

Workshop in Verification of Distributed Protocols

Mooly Sagiv, Oded Padon

08-March-2018

<http://www.cs.tau.ac.il/~odedp/workshop18/>

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

Administration

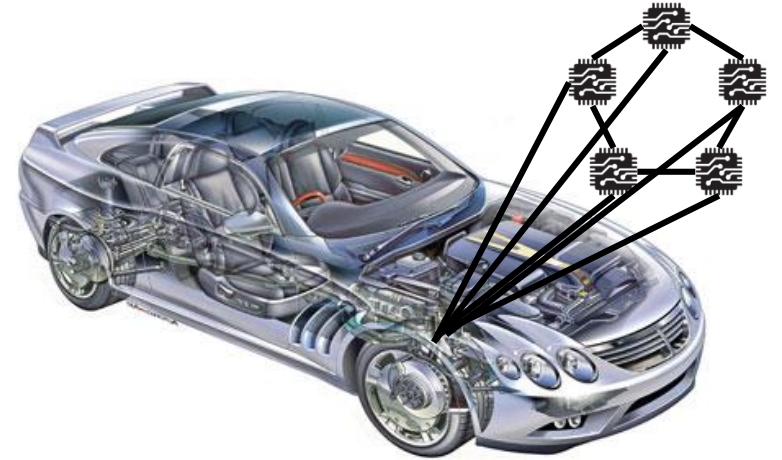
- Start-off meeting (today)
- Project teams:
 - 2-3 students
 - Each team will take different a project, and work independently during the semester
 - Meet with Oded / Mooly as needed
- If needed, we'll have more workshop meeting during the semester
- 14/6 – project presentation meeting
 - Each team will present project
 - Project must be finished and approved by Oded / Mooly before

Possible Projects

- Use Ivy to verify any distributed / shared memory algorithm
- Paxos variants
 - Disk Paxos, Generalised Paxos, EPaxos (see <http://paxos.systems/variants.html> for ideas)
 - Prove reconfiguration / failure recovery / log truncation / liveness
- Mutual Exclusion Algorithms
 - Knuth's Algorithm, Lamport's Bakery, Patterson, ...
 - Prove safety and liveness
- Blockchain algorithms
 - Algorand, HoneyBadgerBFT, Bitcoin-NG, ...
- Improve Ivy
 - Experiment with other SMT solvers (e.g. iProver, CVC4, Vampire, SPASS)

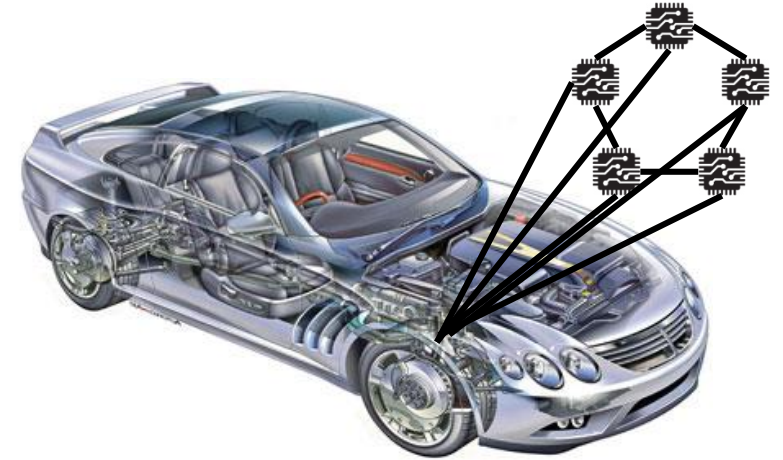
Why verify distributed protocols?

- 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

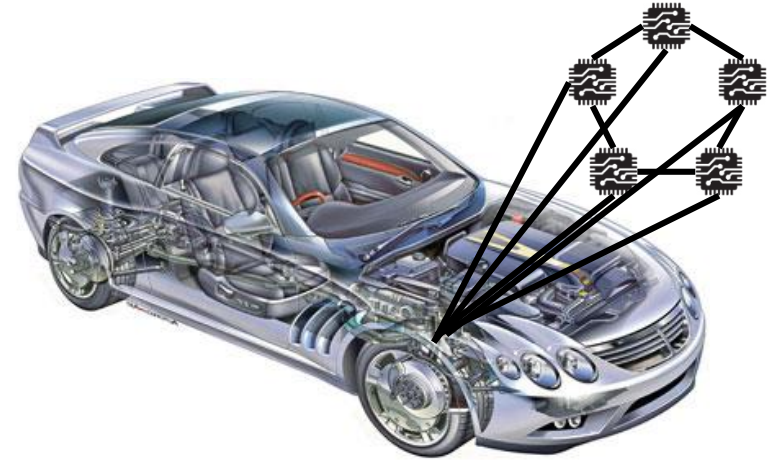
Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

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
- Distributed systems are notoriously hard to get right



SIGCOMM'01

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

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

Attractive features of Chord include i
correctness, and provable performance
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 satis-
fies one invariant, but is still not correct. The
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
protocols: (1) agreement, (2) view change, and (3)
agreement protocol orders requests for exe-
cution. View change protocol coordinates

CACM'08

Zyzzyva: Speculative Byzantine Fault Tolerance

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.

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

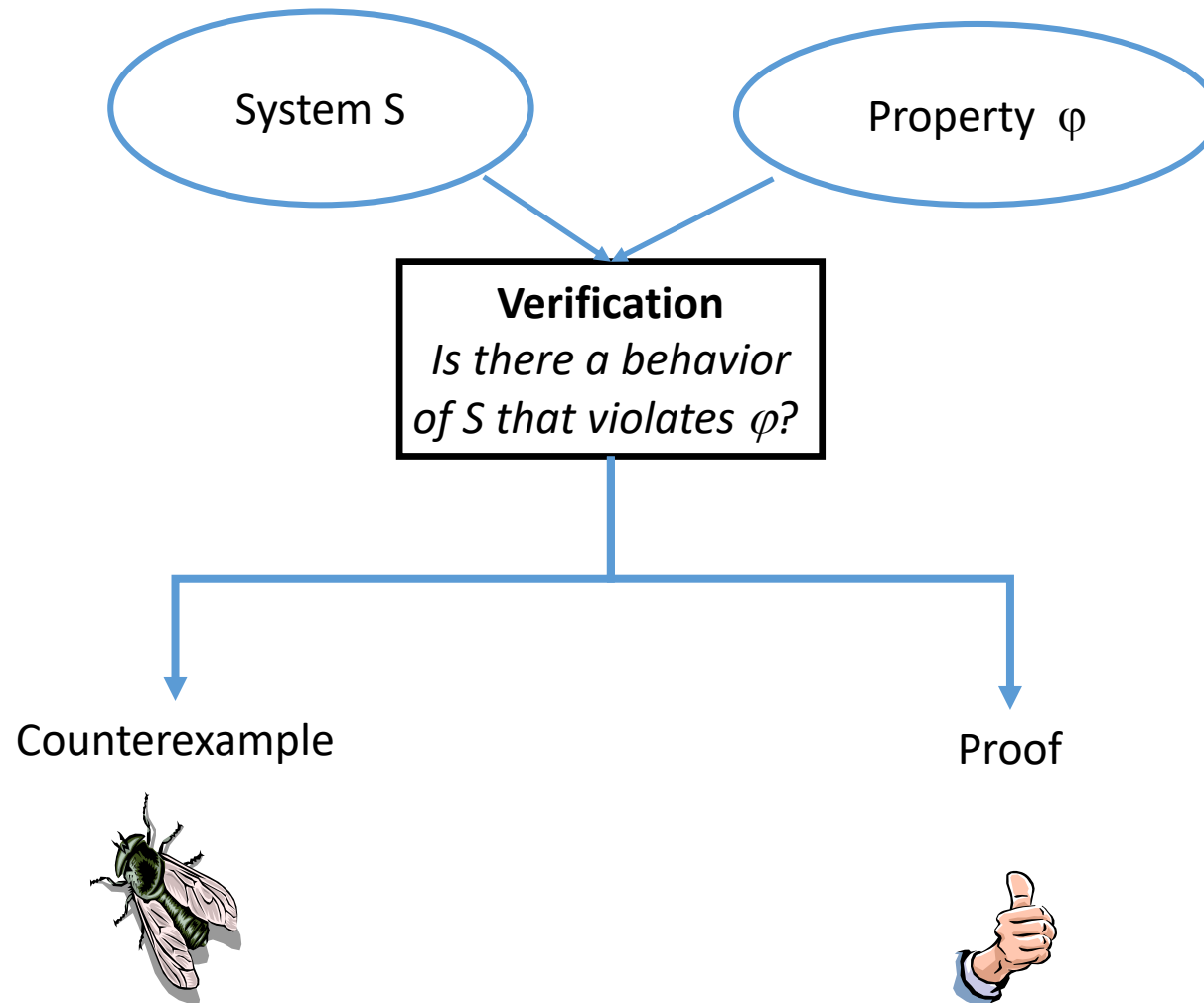
Our approach: decompose verification into decidable problems

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

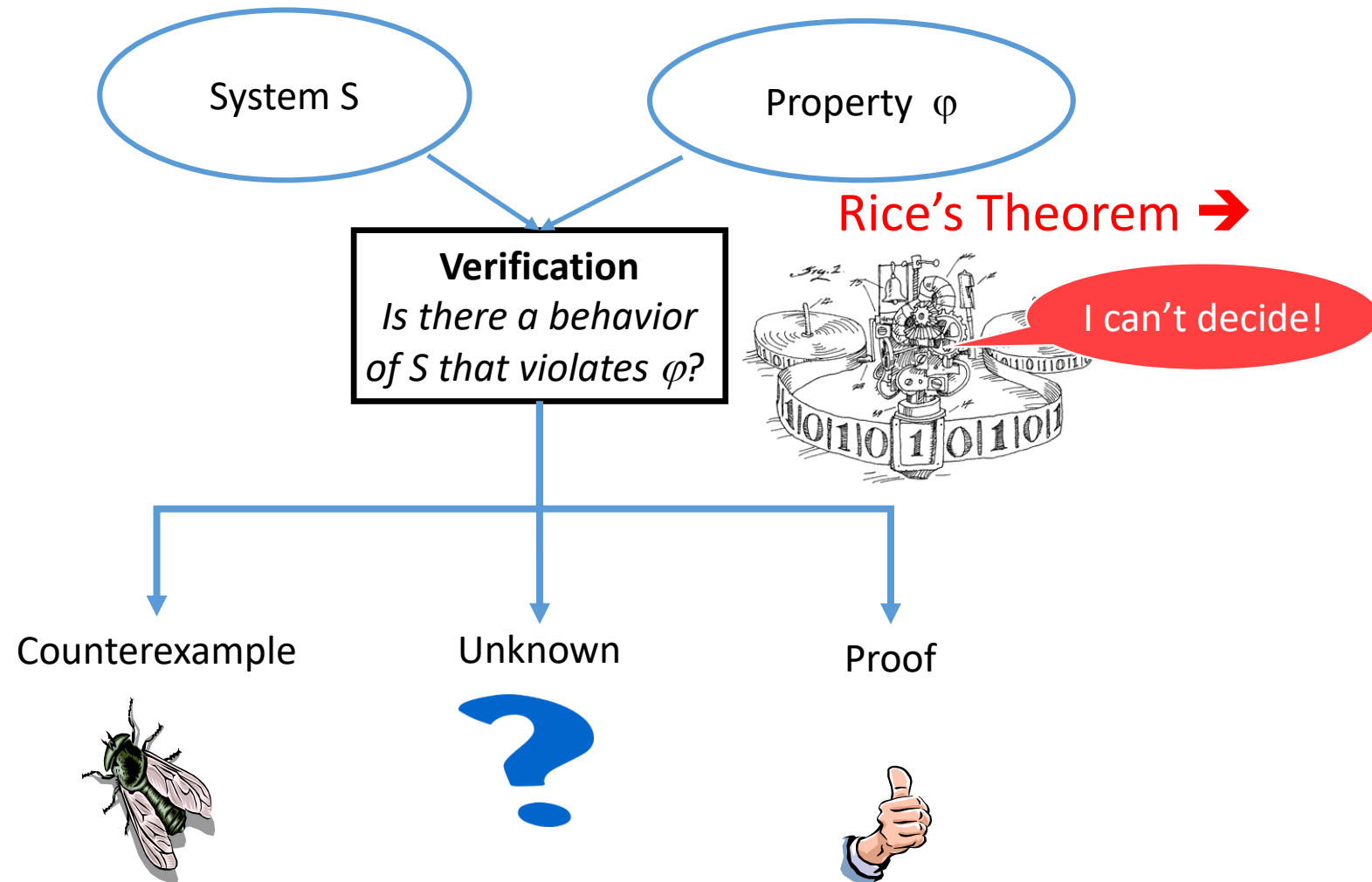
[SOSP'15] Hawblitzel et al. IronFleet: proving practical distributed systems correct

