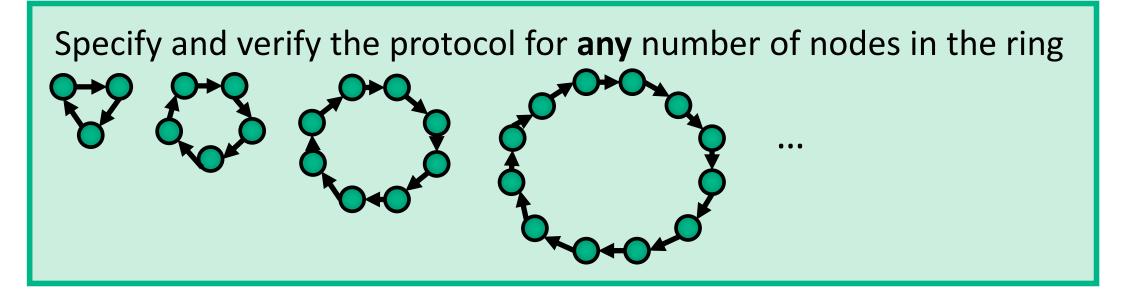
## Leader election protocol – first-order logic

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

protocol state

- Axiomatized in first-order logic

first-order structure



## Leader election protocol – first-order logic

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

action send(n: Node) = {
 "s := next(n)";
 pending(id(n),s) := true
}

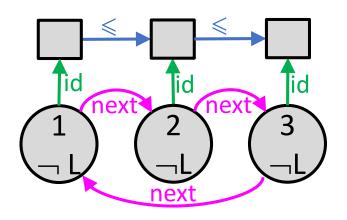
```
action receive(n: Node, m: ID) = {
  requires pending(m, n);
  if id(n) = m then
    // found Leader
    leader(n) := true
  else if id(n) ≤ m then
    // pass message
    "s := next(n)";
    pending(m, s) := true
}
```

TR (send):

 $\exists n,s: Node. "s = next(n)" \land \forall x:ID,y:Node. pending'(x,y)↔ (pending(x,y)∨(x=id(n)∧y=s))$ 

Bad:

```
assert I0 = \forall x,y: Node. leader(x) \land leader(y) \Rightarrow x = y
```

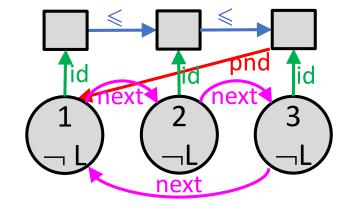


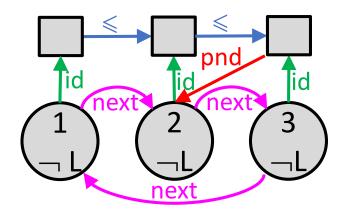


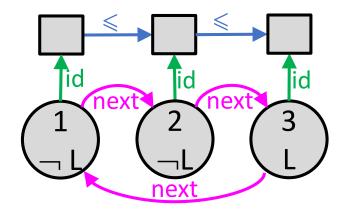


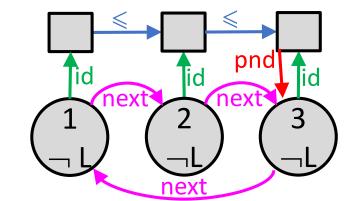
rcv(2, id(3))

rcv(3, id(3))









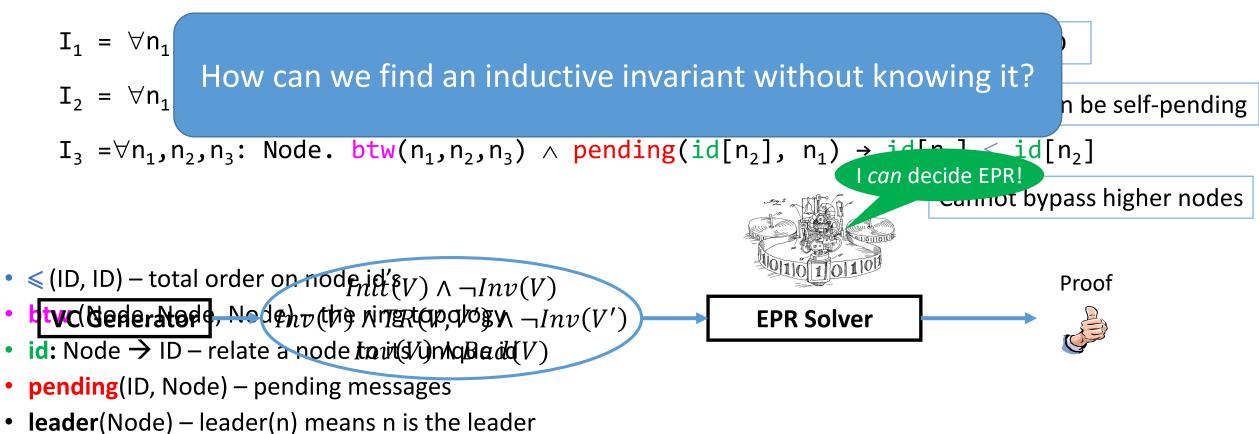
#### Leader election protocol – inductive invariant