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

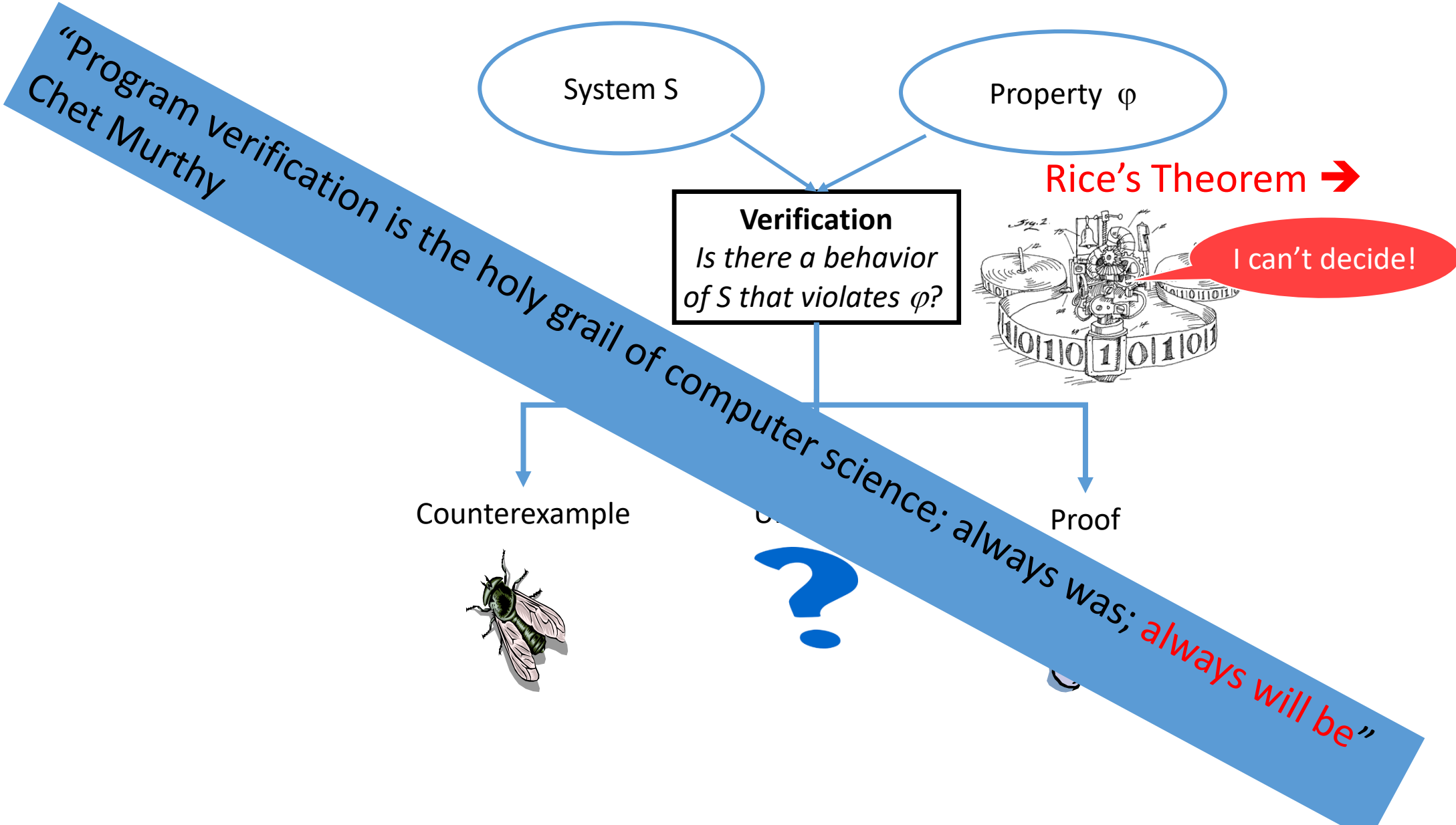
Automatic verification of infinite-state systems



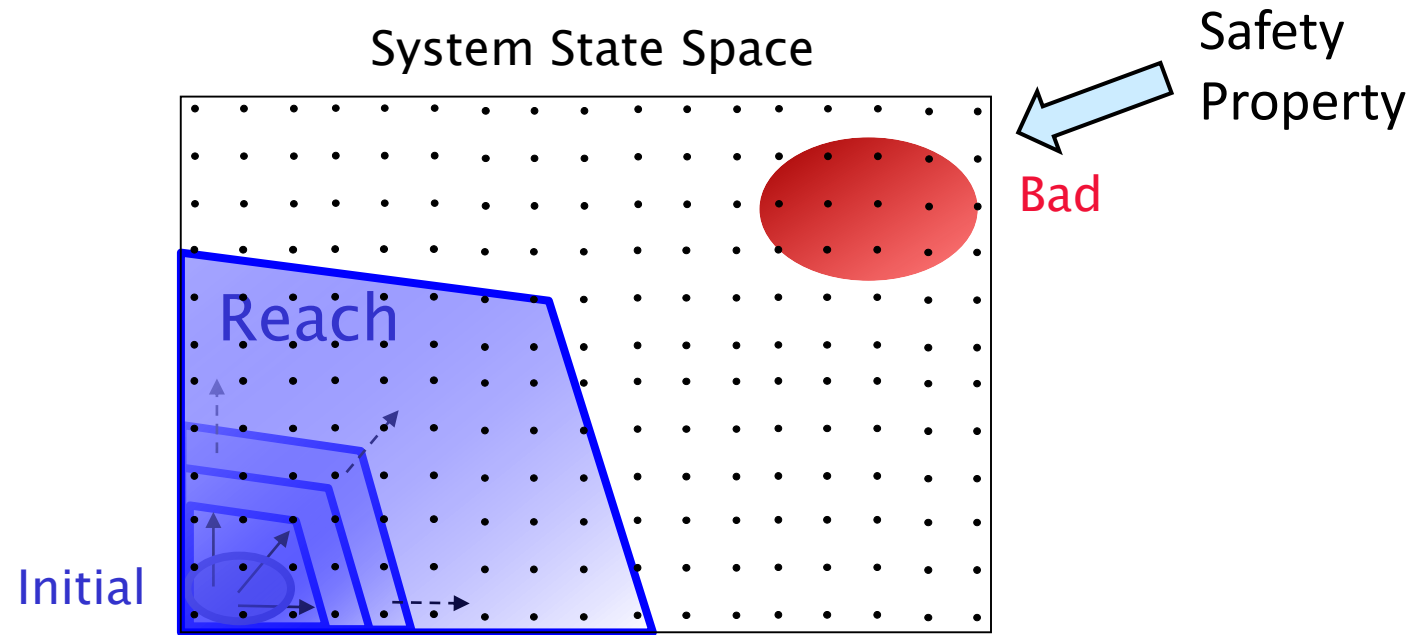
Automatic verification of infinite-state systems



Automatic verification of infinite-state systems

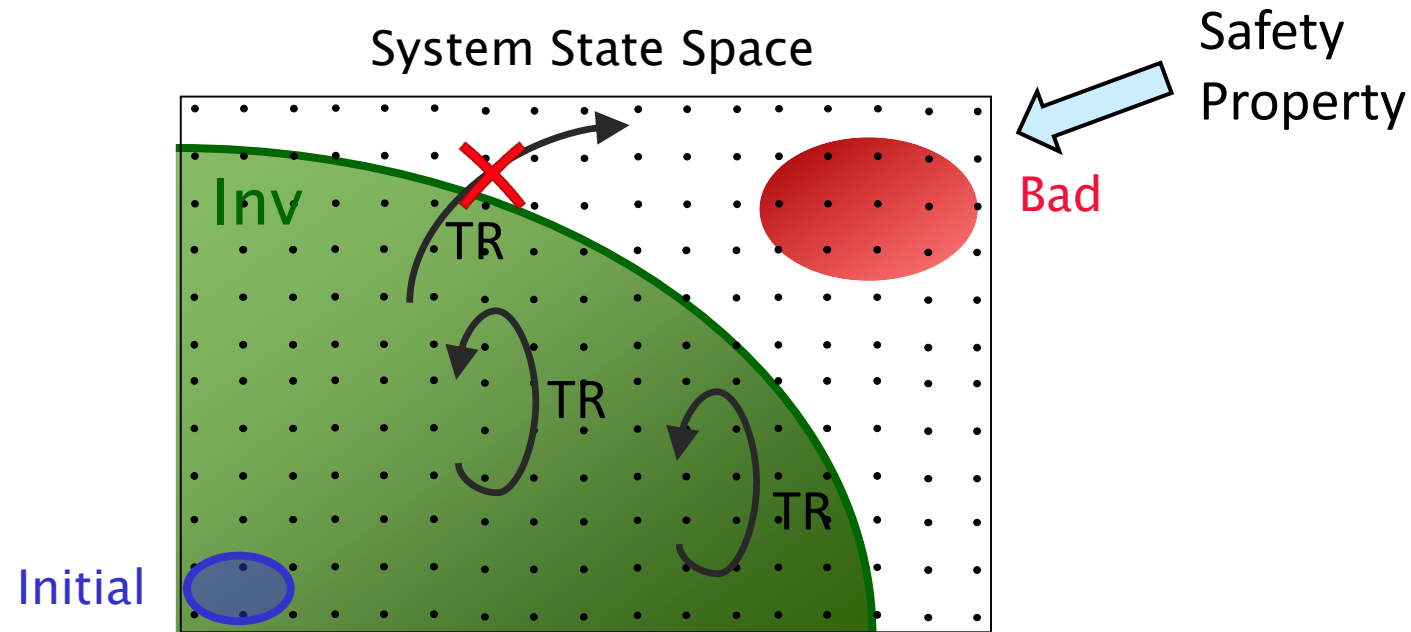


Inductive invariants



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

Inductive invariants



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

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

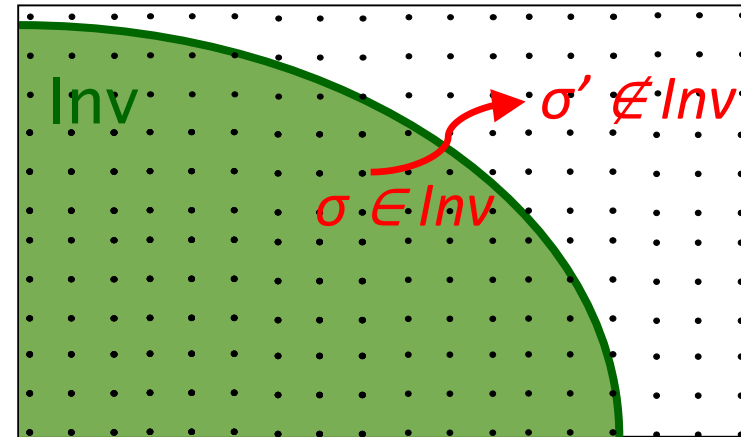
$$\text{Inv} \cap \text{Bad} = \emptyset \text{ (Safety)}$$

$$\text{Init} \subseteq \text{Inv} \text{ (Initiation)}$$

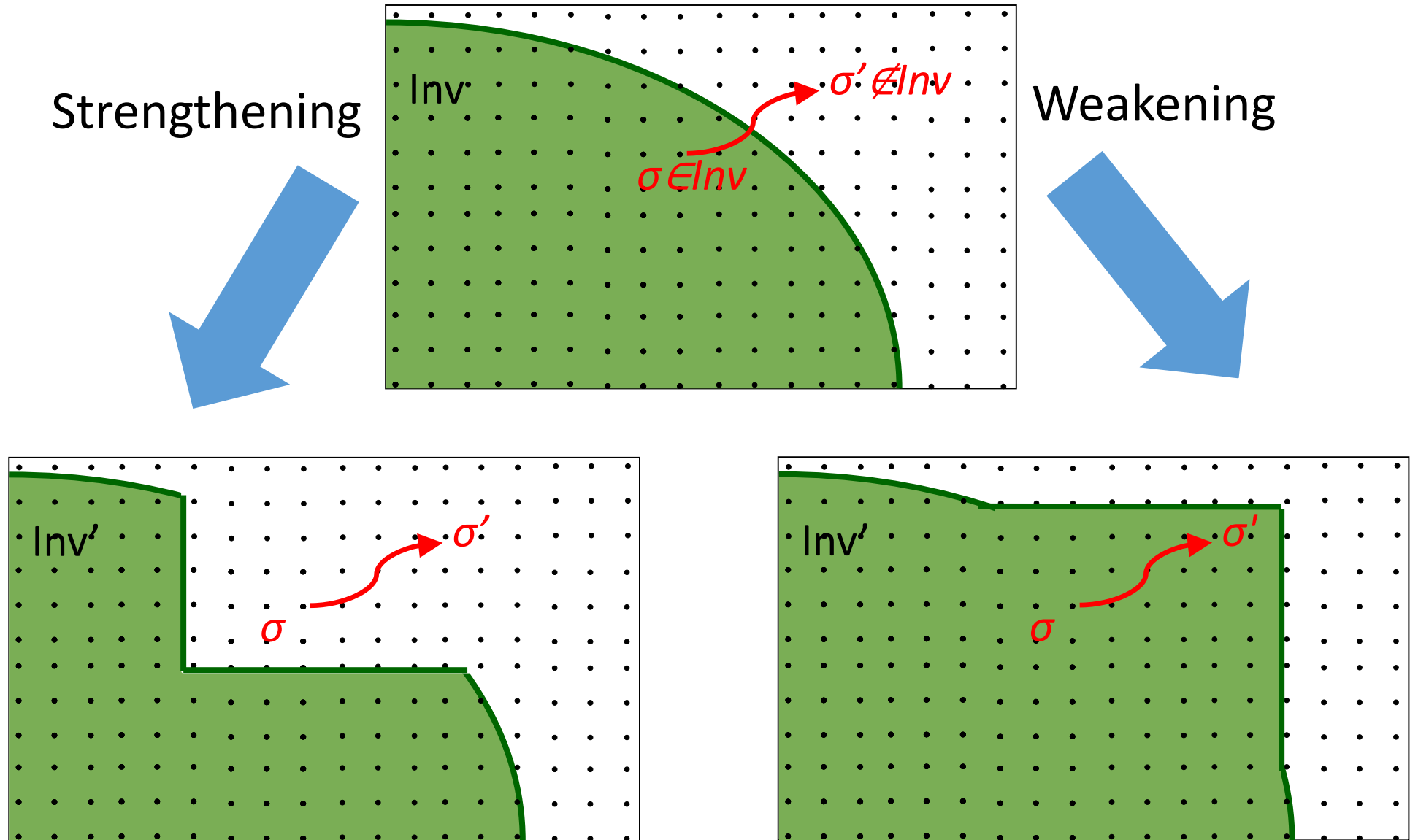
$$\text{if } \sigma \in \text{Inv} \text{ and } \sigma \rightarrow \sigma' \text{ then } \sigma' \in \text{Inv} \text{ (Consecution)}$$