Safety property: I<sub>a</sub>

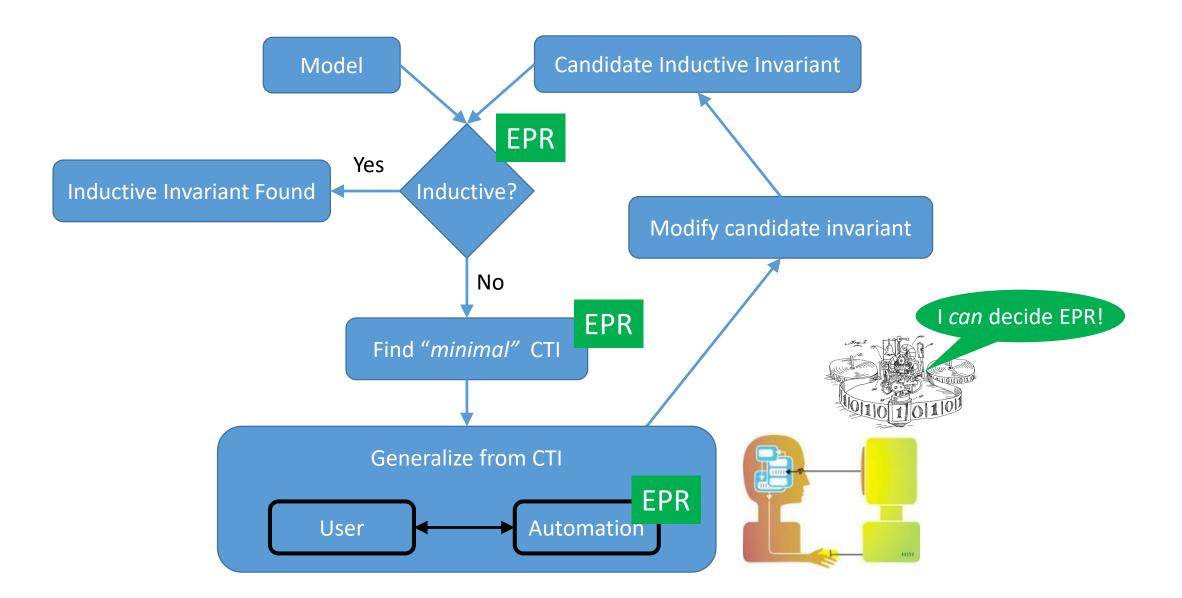
 $I_{0} = \forall x, y: Node. leader(x) \land leader(y) \rightarrow x = y$ 

•

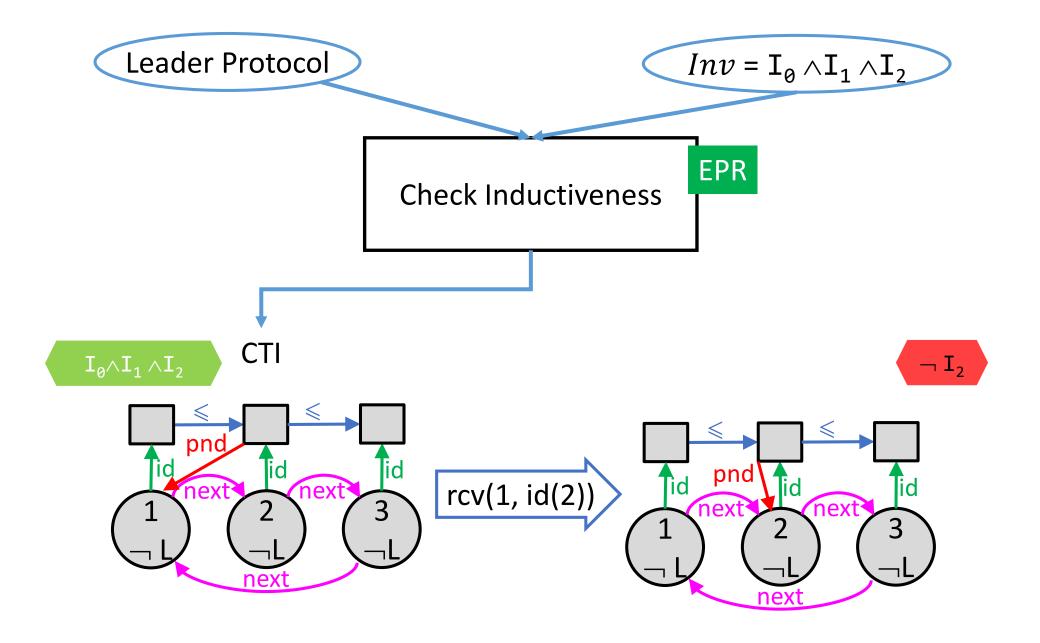
**Inductive invariant:** Inv =  $I_0 \wedge I_1 \wedge I_2 \wedge I_3$ 



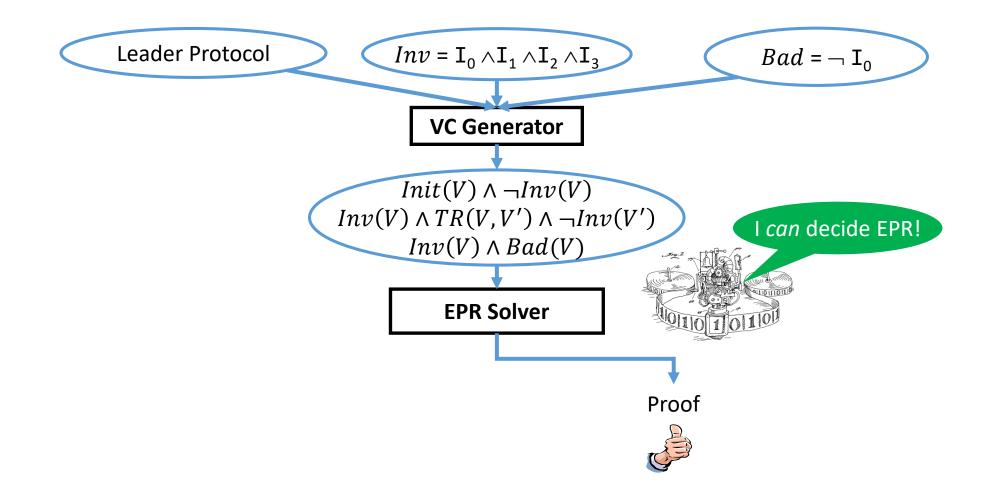
#### Interactive invariant inference [PLDI'16]



#### Ivy: check inductiveness

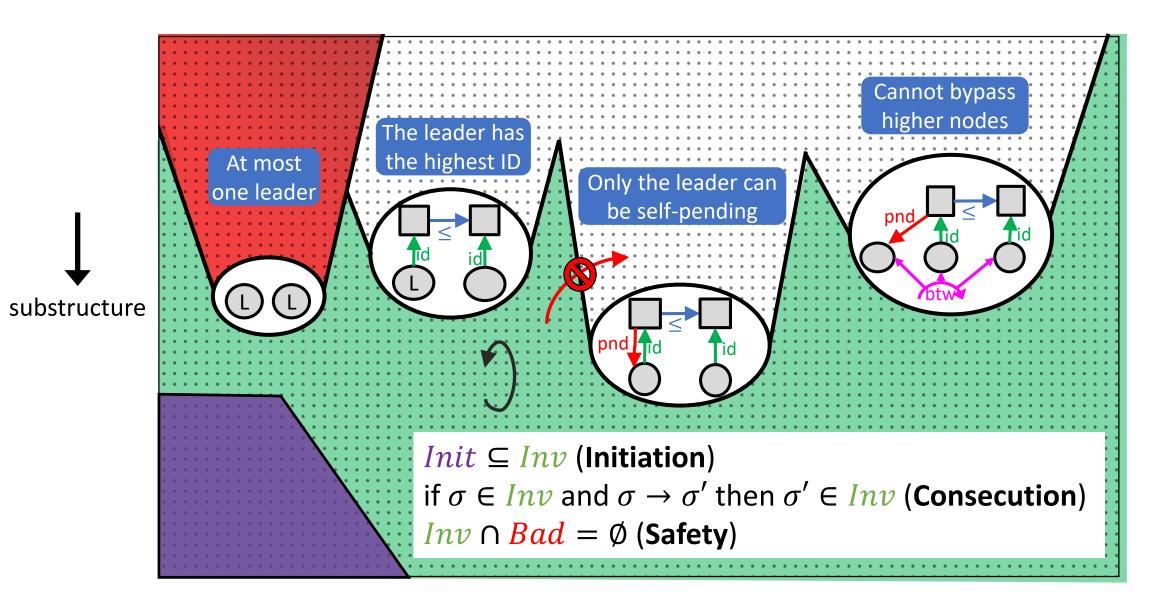


#### Ivy: check inductiveness



 $I_0 \wedge I_1 \wedge I_2 \wedge I_3$  is an inductive invariant for the leader protocol, proving its safety

#### $\forall^*$ invariant – excluded substructures



## Principle: first-order abstractions/modularity

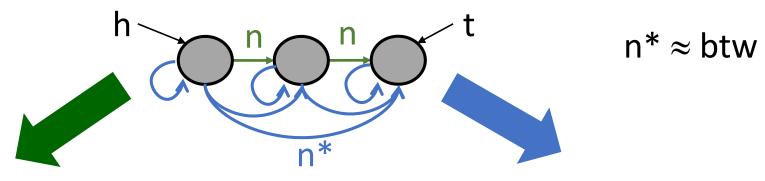
Concept	Intention	First-order abstraction
Node ID's	Integers	function id: Node $\rightarrow$ ID relation $\leq$ (ID, ID) axiom $\forall x:ID. x \leq x$ reflexive axiom $\forall x,y,z:ID. x \leq y \land y \leq z \rightarrow x \leq z$ transitive axiom $\forall x,y:ID. x \leq y \land y \leq x \rightarrow x=y$ anti-symmetric axiom $\forall x,y:ID. x \leq y \lor y \leq x$ total axiom $\forall x, y: Node. id(x) = id(y) \rightarrow x=y$ injective
Ring Topology	Next edges + Transitive closure	relation btw (Node, Node, Node)axiom $\forall x, y, z:$ Node. btw(x, y, z) $\rightarrow$ btw(y, z, x) circularaxiom $\forall x, y, z, w:$ Node. btw(w, x, y) $\wedge$ btw(w, y, z) $\rightarrow$ btw(w, x, z) transitiveaxiom $\forall x, y, w:$ Node. btw(w, x, y) $\rightarrow \neg$ btw(w, y, x) anti-symmetricaxiom $\forall x, y, w:$ Node. $\neq$ (w, x, y) $\rightarrow$ btw(w, x, y) $\vee$ btw(w, y, x) totalmacro "next(a)=b" $\equiv \forall x:$ Node. x=a $\lor x=b \lor$ btw(a,b,x) edges

Challenge: How to use restricted first-order logic to verify interesting systems?

- Expressing transitive closure
  - Linked lists
  - Ring protocols
- Expressing sets and cardinalities
  - Paxos, Multi-Paxos
  - Reconfiguration
  - Byzantine Fault Tolerance
- Liveness and temporal properties

# Key idea: representing deterministic paths

[Itzhaky SIGPLAN Dissertation Award 2016]

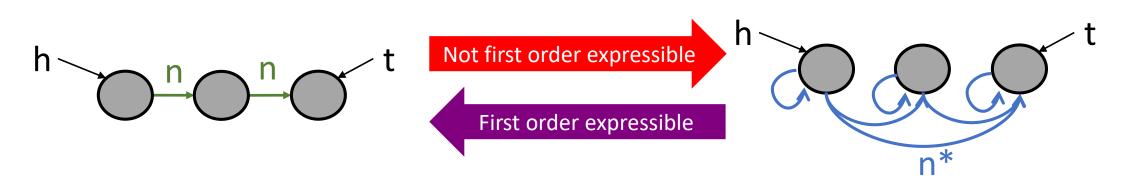


#### Alternative 1: maintain n

- n<sup>\*</sup> defined by transitive closure of n
- not definable in first-order logic

#### Alternative 2: maintain n<sup>\*</sup>

- n defined by transitive reduction of n<sup>\*</sup>
- Unique due to outdegree  $\leq 1$
- Definable in first order logic (for roots)
  - $n^+(a,b) \equiv n^*(a, b) \land a \neq b$
  - $n(a, b) \equiv n^+(a,b) \land \forall z: n^+(a, z) \rightarrow n^*(b, z)$



Challenge: How to use restricted first-order logic to verify interesting systems?

- Expressing transitive closure
  - Linked lists
  - Ring protocols
- Expressing sets and cardinalities
  - Paxos and its variants
  - Byzantine Fault Tolerance
  - Reconfiguration
- Liveness and temporal properties

#### 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
  - VMware NSX
  - Amazon AWS
  - Many more...

## Challenge: sets and cardinalities in FOL

- Consensus algorithms use set cardinalities
  - Wait for messages from more than N / 2 nodes
- Insight: set cardinalities are used to get a simple effect

Can be modeled in first-order logic!