Counterexample To Induction (CTI)

- States σ, σ' are a CTI of Inv if:
 - $\sigma \in Inv$
 - $\sigma' \notin Inv$
 - $\sigma \rightarrow \sigma'$
- 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

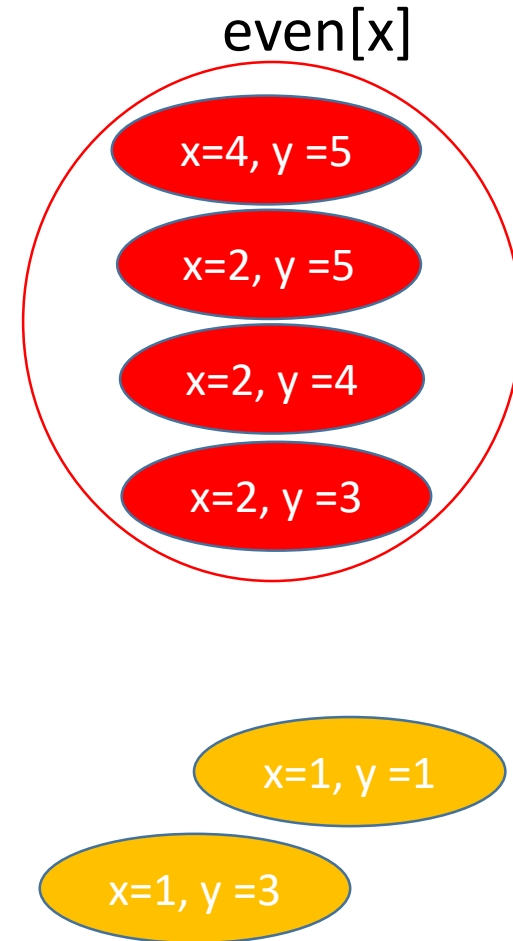
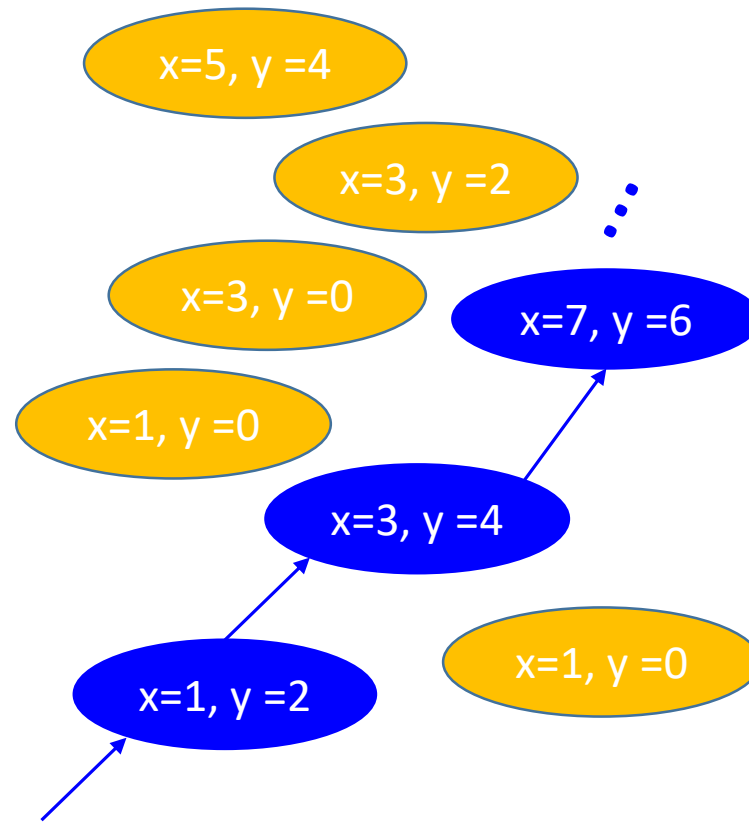


Strengthening & weakening from CTI



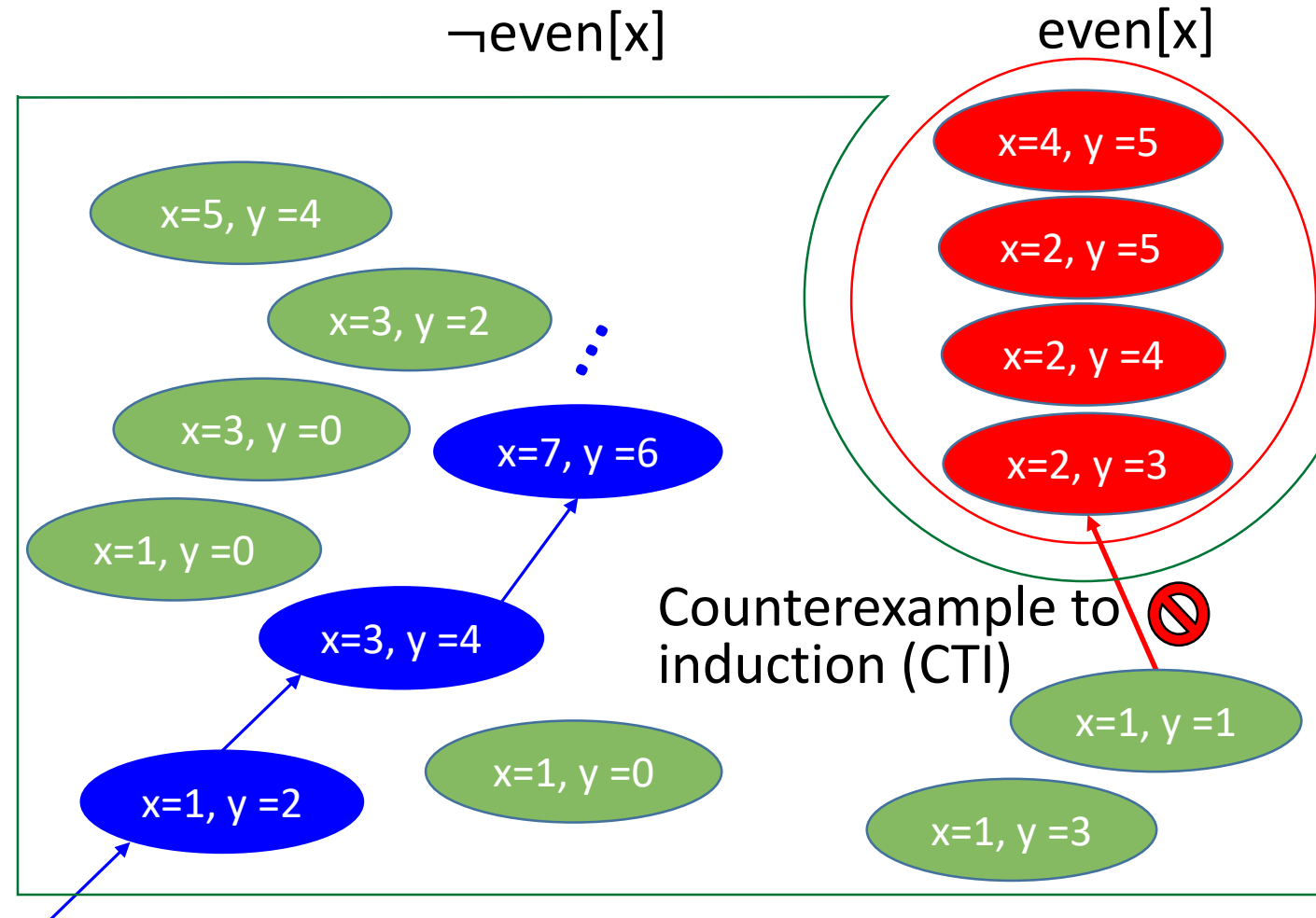
Simple example: loop invariants

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



Simple example: loop invariants

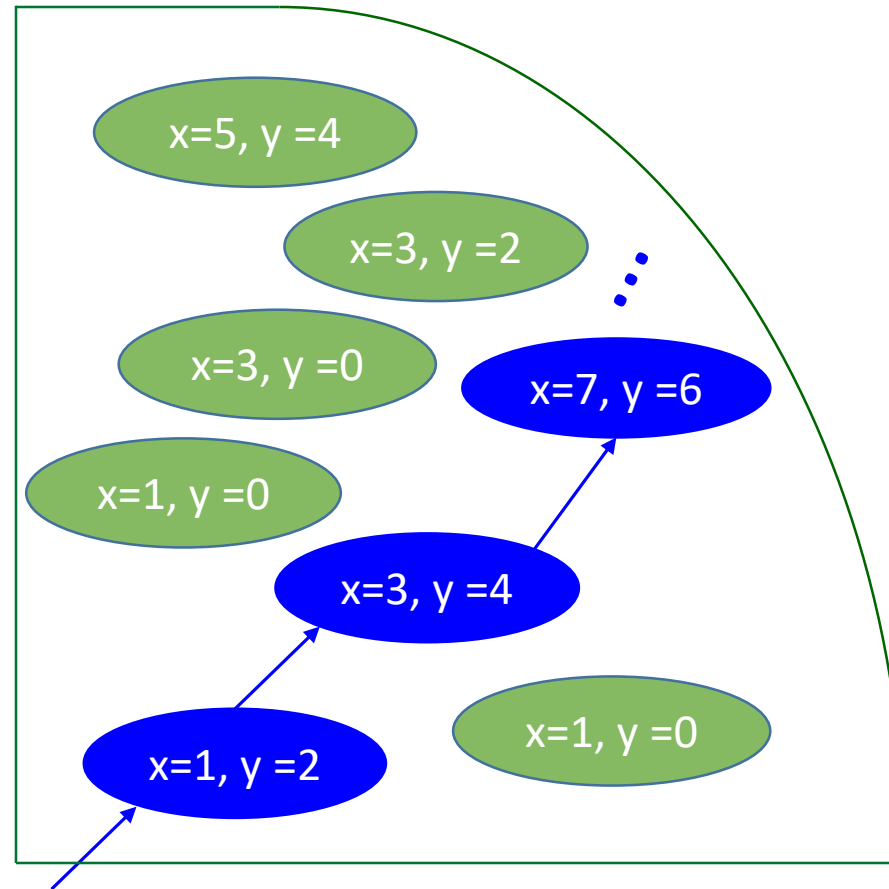
```
x := 1;  
y := 2;  
while * do {  
  assert  $\neg\text{even}[x]$ ;  
  TR | x := x + y;  
    y := y + 2;  
}
```



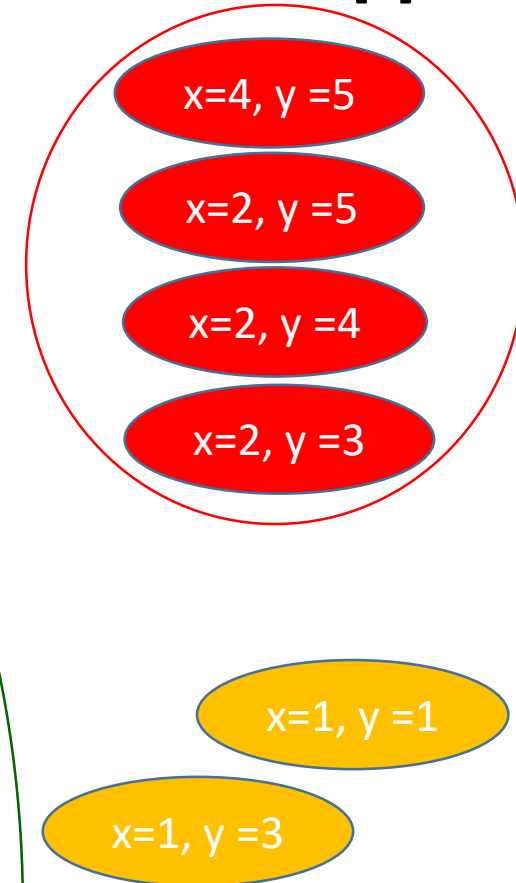
Simple example: loop invariants

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

$$\text{Inv} = \neg\text{even}[x] \wedge \text{even}[y]$$



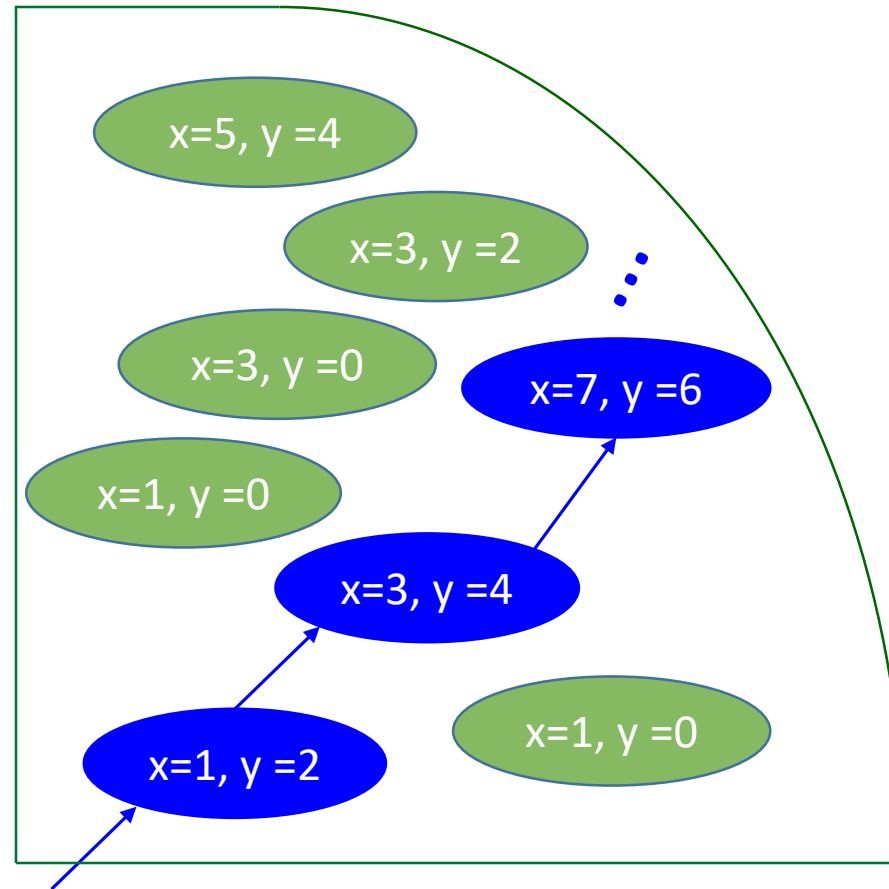
$$\text{even}[x]$$



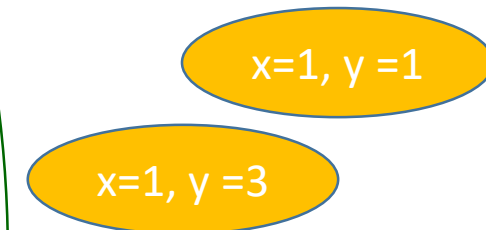
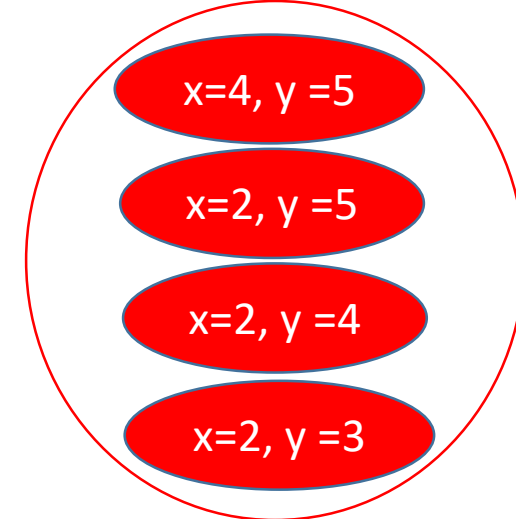
Simple example: loop invariants

```
x := 1;  
y := 2;  
while * do {  
  assert  $\neg\text{even}[x]$ ;  
  TR  $x := (x*x - y*y)/(x - y);$   
  y := y + 2;  
}
```

$\text{Inv} = \neg\text{even}[x] \wedge \text{even}[y]$



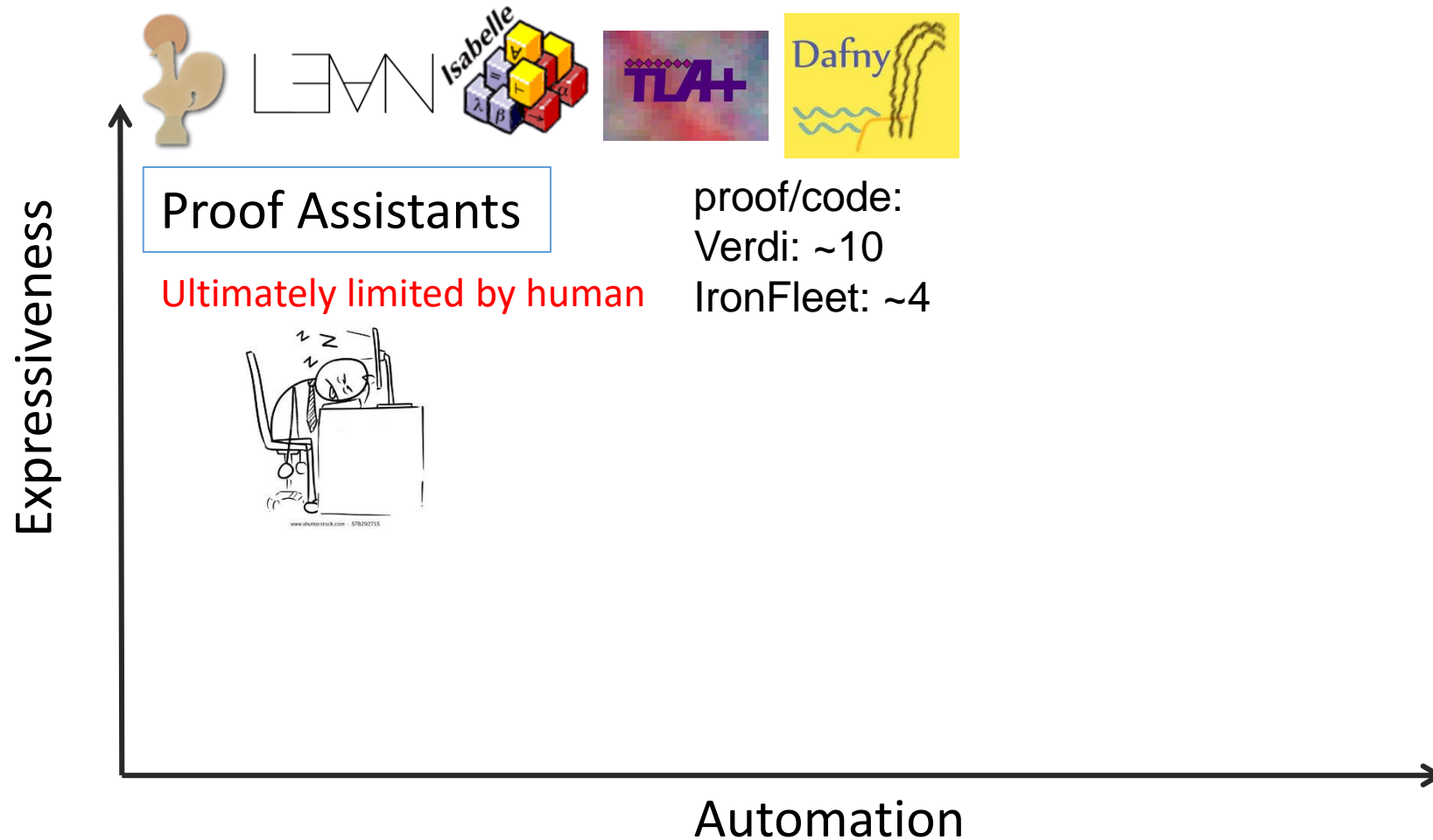
$\text{even}[x]$



Challenges in Deductive Verification

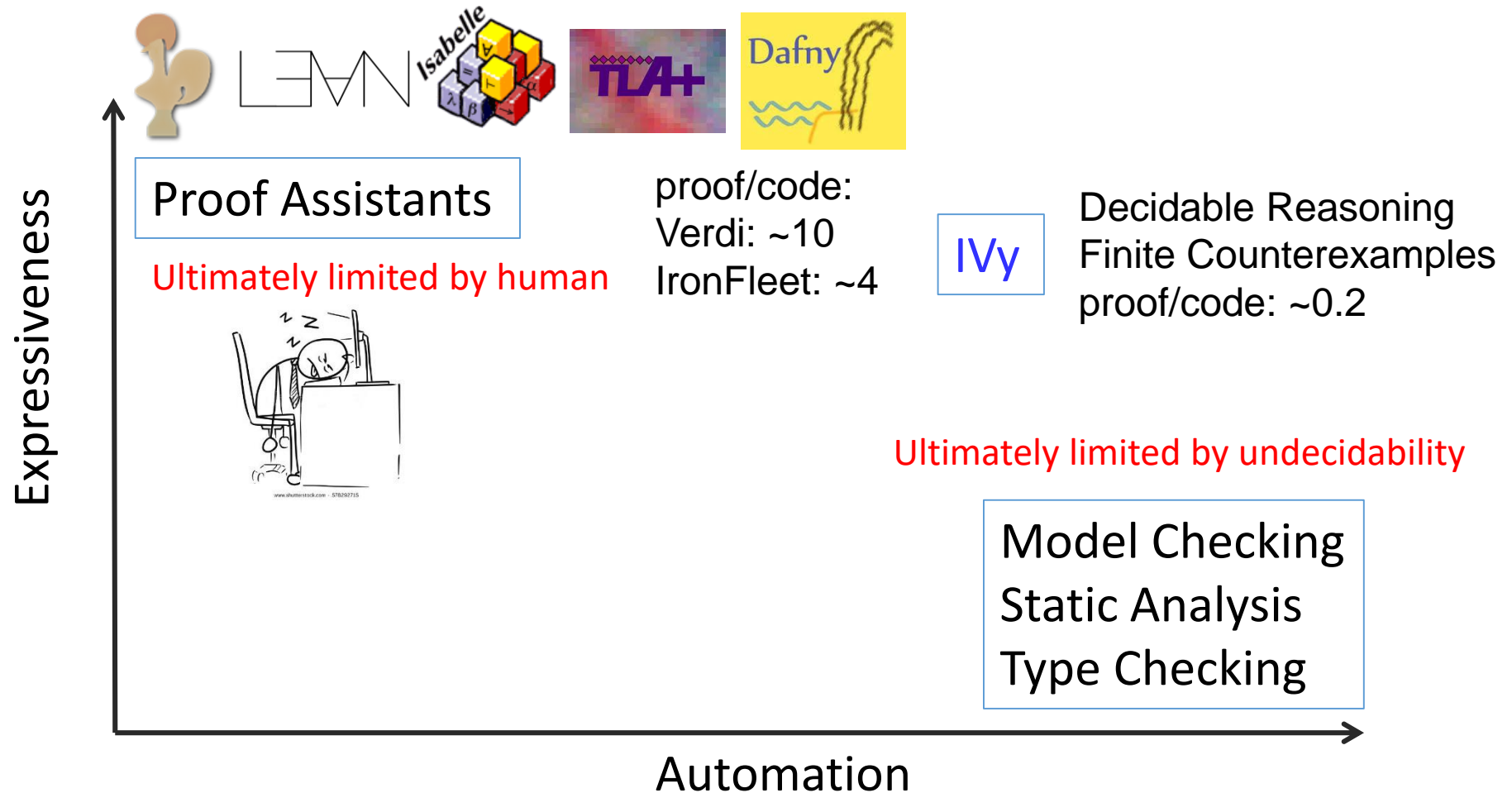
1. **Formal specification**: formalizing infinite-state systems
 - Modeling the system and property (TR, Init, Bad)
2. **Deduction**: checking inductiveness
 - Undecidability of implication checking
 - Unbounded state (threads, messages), arithmetic, quantifier alternation
3. **Inference**: inferring **inductive invariants** (Inv)
 - Hard to specify
 - Hard to infer
 - Undecidable even when deduction is decidable

State of the art in formal verification



"the proofs consisted of about 5000 lines and assumed several nontrivial invariants of the Raft protocol. This paper discusses the verification of Raft as a whole, including all the invariants from the original Raft paper [32]. These new proofs consist of about 45000 additional lines" [Verdi, CPP'16]

State of the art in formal verification



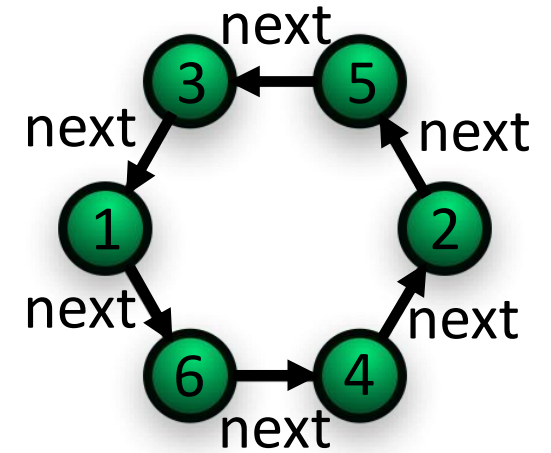
"but our input language cannot compete in generality with mechanized proof methods that rely heavily on human expertise, e.g., IVY [55], Verdi [68], IronFleet [38], TLAPS [16]" [Konnov et al, POPL'17]

IVy's Principles

- Specify systems and properties in decidable fragment of first-order logic (EPR)
 - Allows quantifiers to reason about unbounded sets
 - Decidable to check inductiveness
 - Finite counterexamples to induction, display graphically
 - Logic is mostly hidden
- Interact with the user to find inductive invariants
- Challenge: use restricted logic to verify interesting systems
 - Paxos, Reconfiguration, Byzantine Fault Tolerance
 - Liveness and Temporal Properties

Example: Leader Election in a Ring

- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:
 - Each node sends its id to the next
 - A node that receives a message passes it (to the next) if the id in the message is higher than the node's own id
 - A node that receives its own id becomes the leader
- Theorem:
 - The protocol selects at most one leader



[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

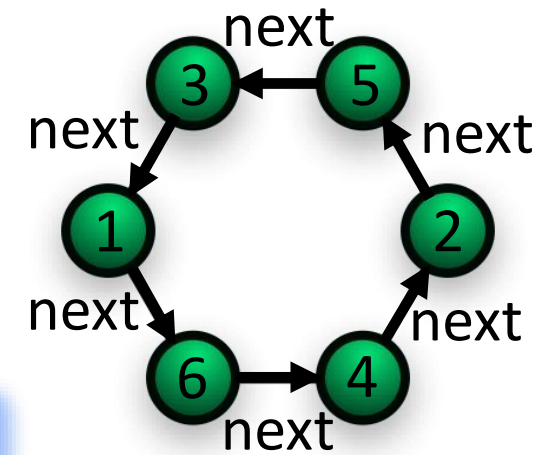
- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:

- **Proposition:** This algorithm detects one and only one highest number.