Solution: axiomatize quorums in first-order logic
 sort Quorum

relation member (Node, Quorum)

- set membership (2<sup>nd</sup>-order logic in first-order)

**axiom**  $\forall q_1, q_2$ : Quorum.  $\exists n$ : Node. member(n,  $q_1$ )  $\land$  member(n,  $q_2$ )

```
action propose(r:Round) {
 require ">N/2 join msg's"
 ...
action propose(r:Round) {
 require \exists q. \forall n. member(n,q) \rightarrow
   ∃r',v'.join msg(n,r,r',v')
 ...
```

#### Principle: first-order abstractions

Concept	Intention	First-order abstraction
Quorums	Majority sets	<b>relation</b> member (Node, Quorum) <b>axiom</b> $\forall q_1, q_2$ :Quorum $\exists n$ :Node. member(n, q_1) $\land$ member(n, q_2)
Rounds	Natural numbers	<b>relation</b> $\leq$ (Round, Round) <b>axiom</b> $\forall$ x:Round. $x \leq x$ <i>reflexive</i> <b>axiom</b> $\forall$ x,y,z:Round. $x \leq y \land y \leq z \rightarrow x \leq z$ <i>transitive</i> <b>axiom</b> $\forall$ x,y:Round. $x \leq y \land y \leq x \rightarrow x = y$ <i>anti-symmetric</i> <b>axiom</b> $\forall$ x,y:Round. $x \leq y \lor y \leq x$ <i>total</i>
Messages	Network with: dropping duplication reordering	<pre>relation start_msg(Round) relation join_msg(Node, Round, Round, Value) relation propose_msg(Round, Value) relation vote_msg(Node, Round, Value)</pre>

#### Paxos in first-order logic

	21	41 # voting, and the corresponding vote.
1 sort node, quorum, round, value	22 action start_round(r : round) {	42 # v is arbitrary if the nodes reported not voting.
2	23 assume $r \neq \bot$	43 local maxr, $v := \max \{(r', v') \mid \exists n. member(n, q)\}$
3 relation ≤ : round, round	<pre>24 start_round_msg(r) := true</pre>	44 $\wedge join\_ack\_msg(n, r, r', v') \wedge r' \neq \bot$
4 axiom total_order(≤)	25 }	45 propose_msg(r, v) := true # propose value v
5 constant ⊥ : round	<pre>26 action join_round(n : node, r : round) {</pre>	46 }
6	27 assume $r \neq \bot$	47 action vore(n : node, r : round, v : value) {
7 relation member : node, quorum	28 assume start_round_msg(r)	48 assume $r \neq \perp$
8 <b>axiom</b> $\forall q_1, q_2$ : quorum. $\exists n$ : node. member $(n, q_1) \land$ member $(n, q_2)$	29 assume $\neg \exists r', r'', v. r' > r \land join_ack_msg(n, r', r'', v)$	49 assume propose_msg(r, v)
9	30 # find maximal round in which n voted, and the corresponding vote.	50 assume $\neg \exists r', r'', v, r' > r \land join_ack_msg(n, r', r'', v)$
10 relation start_round_msg : round	31 # maxr = ⊥ and v is arbitrary when n never voted.	51 vote_msg(n, r, v) := true
11 relation join_ack_msg : node, round, round, value	32 local maxr, $v := \max \{(r', v') \mid vote_msg(n, r', v') \land r' < r\}$	52 }
12 relation propose_msg : round, value	33 join_ack_msg(n, r, maxr, v) := true	53 action LEARN(n : node, r : round, v : value, q : quorum) {
13 relation vote_msg : node, round, value	34 }	54 assume $r \neq \perp$
14 relation decision : node, round, value	35 action PROPOSE(r : round, q : quorum) {	55 # 2b from quorum q
15	36 assume $r \neq \bot$	56 assume $\forall n. member(n, q) \rightarrow vote_msg(n, r, v)$
16 init ∀r. ¬start_round_msg(r)	37 assume $\forall v. \neg propose_msg(r, v)$	57 decision(n, r, v) := true
17 init $\forall n, r_1, r_2, v. \neg Join_ack_msg(n, r_1, r_2, v)$	38 # 1b from quorum q	58 }
18 init $\forall r, v. \neg propose_msg(r, v)$	39 <b>assume</b> $\forall n$ . member $(n, q) \rightarrow \exists r', \upsilon$ . join_ack_msg $(n, r, r', \upsilon)$	
19 init $\forall n, r, v. \neg vote\_msg(n, r, v)$	40 # find the maximal round in which a node in the quorum reported	
20 init $\forall n, r, v, \neg decision(n, r, v)$		

 $\forall n_1, n_2 : \text{node}, r_1, r_2 : \text{round}, v_1, v_2 : \text{value}. \ decision(n_1, r_1, v_1) \land decision(n_2, r_2, v_2) \rightarrow v_1 = v_2$  $\forall r : round, v_1, v_2 : value. propose msg(r, v_1) \land propose msg(r, v_2) \rightarrow v_1 = v_2$  $\forall n : \text{node}, r : \text{round}, v : \text{value}. vote\_msg(n, r, v) \rightarrow propose\_msg(r, v)$  $\forall r : \text{round}, v : \text{value}.(\exists n : \text{node}. decision(n, r, v)) \rightarrow \exists q : \text{quorum}. \forall n : \text{node}. member(n, q) \rightarrow vote\_msg(n, r, v)$  $\forall n : \text{node}, r, r' : \text{round}, v, v' : \text{value}. join_ack_msg(n, r, \perp, v) \land r' < r \rightarrow \neg vote_msg(n, r', v')$  $\forall n : \text{node}, r, r' : \text{round}, v : \text{value}. \text{ join ack } msg(n, r, r', v) \land r' \neq \bot \rightarrow r' < r \land \text{vote } msg(n, r', v)$  $\forall n : \text{node}, r, r', r'' : \text{round}, v, v' : \text{value.} join\_ack\_msg(n, r, r', v) \land r' \neq \bot \land r' < r'' < r \rightarrow \neg vote\_msg(n, r'', v')$  $\forall n : \text{node}, v : \text{value}, \neg vote\_msg(n, \bot, v)$  $\forall r_1, r_2 : \text{round}, v_1, v_2 : \text{value}, q : \text{quorum}. \ propose\_msg(r_2, v_2) \land r_1 < r_2 \land v_1 \neq v_2 \rightarrow v_2 \rightarrow v_1 \neq v_2 \rightarrow v_2 \rightarrow v_2 \neq v_2 \rightarrow v_2 \rightarrow v_2 \neq v_2 \rightarrow v_2 \neq v_2 \rightarrow v_2$  $\exists n : \text{node}, r', r'' : \text{round}, v : \text{value}. member(n,q) \land \neg vote_msg(n,r_1,v_1) \land r' > r_1 \land join\_ack\_msg(n,r',r'',v)$ 