- **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

- A node initiator. Only one message, that with the highest number, will not encounter a higher number on its way

- Theorem
 - The back is the one with the highest number.



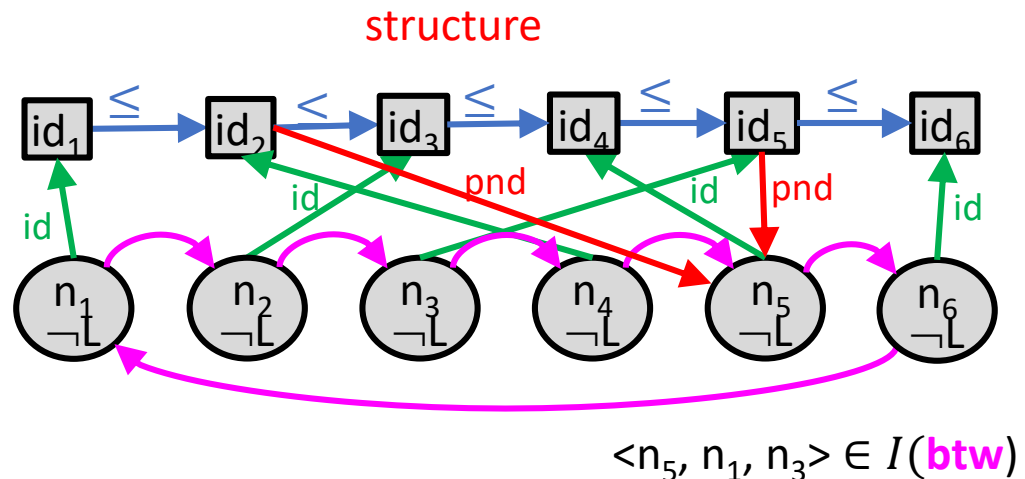
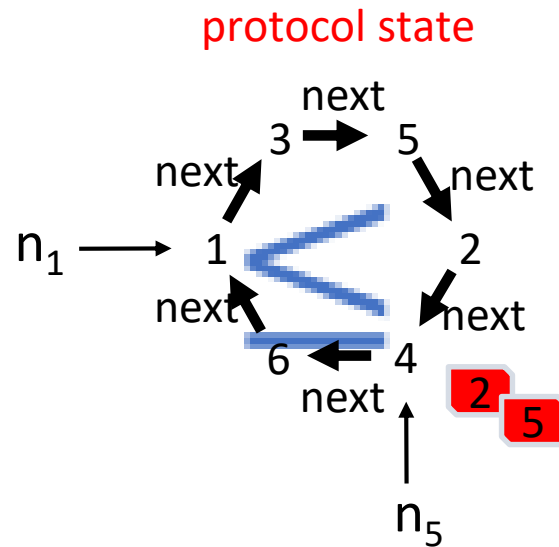
if the id in the

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

Leader Election Protocol (IVy)

- \leq (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

Axiomatized in
first-order logic



Leader Election Protocol (IVy)

- \leq (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

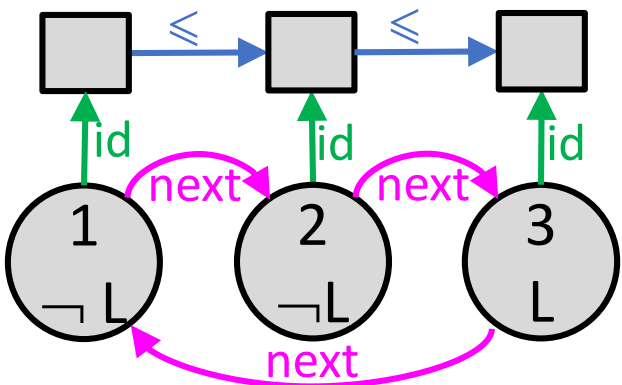
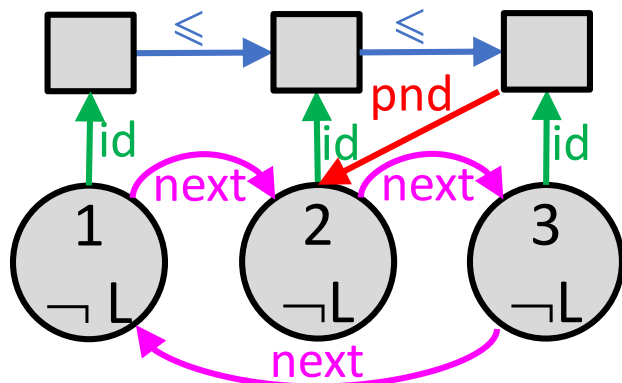
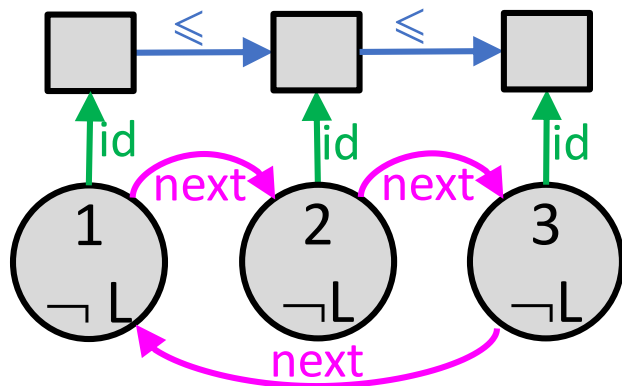
```
action send(n: Node) = {  
    "s := next(n)";  
    pending(id(n), s) := true  
}
```

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

$\exists n, s: \text{Node}. \text{"s := next(n)"} \wedge \forall x: \text{ID}, y: \text{Node}. \text{pending}'(x, y) \leftrightarrow (\text{pending}(x, y) \vee (x = \text{id}(n) \wedge y = s))$

protocol = (send | receive)*

assert I0 = $\forall x, y: \text{Node}. \text{leader}(x) \wedge \text{leader}(y) \rightarrow x = y$

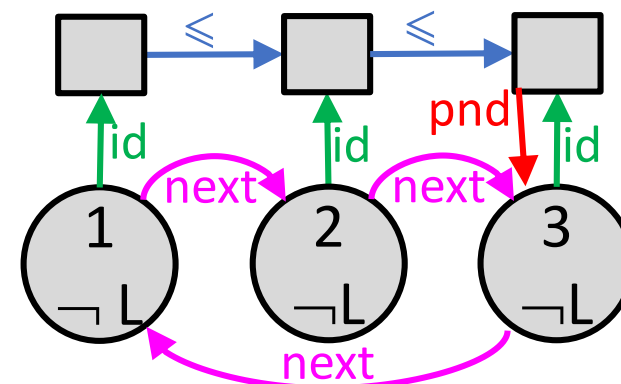
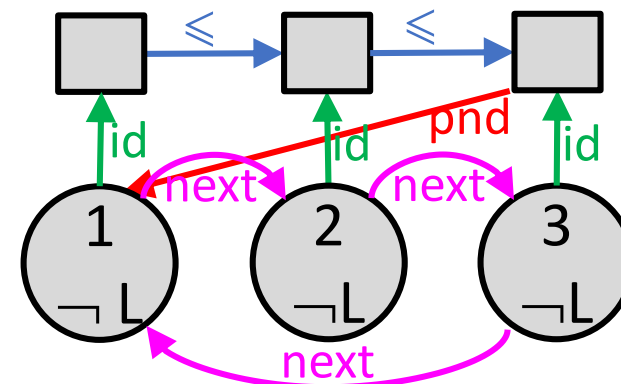


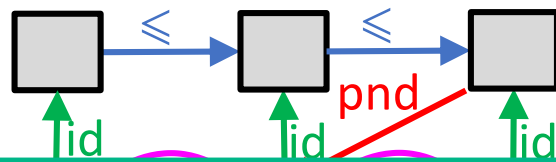
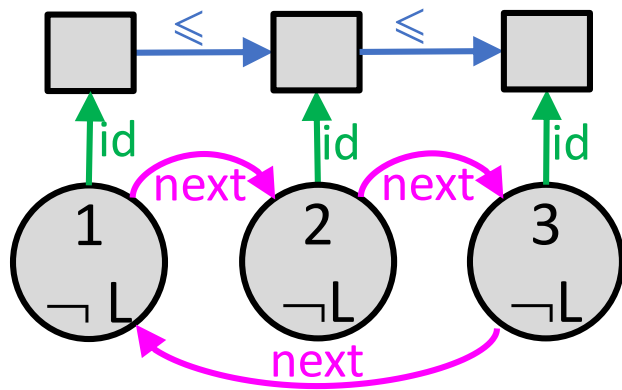
$send(3)$

$rcv(1, id(3))$

$rcv(2, id(3))$

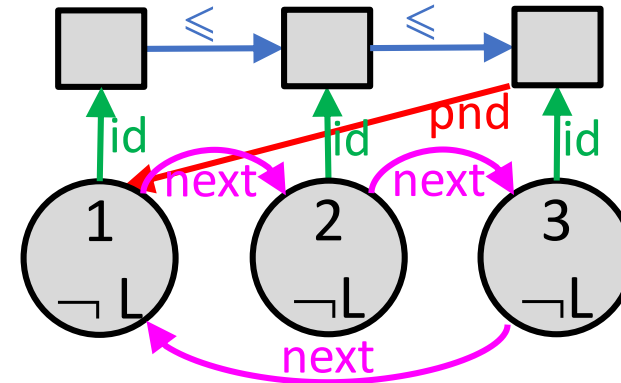
$rcv(3, id(3))$



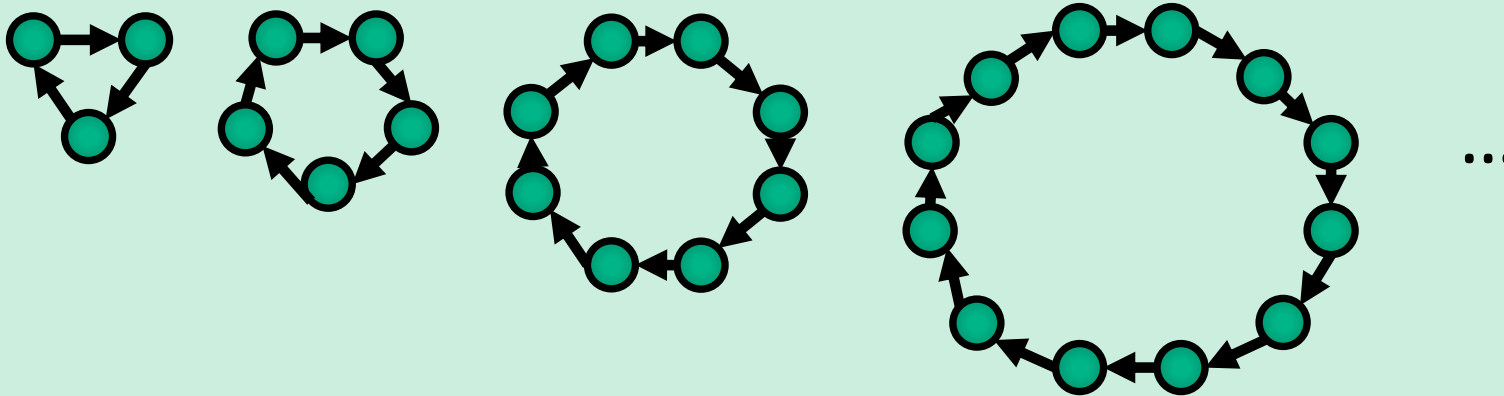


send(3)

rcv(1, id(3))



Specify and verify the protocol for **any** number of nodes in the ring



Inductive Invariant for Leader Election

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

Safety property: I_0

$$I_0 = \forall x, y: \text{Node}. \text{leader}(x) \wedge \text{leader}(y) \Rightarrow x = y$$

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

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

The leader has the highest ID

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

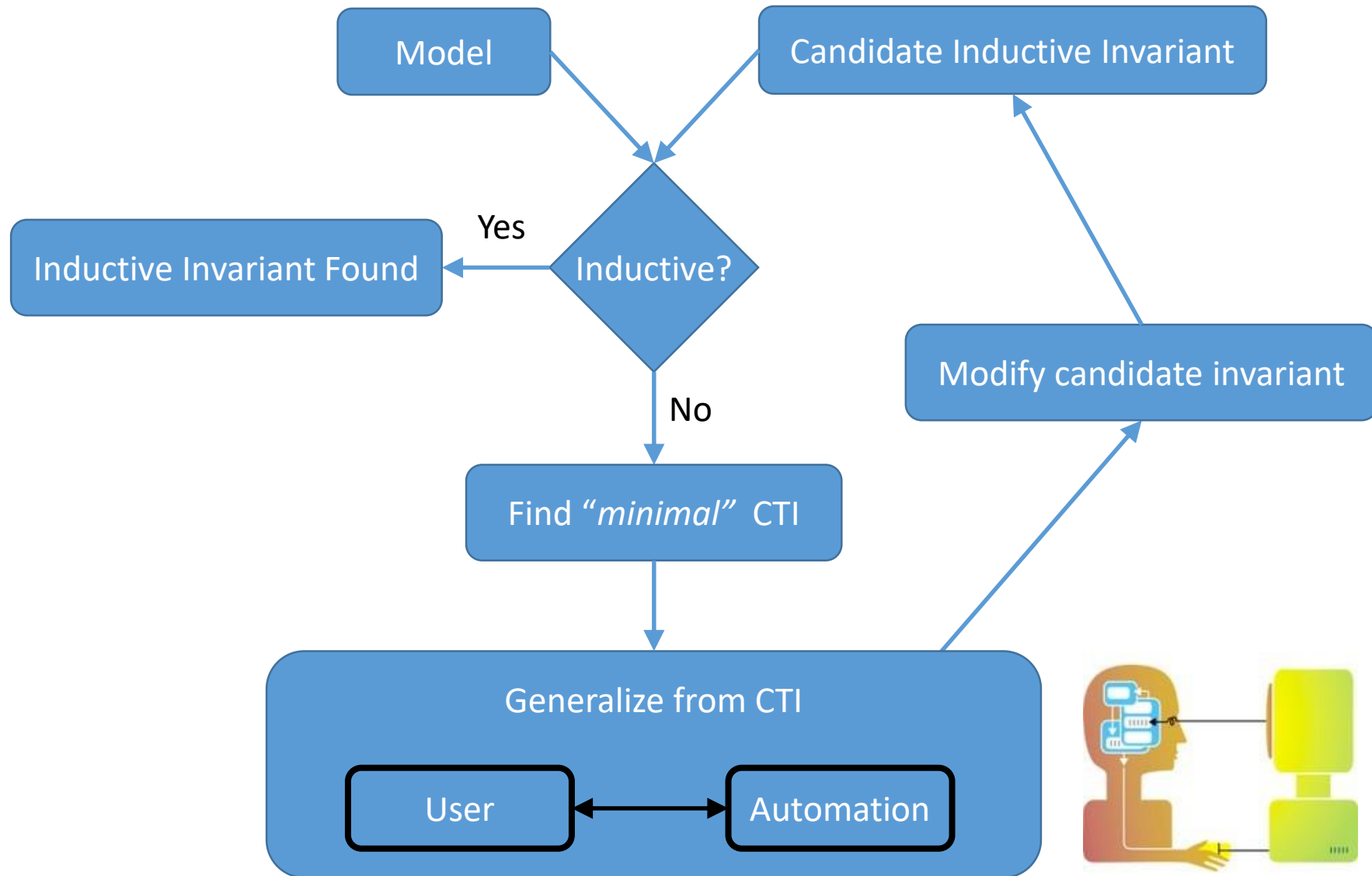
Only the leader can be self-pending

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

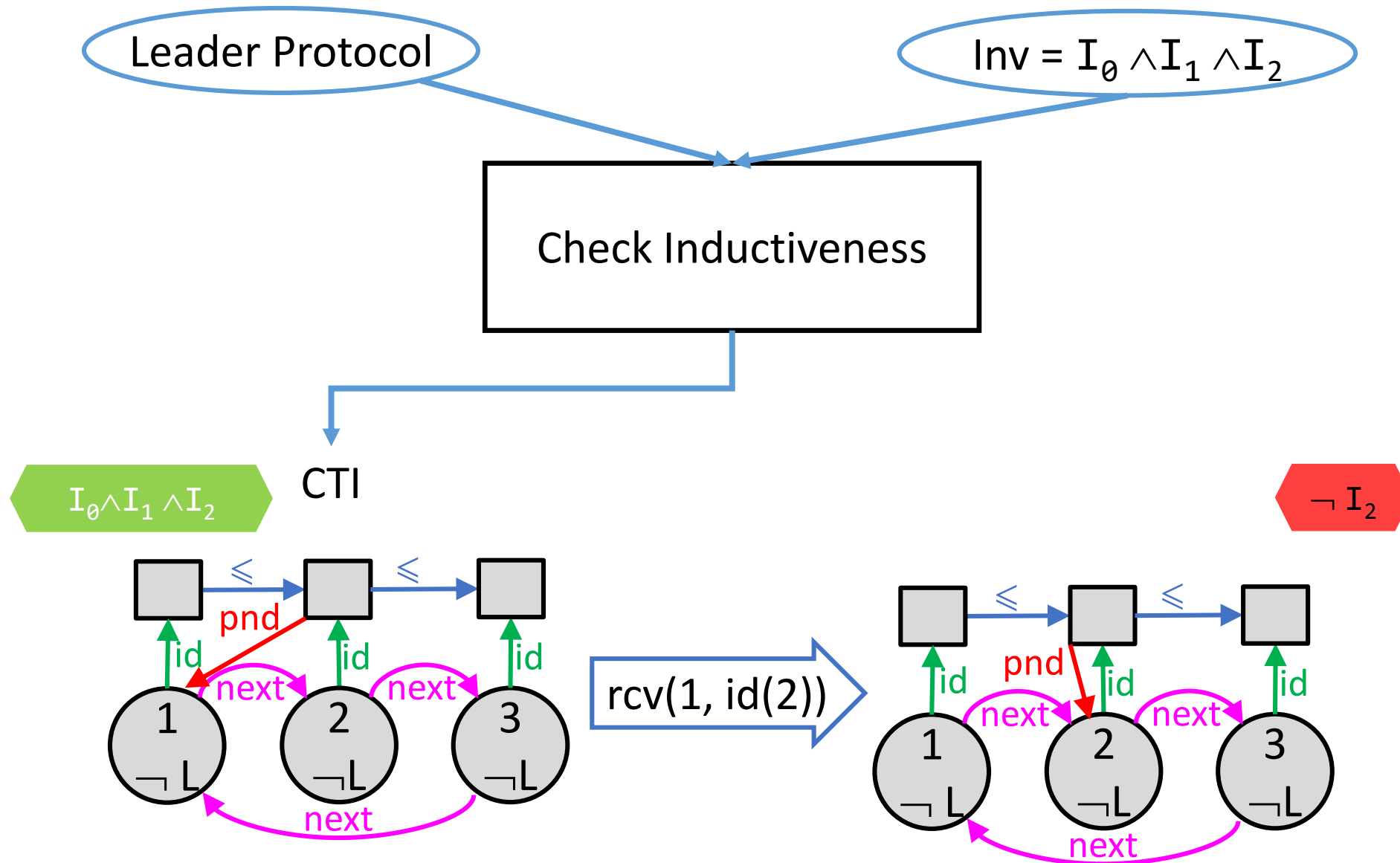
Cannot bypass higher nodes

How can we find an inductive invariant without knowing it?

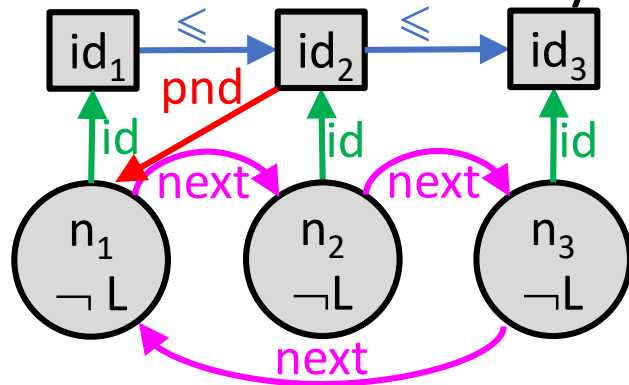
Invariant Inference in IVy



IVy: Check Inductiveness

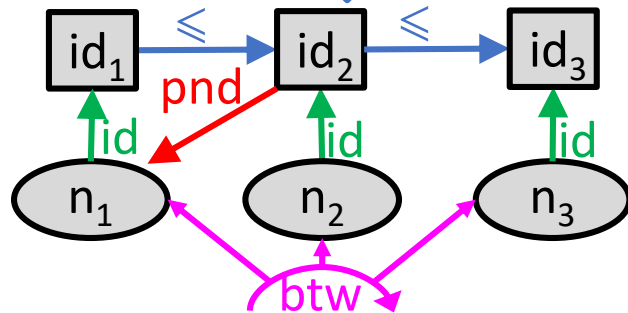


IVy: Generalize from CTI



Cannot bypass nodes with higher ids

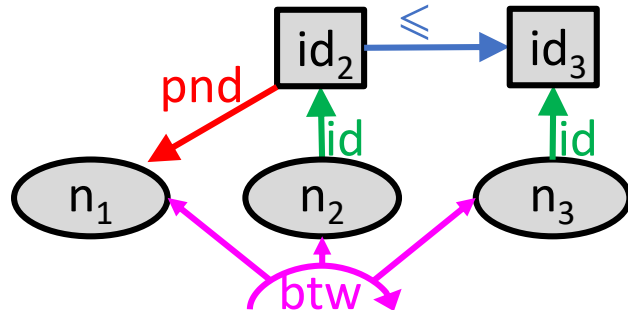
Project to {pnd, ≤, id, btw}



$$C_3 = \neg \exists n_1, n_2, n_3 : \text{Node. } \neq(n_1, n_2, n_3) \wedge \\ \neq(\text{id}[n_1], \text{id}[n_2], \text{id}[n_3]) \wedge \\ \text{id}[n_1] \leq \text{id}[n_2] \leq \text{id}[n_3] \wedge \\ \text{pnd}(\text{id}[n_2], n_1) \wedge \text{btw}(n_1, n_2, n_3)$$

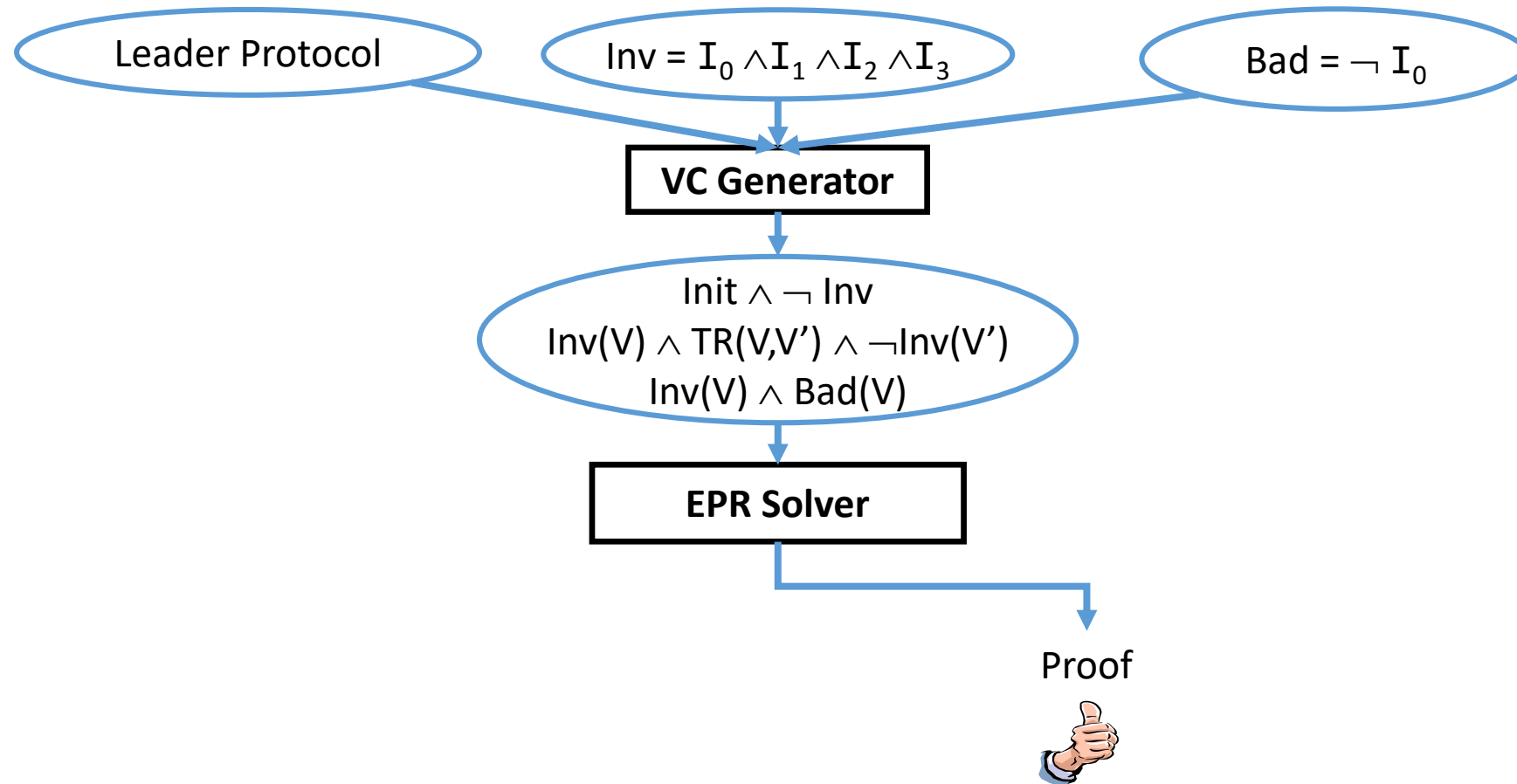
Interp(3)