VC's in first-order logic

## Quantifier alternation cycles

• Axiom

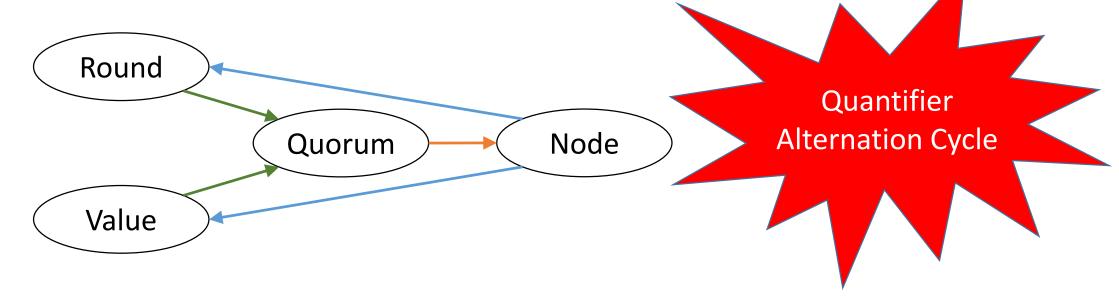
 $\forall q_1, q_2$ : Quorum.  $\exists n$ : Node. member(n,  $q_1$ )  $\land$  member(n,  $q_2$ )

Propose action precondition

 $\exists q:Quorum. \forall n:Node. member(n,q) \rightarrow \exists r':Round,v':Value. join_msg(n,r,r',v')$ 

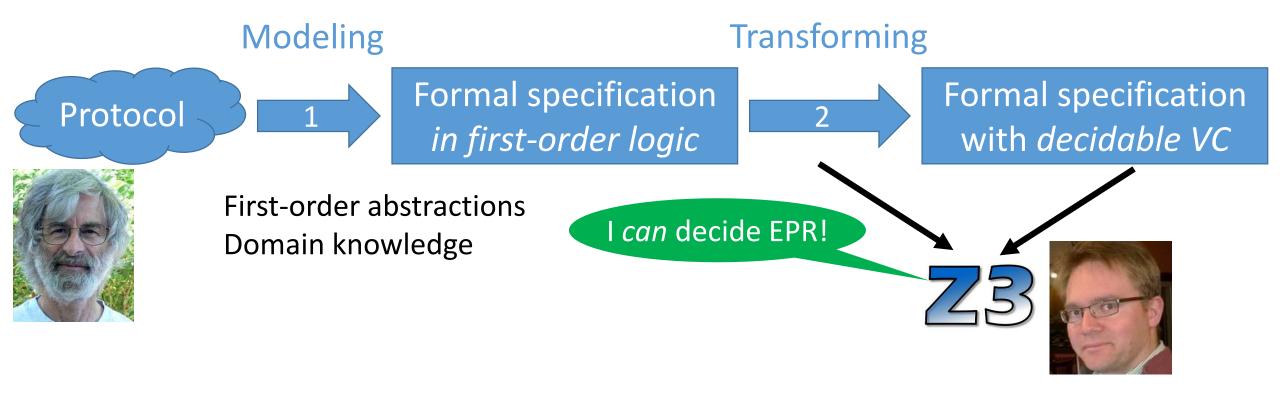
• Inductive invariant

 $\forall r: Round, v: Value. decision(r,v) \rightarrow \exists q: Quorum. \forall n: Node. member(n,q) \rightarrow vote msg(n,r,v)$ 



## Paxos made EPR [OOPSLA'17]

Methodology for decidable verification of infinite-state systems



#### Inductive invariant of Paxos

# safety property

```
conjecture decision(N1,R1,V1) & decision(N2,R2,V2) -> V1 = V2
```

# proposals are unique per round

```
conjecture proposal(R,V1) & proposal(R,V2) -> V1 = V2
```

# only vote for proposed values

```
conjecture vote(N,R,V) -> proposal(R,V)
```

```
# decisions come from quorums of votes:
```

conjecture 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

```
conjecture one_b_max_vote(N,R2,none,V1) & ~le(R2,R1) -> ~vote(N,R1,V2)
conjecture one_b_max_vote(N,R,RM,V) & RM ~= none -> ~le(R,RM) & vote(N,RM,V)
conjecture 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
conjecture ~le(R2,R1) & proposal(R2,V2) & V1 ~~V2 ~> ovists N = member(N, O) & left = nod(N, R1, V2)
```

conjecture ~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

conjecture one\_b(N,R2) & ~le(R2,R1) -> left\_rnd(N,R1)

Protocol	Model	Invariant	EPR	RW	
riotocor	[LOC]	[Conjectures]	μ	σ	[sec]
Paxos	85	11	1.0	0.1	1.2
Multi-Paxos	98	12	1.2	0.1	1.4
Vertical Paxos*	123	18	2.2	0.2	-
Fast Paxos*	117	17	4.7	1.6	1.5
Flexible Paxos	88	11	1.0	0	1.2
Stoppable Paxos*	132	16	3.8	0.9	1.6

Protocol	Model [LOC]	Invariant [Conjectures]	EPR µ	RW [sec]	
Paxos	85	11	ра 1.0	σ 0.1	1.2
Multi-Paxos	98	12	1.2	0.1	1.4
Vertical Paxos*	123	18	2.2	0.2	-
Fast Paxos*	117	17	4.7	1.6	1.5
Flexible Paxos	88	11	1.0	0	1.2
Stoppable Paxos*	132	16	3.8	0.9	1.6

Proof / code ratio: IronFleet: ~4 Verdi: ~10 Ivy: ~0.2

Protocol	Model [LOC]	Invariant [Conjectures]	EPR µ	[sec] σ	RW [sec]
Paxos	85	11	1.0	0.1	1.2
Multi-Paxos	98	12	1.2	0.1	1.4
Vertical Paxos*	123	18	2.2	0.2	-
Fast Paxos*	117	17	4.7	1.6	1.5
Flexible Paxos	88	11	1.0	0	1.2
Stoppable Paxos*	132	16	3.8	0.9	1.6

 $\mu$  – mean  $\sigma$  – std. deviation

Protocol	Model [LOC]	Invariant [Conjectures]		[sec] σ	RW [sec]		Rounds	FOL µ	[sec] σ	т.о.
Paxos	85	11	1.0	0.1	1.2		2	1.2	0.1	0
Multi-Paxos	98	12	1.2	0.1	1.4	٦	4	1.8	0.4	0
Vertical Paxos*	123	18	2.2	0.2	-		8	107	129	30%
Fast Paxos*	117	17	4.7	1.6	1.5		16	229	110	70%
Flexible Paxos	88	11	1.0	0	1.2		Multi-	Paxo	os in	FOL
Stoppable Paxos*	132	16	3.8	0.9	1.6					

Protocol	Model [LOC]	Invariant [Conjectures]	EPR µ	[sec] <i>o</i>	RW [sec]		Rounds	FOL µ	[sec] <i>σ</i>	Т.О.
Paxos	85	11	1.0	0.1	1.2		2	186	123	50%
Multi-Paxos	98	12	1.2	0.1	1.4		4	300	0	100%
Vertical Paxos*	123	18	2.2	0.2	-		8	300	0	100%
Fast Paxos*	117	17	4.7	1.6	1.5		16	300	0	100%
Flexible Paxos	88	11	1.0	0	1.2	S	Stoppable Paxos in FO			
Stoppable Paxos*	132	16	3.8	0.9	1.6					

#### \*first mechanized verification

Transformation to EPR reusable across all variants!

#### 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 *e ally* [13]. We use a linear-time logic, so  $\Diamond$  can be defined by  $\Diamond F \stackrel{a}{=}$ 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  $\Box \Box P$  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 com-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 y<sup>th</sup> 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 Paxos. First, we define:

NotChoomhle(i k n) 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 sta

algorithm

 $\begin{array}{l} \textbf{Lemma 1} \\ 1, \forall i, b, v : \Box \left( \textit{Chosen}(i, b, v) \rightarrow \textit{Done2a}(i, b, v) \right). \end{array}$  $2, \forall i, b, v, w : \Box ((Done2a(i, b, v) \land Done2a(i, b, v)))$ 3.  $\forall i, b, a, v : \Box ((vote_i[a]|b] = v) \Rightarrow Done2a(i, b, b)$ 

some more definitions, culminating in the key invariant **Stoppable Paxos**  $\stackrel{a}{=} \forall c < b, w \neq v : NotChoosable(i, c, w)$  $\pi(i, b) \stackrel{\Lambda}{=}$  $b, w \in StopCmd : NotChoosable(j, c, w)$ (Rer(i, h, n) A  $md) \Rightarrow \forall j > i, c < b, w : NotChoosable(j, c, w)$  $\triangleq$  Done2a(i, b, v)  $\Rightarrow$  SafeAt(i, b, v)  $\land$  NoReconfigBefore(i, b) Dahlia Malkhi Leslie Lamport Lidong Zhou ∧ NoneChoosableAfter(i, b, z) safety proof is the following proof that PropInv is invariant. i, b, v : PropInv(i, b, v)) $t, o, c \rightarrow respins(t, u, c))$  PropIne(t, u, c) is true in the initial state because Done2a(...)We therefore need only show that it is stable. We do this by a true in a state *s* and proving it is true in state *t*. For any state *f* be its value in state *s* and *f'* be its value in state *t*. April 28, 2008  $\forall j, c, w : PropInv(j, c, w)$  *i* is an instance number, *b* a ballot number, *v* a command, and  $\ensuremath{\begin{array}{c}Q \ensuremath{a}\xspace{0.5ex} q \ensuremath{a}\xspace{0.5ex} q \ensuremath{a}\xspace{0.5ex} q \ensuremath{a}\xspace{0.5ex} s \ensuremath$ E1(b, Q) SafeAt(1, b, v) NoReconfigBefore(i, b)' ler(i, b, v)'тт  $\circ$ Inv(i, b, v))', it suffices to prove it for a partic-emma 1.7 (the stability of *NotCheosable*(...)) have been chosen as the  $j^{\text{th}}$  command for some j < i. Although the basic  $efore(i, b) \land NoneChoosableAfter(i, b, v)$ hat can possibly make PropInv(i, b, v)true. We can therefore assume  $s \to t$  is a purrum Q. Formula E1(b, Q) holds because it ase2a(i, b, v, Q) action. idea of the algorithm is not complicated, getting the details right was not ROVE" clause of  $\langle 1 \rangle 1$  are proved as steps  $\langle 1 \rangle 5$ , e steps are used in their proofs.  $\Rightarrow$  Done2a(j, mbal2a(j, b, Q), val2a(j, b, Q)) 17  $c \leq mbal2a(k, b, Q)$ ; and (4)1 and case assumption (3)3 imply  $w \in StopCmd$ . Therefore, NoReconfigBefore(k, mbal2a(k, b, Q)) implies nition of sval2a, we then have  $w \neq val2a(j, b, Q)$ . The (4)2 case as-sumption (which implies  $mval2a(j, b, Q) \neq -\infty$ ) and (1)3 then imply NotChoosable(j, c, w).NotChoosable(i, c, w). Chosen(i, b, v). (1)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). (1)5. SafeAt(i, b, v)' PROOF: We assume c < b and  $w \neq v$  and prove NotChooseMe(i, c, w)'. By Lemma 1.7, it suffices to prove NotChooseMe(i, c, w). We split the proof into two 2. Case: b = cWe split the proof into two cases. Theorem 2 D Stopping (2)1 Cast: mal2a(i + O) - T