This looks good, add to the invariant as I_3

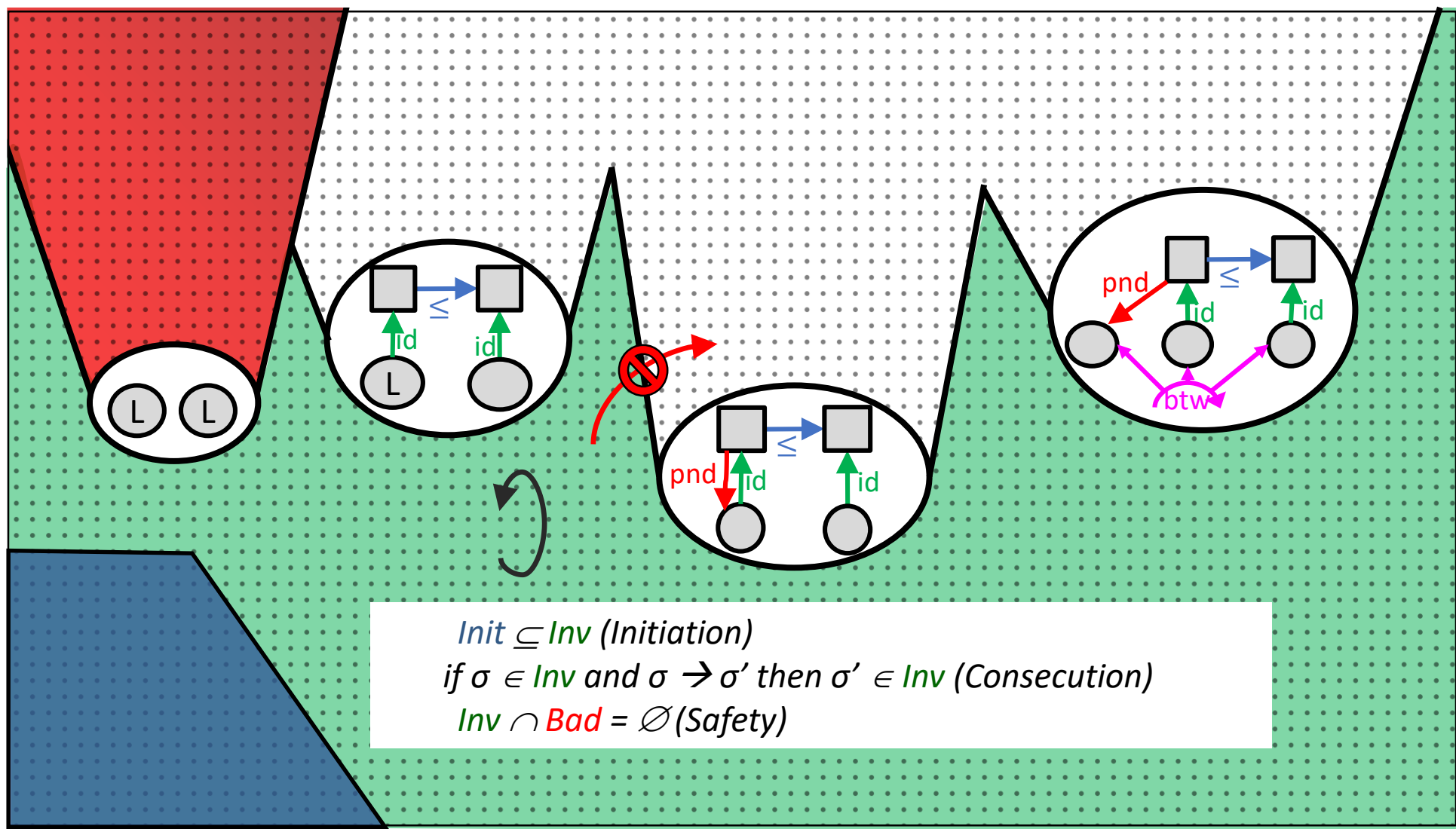


$$I_3 = \neg \exists n_1, n_2, n_3 : \text{Node. } \text{btw}(n_1, n_2, n_3) \wedge \\ \text{pnd}(\text{id}[n_2], n_1) \wedge \\ \text{id}[n_2] \leq \text{id}[n_3]$$

IVy: Check Inductiveness



$I_0 \wedge I_1 \wedge I_2 \wedge I_3$ is an inductive invariant for the leader protocol,
which proves the protocol is safe



Leader Election Protocol (axioms)

- \leq (ID, ID) – total order on node id's
- **btw** (a: Node, b: Node, c: 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

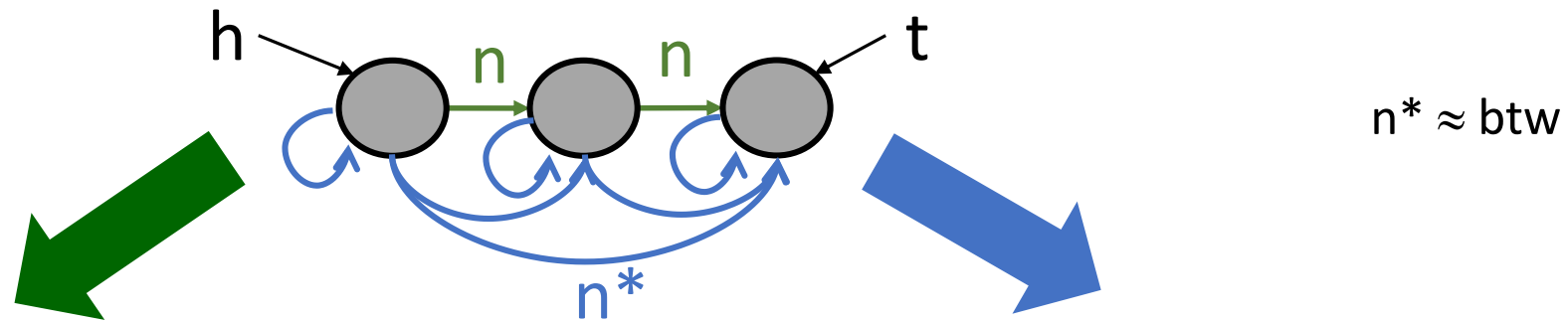
	Natural Interpretation	EPR Modeling
Node ID's	Integers	$\forall i: \text{ID}. i \leq i$ Reflexive $\forall i, j, k: \text{ID}. i \leq j \wedge j \leq k \Rightarrow i \leq k$ Transitive $\forall i, j: \text{ID}. i \leq j \wedge j \leq i \Rightarrow i = j$ Anti-Symmetric $\forall i, j: \text{ID}. i \leq j \vee j \leq i$ Total $\forall x, y: \text{Node}. \text{id}(x) = \text{id}(y) \Rightarrow x = y$ Injective
Ring Topology	Next edges + Transitive closure	$\forall x, y, z: \text{Node}. \text{btw}(x, y, z) \Rightarrow \text{btw}(y, z, x)$ Circular shifts $\forall x, y, z, w: \text{Node}. \text{btw}(w, x, y) \wedge \text{btw}(w, y, z) \Rightarrow \text{btw}(w, x, z)$ Transitive $\forall x, y, w: \text{Node}. \text{btw}(w, x, y) \Rightarrow \neg \text{btw}(w, y, x)$ A-Symmetric $\forall x, y, z, w: \text{Node}. \text{distinct}(x, y, z) \Rightarrow \text{btw}(w, x, y) \vee \text{btw}(w, y, x)$
		$\text{"next}(a) = b"$ $\equiv \forall x: \text{Node}. X = a \vee X = b \vee \text{btw}(a, b, x)$

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

- Expressing transitive closure
 - Linked lists
 - Ring protocols
- Expressing Consensus
 - Paxos, Multi-Paxos
 - Reconfiguration
 - Byzantine Fault Tolerance
- Liveness and temporal Properties

Key idea: representing deterministic paths

[Itzhaky SIGPLAN Dissertation Award 2016]

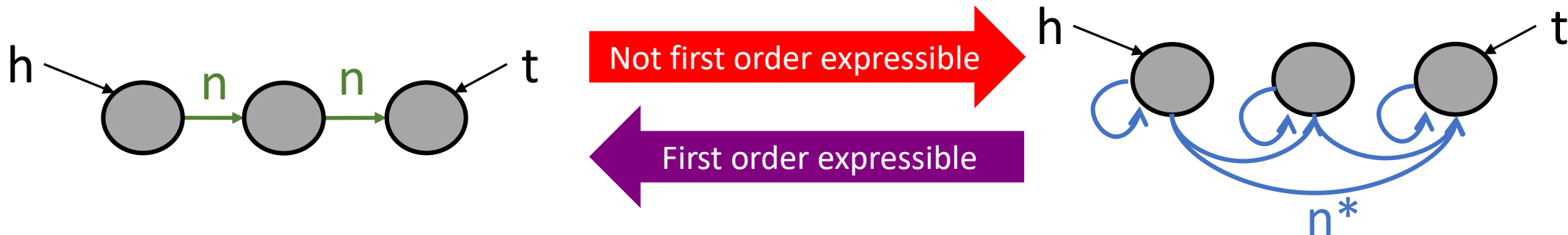


Alternative 1: maintain n

- n^* defined by transitive closure of n
- **not definable in first-order logic**

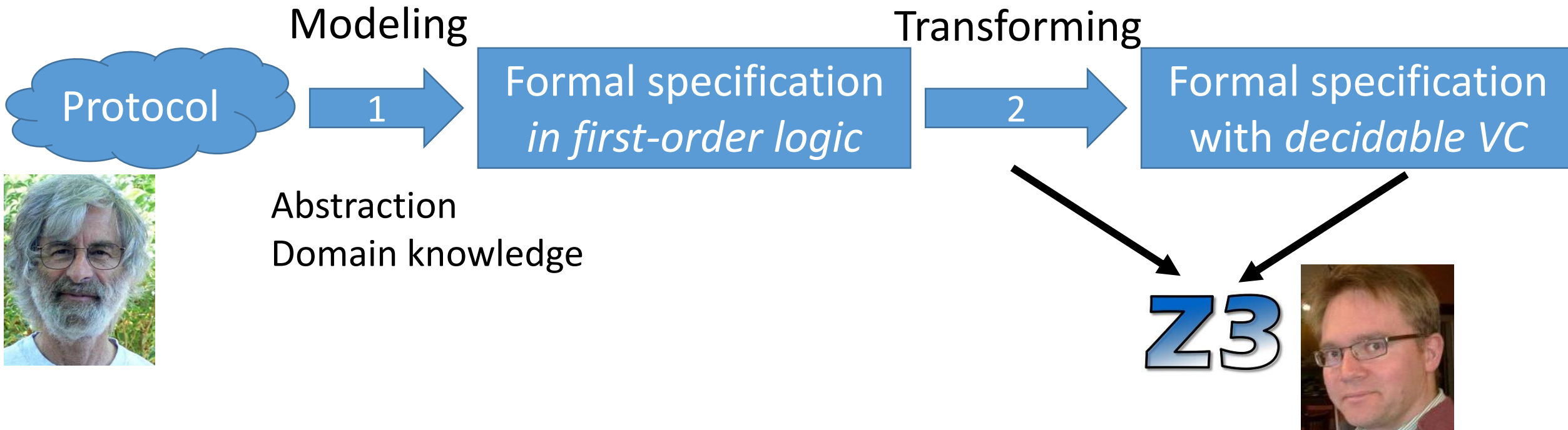
Alternative 2: maintain n^*

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



Paxos made EPR

Methodology for decidable verification of infinite-state systems



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
 - AWS
 - Many more...

Challenge: reasoning about Paxos 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!

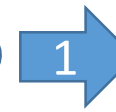
- Solution: axiomatize quorums in first-order logic
sort **quorum**
relation **member** (node, quorum)
 - set membership (2^{nd} -order logic in first-order)**axiom** $\forall q_1, q_2: \text{quorum}. \exists n: \text{node}. \text{member}(n, q_1) \wedge \text{member}(n, q_2)$

```
action propose(r:round) {  
  requires ">N/2 join_msg's"  
  ...  
}
```



```
action propose(r:round) {  
  requires  $\exists q. \forall n. \text{member}(n, q) \rightarrow$   
     $\exists r', v'. \text{join\_msg}(n, r, r', v')$   
  ...  
}
```

Principle: domain knowledge



Formal specification
in first-order logic

Concept	Intention	First-order abstraction
Quorums	Majority sets	relation <code>member</code> (node, quorum) axiom $\forall q_1, q_2: \text{quorum} \exists n: \text{node}. \text{member}(n, q_1) \wedge \text{member}(n, q_2)$
Rounds	Natural numbers	relation \leq (round, round) axiom $\forall x: \text{round}. x \leq x$ <i>reflexive</i> axiom $\forall x, y, z: \text{round}. x \leq y \wedge y \leq z \rightarrow x \leq z$ <i>transitive</i> axiom $\forall x, y: \text{round}. x \leq y \wedge y \leq x \rightarrow x = y$ <i>anti-symmetric</i> axiom $\forall x, y: \text{round}. x \leq y \vee y \leq x$ <i>total</i>
Messages	Network with: dropping duplication reordering	relation <code>start_msg</code> (round) relation <code>join_msg</code> (node, round, round, value) relation <code>propose_msg</code> (round, value) relation <code>vote_msg</code> (node, round, value)

Paxos in first-order logic

```

1 sort node, quorum, round, value
2
3 relation ≤ : round, round
4 axiom total_order(≤)
5 constant ⊥ : round
6
7 relation member : node, quorum
8 axiom ∀q1, q2 : quorum. ∃n : node. member(n, q1) ∧ member(n, q2)
9
10 relation start_round_msg : round
11 relation join_ack_msg : node, round, round, value
12 relation propose_msg : round, value
13 relation vote_msg : node, round, value
14 relation decision : node, round, value
15
16 init ∀r. ¬start_round_msg(r)
17 init ∀n, r1, r2, v. ¬join_ack_msg(n, r1, r2, v)
18 init ∀r, v. ¬propose_msg(r, v)
19 init ∀n, r, v. ¬vote_msg(n, r, v)
20 init ∀n, r, v. ¬decision(n, r, v)