in Q has sent a ("1b", a, b, (mbal2a(j, b, Q), val2a(j, b, Q)))<sub>j</sub> message, which implies  $votc_j[a](mbd2a(j, b, Q)) = val2a(j, b, Q)$  when the message was sent. Lemma 1.3 then implies Done2a(j, b, Q), val2a(j, b, Q)) was true when the message was sent, and is still true because Done2a(...) is stable. (1)3.  $\forall j, c < b, w : (c \le mbal2a(j, b, Q)) \land (w \ne val2a(j, b, Q)) \Rightarrow$ PROOF: We assume  $e \le \operatorname{schull construct}(q, e, w)$ PROOF: We assume  $e \le \operatorname{schull construct}(q, e, w)$ as  $e = \operatorname{schull construct}(q, e, w)$ . Since  $-\infty < e \le \operatorname{schull construct}(q, e, e, e)$ (2): implies  $\operatorname{Jonethan}(q)$ ,  $\operatorname{schull construct}(q)$ , conjecture(), monator(), is  $q_{ij}$  (variantly (i.e.  $q_{ij}$ ) main import intro-monomoup(), is  $q_{ij}$ . (j. (j. 4,  $v_{ij}$ ,  $c \in k$  is  $(mat2a_{ij}), k_Q = T = > NGChoosable(j, c, w)$ Parson: We assume c < b and  $sent2a_{(j, b, Q)} = T$  and prove NotChoosable(j, c, w). We explit the proof into two cases. (2)1. Case: mbd2a\_{(j, b, Q)} =  $-\infty$ 2)1. CASE. INSERTIC, i, Q) = −0. PROOF: The case assumption implies mbd2a(j, b, Q) < c, so assumption (1)1.4 and Lemma 3 imply NotChoosable(j, c, w). (3) Contact moment(), u(q) p = 0.0 PROOP: Since < b, we can split the proof into the following three cases. (3)1. CASE: mbal2a(j, b, Q) < c < b PROOP: By assumption (1)1.4, the case assumption and Lemma 3 imply NetChoosenble(j, c, w). (3)2. CASE:  $c \le mbal2a(j, b, Q)$  and  $w \ne val2a(j, b, Q)$ 

easy.

PROOF: BY (13. (3). CASE:  $\epsilon \ge mhol2a(j, b, Q)$  and w = val2a(j, b, Q)(4).1.  $val2a(j, b, Q) \in SopCmd$  and wc can choose k > j such that  $mhol2a(k, b, Q) \ge mhol2a(j, b, Q)$ . PROOF: We deduce that  $val2a(j, b, Q) \in StopCmd$  and such a k exists. by the (2)2 case assumption, the assumption  $sval2a(i, b, Q) = \top$ , and the definition of soul2a. (4)2. Done2a(k, mbal2a(k, b, O), val2a(k, b, O)) 

18

PROOF: Assume  $mbal2a(j, b, Q) \neq -\infty$ . By definition of mbal2a, this implies val2a(j, b, Q) is a command (and not  $\top$ ). Since E1(b, Q) holds by assump-tion (1)1.4, the definitions of mbal2a and val2a imply that some acceptor a

NotChoosable(1, c, w)

(2)2. CASE:  $mbal2a(j, b, Q) \neq -\infty$ 

PROOF: By (1)3.

 $\begin{array}{l} (2) = C_{cont} = e^{-i\pi k k t} (k,k,k,q) \\ Parcore A sumption (k,k,q) \\ Parcore A sumption (k),k,q) \\ Parcore A sumption (k),k \\ Parcore A sumption ($ (1)6. NoReconfigBefore(i, b)<sup>i</sup> (1)o, Noncompany (n)(i, o) 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. (2)1. CASE: b = c 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)<sup>2</sup> (sub-stituting i and v for the existentially quantified variables), which by the case assumption proves NotChoosable(j, c, w)<sup>2</sup>. (2)2. CASE: c < b (2) CASH: c < b PROOF: We consider two sub-cases. (3) I. CASH: sna22c(f, b, Q) = T PROOF: (1) And case assumption (2)2 imply NotCheosoble(f, c, w). (5): C CASH: sna22c(f, b, Q) ≠ T PROOF: By case assumption (2)2, we have the following two sub-cases. (4)1. CASE: meal2a(j, b, Q) < c < b PROOF: Assumption (1)1.4, the case assumption, and Lemma 3 imply NotCheosable(j, c, w). Net(*Hosenble(j*, c, w), (§)2. CAN: c: smol2c(*j*, *k*, *Q*)) PROOF. Assumption (1)1.3 implies *E0*(*i*, *k*, *Q*). The (3)2 case assumption, the assumption j < i, and E0(i, k, Q) imply sul2c(*j*, *k*, *Q*)  $\notin$  *StopCode*. The assumption are  $\notin$  *StopCode* then in-plies  $w \neq i = si2(i, k, Q)$ . Whe (3)2 case submption and the defi-ption  $\psi = i = si2(i, k, Q)$ . By the (3)2 case submption and the defi-

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

22. CASE:  $sinu 2n(i, b, Q) \neq i$ PROOF: Since c < b, we can break the proof into two sub-cases. (3)1. CASE: mbd2a(i, b, Q) < c < bPROOF: Assumption (1)1.4 and Lemma 3 imply NotChooseble(i, c, w)

19

1959 Case: mal?a(i h O) + T

we sput the proof min to cause. (2): C.Ast: scal2a(i, b, Q) = 7 Protor: Assumption (1):13 implies E4(i, b, Q, v), so the assumption  $v \in Step-Grin Implies <math>E4b(i, b, Q, v)$ . The case assumption, the assumption j > i, and E4b(i, b, Q, v) imply var2ba(j, b, Q) = 7. The assumption e < band step (1) then imply NarChoosele(j, c, v). and step (1)4 then imply soft-messions(x, w). (2)2. Cossi: scalar(i, b, Q) = v(3)1. sud2a(i, b, Q) = v wd2a(i, b, Q) = vPROOF. Assumption (1)13.3 implies E3(i, b, Q, v), which implies sud2a(i, b, Q) = v. The case assumption and the definition of sval2a then implies ud2a(i, b, Q) = v. mppose rescale, i.e, Q = v.  $(g_{2}, Dorek2(u, mbi2a(i, \xi_{2}, Q), v)$  Pncore; (3), assumption (1)1.4, and the definition of val2a imply  $vort_{i}(a)[mbi2a(i, \xi_{2}, Q)] = v$  for some acceptor v in Q, which by Lemma I.3 implies  $Done2a(i, mbi2a(i, \xi_{2}, Q), v)$ . By the assumption  $v \in A$  i, it suffices to consider the following two cases. By the summption < k, it influes to consider the biologing two cases. (b) Close: < c and k < k, k and k < k, (4)2. NotChoosable(j, c, w) (4) PROOF: (4)1 and case assumption (3)4 imply mbal2a(j, b, Q) < c < b By assumption (1)1.4, Lemma 3 implies NotChoosable(j, c, w). Theorem 1 Consistency 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  $b \leq c$ . We then have two eases. PROOF: We assume  $\pi \neq w$  and obtain a contradiction. Lemma 1.1 and Chosen(i, c, w) imply Done2a(i, c, w). By Lemma 4, this implies 20

SafeAt(i, c, w). The assumptions b < c, an  $v \neq w$  then im-ply NotChoosable(i, b, v). By Lemma 2, this contradicts the assumption

PROOF: Lemma 1.1 implies  $Done2a(i, b, v) \wedge Done2a(i, c, w)$ , 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. 1. CASE:  $c \in A$ PIGOD: *Chosen(i, b, v)* and Lemma 1.1 imply *Denc2a(i, b, v)*. This and Lemma 1 imply *NeucChoosableAfter(i, b, v)*, which by the case assumption and the assumption *Chosen(j, c, v)* and Lemma 2 then provide the required contradiction. 2. CASE:  $c \ge b$ 

vide the required contradiction

#### A.2 The Proof of Progress.

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  $\Diamond$  Chosen(i, b, v) or  $(v \in StopCmd) \land \Diamond$  Chosen(j, b, v), for some i < i(1)1. ○□E1(b, Q)

(11) COLLAIS, (2) (11) COLLAIS, (2) (12) COLLAIS, (2) (12) (12) COLLAIS, (2) COL (1)2.  $\forall i = m : \square(Dame2a(i, h, w) \Rightarrow \bigcirc(Dame1(i, h, w))$ 

1/2:  $\forall i, v : \Box(Dmc2a(i, b, w)) \Rightarrow OCassen(i, b, w)$ Photor: Dmc2a(i, b, w) mass that a  $Plant^2a(i, b, w)$  ration has been executed sending  $a^{-}(2a^*, b, w)$ , message to every acceptor a. If a is in G, then assemption Pl(b, G) implies that it eventually receives that the message. Assumptione Pl(b, G)implies  $bra(a) \leq b$ , so Pl(b, G) implies that every a in G eventually executes Pare2b(i, a, b), setting va(a) (G)[b] to w. Hence, eventually Casen(i, b, w)

21

(1)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:  $sval2a(i, b, Q) = \top$ 

PROOF: Assumption  $\langle 1 \rangle 1.3$  implies E4(i, b, Q, v), so the assumption  $v \in StopCmd$  implies E4b(i, b, Q, v). The case assumption, the assumption j > i, and E4b(i, b, Q, v) imply  $sval2a(j, b, Q) = \top$ . The assumption c < b and step  $\langle 1 \rangle 4$  then imply NotChoosable(j, c, w).

(2)2. Case:  $sval2a(i, b, Q) \neq \top$ 

 $\langle 3 \rangle 1. \ sval2a(i, b, Q) = val2a(i, b, Q) = v$ 

PROOF: Assumption  $\langle 1 \rangle 1.3$  implies E3(i, b, Q, v), which implies 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.

(4)2. NotChoosable(j, c, w)

PROOF: (4)1 and case assumption (3)4 imply mbal2a(j, b, Q) < c < b. By assumption (1)1.4, Lemma 3 implies NotChoosable(j, c, w). Challenge: How to use restricted first-order logic to verify interesting systems?

- Expressing transitive closure
  - Linked lists
  - Ring protocols
- Expressing sets and cardinalities
  - Paxos and its variants
  - Byzantine Fault Tolerance
  - Reconfiguration

• Liveness and temporal properties [POPL'18]

[POPL'18] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, MS, Sharon Shoham Reducing Liveness to Safety in First-Order Logic

Protocol	Model [LOC]	Invariant [conjectures]	Time [sec]	
Leader in Ring	59	4	1.5	
Learning Switch	50	5	1.5	
DB Chain Replication	143	9	1.7	
Chord	155	12	2.4	P Ir
Lock Server (500 Coq lines [Verdi])	122	9	2	V
Distributed Lock (1 week [IronFleet])	41	7	1.4	h
Single Decree Paxos (+liveness)	85	11	10.7	
Multi-Paxos (+liveness)	98	12	14.6	
Vertical Paxos*	123	18	2.2	
Fast Paxos	117	17	6.2	
Flexible Paxos	88	11	2.2	
Stoppable Paxos (+liveness) *	132	16	18.4	
Ticket Protocol (+liveness)	86	37	6	
Alternating Bit Protocol (+liveness)	161	35	10	
TLB Shootdown (+liveness) *	385	91	380 (FOL)	
Practical Byzantine Fault Tolerance	\ <b>\</b> /o			
Reconfiguration	VVO	ork in progress		

Proof / code ratio: IronFleet: ~4 Verdi: ~10 Ivy: ~0.2

\* First mechanized liveness proof

## Summary

- Distributed protocols are interesting for verification
  - But real distributed systems are more complex
- Decidable logics can be used to reason about interesting systems
  - No more butterfly effects
  - But some jagged corners
  - Details on Wednesday