```

```

21
22 action START_ROUND(r : round) {
23   assume r ≠ ⊥
24   start_round_msg(r) := true
25 }
26 action JOIN_ROUND(n : node, r : round) {
27   assume r ≠ ⊥
28   assume start_round_msg(r)
29   assume ¬∃r', r'', v. r' > r ∧ join_ack_msg(n, r', r'', v)
30   # find maximal round in which n voted, and the corresponding vote.
31   # maxr = ⊥ and v is arbitrary when n never voted.
32   local maxr, v := max {(r', v') | vote_msg(n, r', v') ∧ r' < r}
33   join_ack_msg(n, r, maxr, v) := true
34 }
35 action PROPOSE(r : round, q : quorum) {
36   assume r ≠ ⊥
37   assume ∀v. ¬propose_msg(r, v)
38   # 1b from quorum q
39   assume ∀n. member(n, q) → ∃r', v. join_ack_msg(n, r, r', v)
40   # find the maximal round in which a node in the quorum reported

```

```

41   # voting, and the corresponding vote.
42   # v is arbitrary if the nodes reported not voting.
43   local maxr, v := max {(r', v') | ∃n. member(n, q)
44                               ∧ join_ack_msg(n, r, r', v') ∧ r' ≠ ⊥}
45   propose_msg(r, v) := true # propose value v
46 }
47 action VOTE(n : node, r : round, v : value) {
48   assume r ≠ ⊥
49   assume propose_msg(r, v)
50   assume ¬∃r', r'', v'. r' > r ∧ join_ack_msg(n, r', r'', v)
51   vote_msg(n, r, v) := true
52 }
53 action LEARN(n : node, r : round, v : value, q : quorum) {
54   assume r ≠ ⊥
55   # 2b from quorum q
56   assume ∀n. member(n, q) → vote_msg(n, r, v)
57   decision(n, r, v) := true
58 }

```

$\forall n_1, n_2 : \text{node}, r_1, r_2 : \text{round}, v_1, v_2 : \text{value}. \text{decision}(n_1, r_1, v_1) \wedge \text{decision}(n_2, r_2, v_2) \rightarrow v_1 = v_2$
 $\forall r : \text{round}, v_1, v_2 : \text{value}. \text{propose_msg}(r, v_1) \wedge \text{propose_msg}(r, v_2) \rightarrow v_1 = v_2$
 $\forall n : \text{node}, r : \text{round}, v : \text{value}. \text{vote_msg}(n, r, v) \rightarrow \text{propose_msg}(r, v)$
 $\forall r : \text{round}, v : \text{value}. (\exists n : \text{node}. \text{decision}(n, r, v)) \rightarrow \exists q : \text{quorum}. \forall n : \text{node}. \text{member}(n, q) \rightarrow \text{vote_msg}(n, r, v)$
 $\forall n : \text{node}, r, r' : \text{round}, v, v' : \text{value}. \text{join_ack_msg}(n, r, \perp, v) \wedge r' < r \rightarrow \neg \text{vote_msg}(n, r', v')$
 $\forall n : \text{node}, r, r' : \text{round}, v : \text{value}. \text{join_ack_msg}(n, r, r', v) \wedge r' \neq \perp \rightarrow r' < r \wedge \text{vote_msg}(n, r', v)$
 $\forall n : \text{node}, r, r', r'' : \text{round}, v, v' : \text{value}. \text{join_ack_msg}(n, r, r', v) \wedge r' \neq \perp \wedge r' < r'' < r \rightarrow \neg \text{vote_msg}(n, r'', v')$
 $\forall n : \text{node}, v : \text{value}. \neg \text{vote_msg}(n, \perp, v)$
 $\forall r_1, r_2 : \text{round}, v_1, v_2 : \text{value}, q : \text{quorum}. \text{propose_msg}(r_2, v_2) \wedge r_1 < r_2 \wedge v_1 \neq v_2 \rightarrow$
 $\exists n : \text{node}, r', r'' : \text{round}, v : \text{value}. \text{member}(n, q) \wedge \neg \text{vote_msg}(n, r_1, v_1) \wedge r' > r_1 \wedge \text{join_ack_msg}(n, r', r'', v)$



VC's in first-order logic

Step 2: Obtaining decidable VC's

Challenge : quantifier alternation cycles

- Axiom

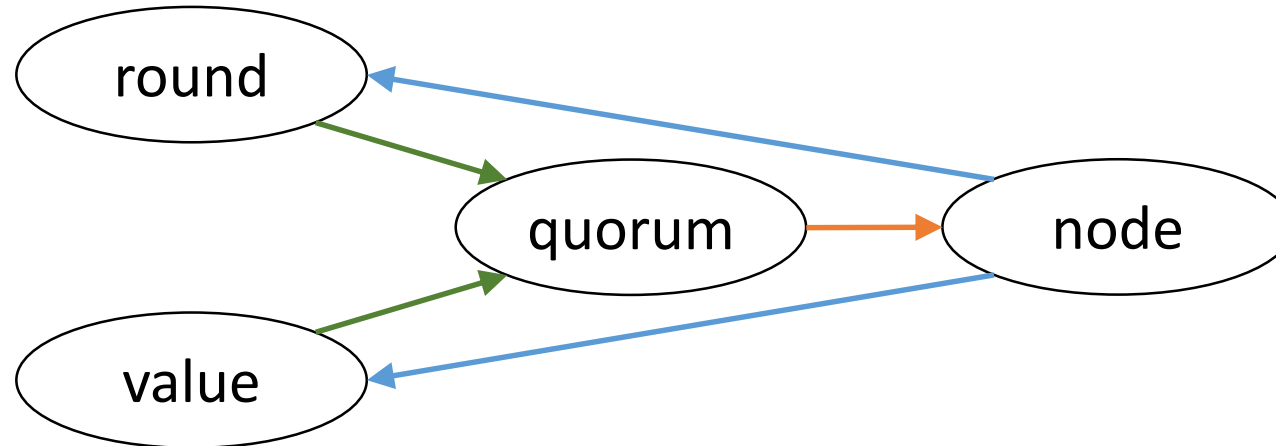
$\forall q1, q2: \text{quorum}. \exists n: \text{node}. \text{member}(n, q1) \wedge \text{member}(n, q2)$

- Propose action precondition

$\exists q: \text{quorum}. \forall n: \text{node}. \text{member}(n, q) \rightarrow \exists r': \text{round}, v': \text{value}. \text{join_msg}(n, r, r', v')$

- Inductive invariant

$\forall r: \text{round}, v: \text{value}. \text{decision}(r, v) \rightarrow \exists q: \text{quorum}. \forall n: \text{node}. \text{member}(n, q) \rightarrow \text{vote_msg}(n, r, v)$

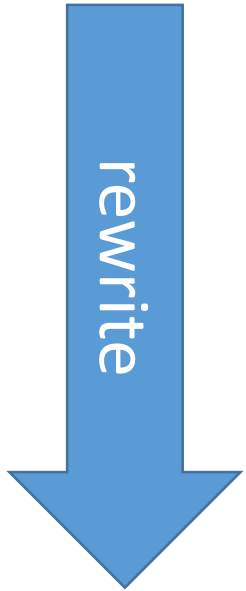


Solution: derived relations and rewrites

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$

Solution: derived relations and rewrites

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$



new relation: $\text{joined}(n:\text{node}, r:\text{round}) \equiv \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$

update code:

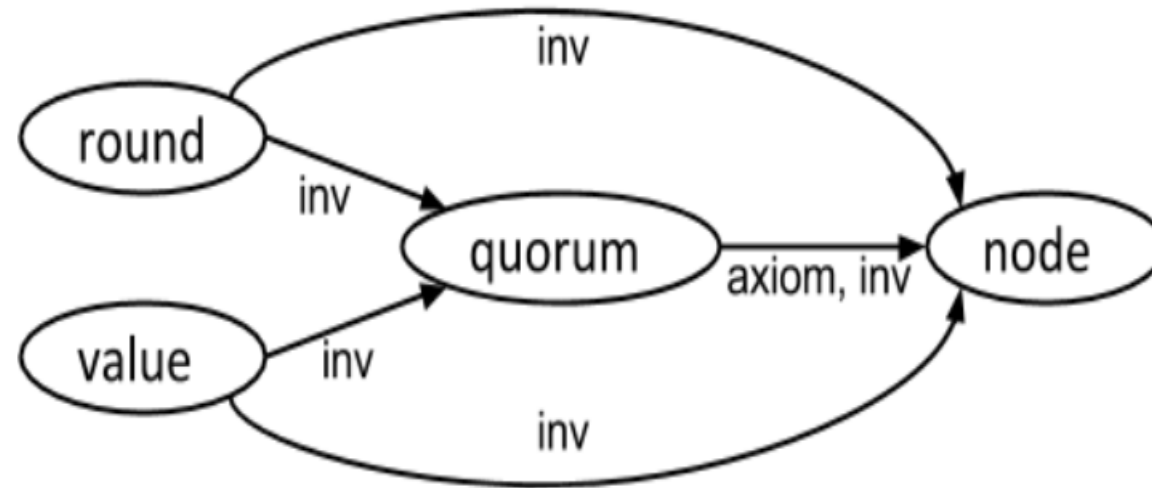
```
action join(n:node, r:round) {  
  requires start_round_msg(r)  
  let maxr,v := ...  
  join_msg(n,r,maxr,v) := true  
  joined(n,r) := true  
}
```

$\exists q:\text{quorum}. \forall n:\text{node}. \text{member}(n,q) \rightarrow \text{joined}(n,r)$

Solution: derived relations and rewrites

$\text{joined}(n:\text{node}, r:\text{round}) \equiv \exists r':\text{round}, v':\text{value}. \text{join_msg}(n, r, r', v')$

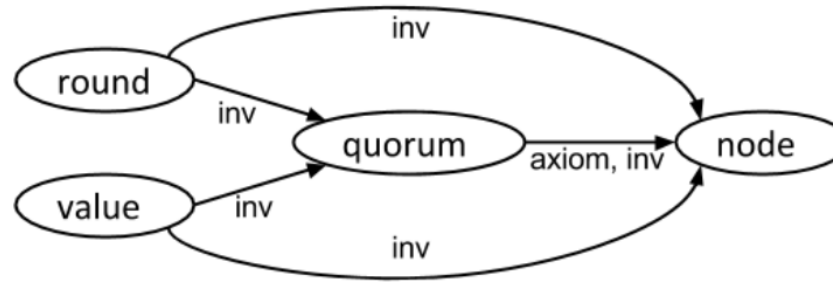
$\text{left}(n:\text{node}, r:\text{round}) \equiv \exists r', r'':\text{round}, v':\text{value}. \text{join_msg}(n, r', r'', v') \wedge r' > r$



VC's are decidable!

Principle: decomposing into decidable checks

- User defines:
 - Derived relations
 - Rewrites
 - Inductive invariants
- Decidable checks:

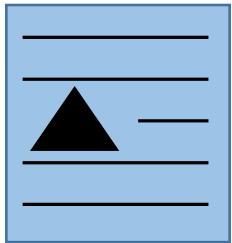


Formal specification
in first-order logic

2

Formal specification
with *decidable VC*

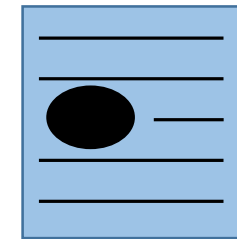
Spec in FOL



$\models \text{Inv}_{\text{aux}}$

$\text{Inv}_{\text{aux}} \models \blacktriangle \leftrightarrow \bullet$

Modified Spec



$\models \text{Inv}$

Z3

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,R0) & ~le(R0,RM) -> ~vote(N,R0,V0)

property of choosable and proposal

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)

Experimental Evaluation

Protocol	Model [LOC]	Invariant [Conjectures]	EPR [sec]		RW [sec]
			μ	σ	
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

*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 properties. For clarity and conciseness, we write simple temporal logic formulas with two temporal operators: \square meaning *always*, and \diamond meaning *eventually* [13]. We use a linear-time logic, so \square can be defined by $\neg \diamond F \triangleq \neg$ for any formula F . For a state predicate P , the formula $\square P$ asserts P is an invariant, meaning that it is true for every reachable state. Temporal formula $\square \diamond P$ asserts that at some point in the execution, P from that point onward.

We define a predicate P to be *stable* if it satisfies the following condition: if P is true in any reachable state s , then P is true in any state reached from s by any action of the algorithm. We let $\text{stable}(P)$ be the assertion that predicate P is stable. It is clear that a stable predicate is invariant; it is true in the initial state. Because stability is an assertion only over reachable states, we can assume that all invariants of the algorithm are true in state s when proving stability.

Our proofs are informal, but careful. The two complicated, multi-proofs are written with a hierarchical numbering scheme in which the number of the j^{th} step of the current level's proof [9]. Although appear intimidating, this kind of proof is easy to check and helps to errors.

A.1 The Proof of Safety

We now prove that *Consistency* and *Stopping* are invariants of Stop Paxos. First, we define:

$\text{NotChosable}(i, b, v) \triangleq$
 $(\exists Q : \forall u \in Q : (\text{bal}[u] > b) \wedge (\text{note}[u][b] \neq$
 $\vee (\exists j < i, u \in \text{StopCnd} : \text{Dose2a}(j, b, w))$
 $\vee (\forall e \in \text{StopCnd} : \exists j > i, w : \text{Dose2a}(j, b, w)))$

We next prove a number of simple invariants and its algorithm.

Lemma 1

- $\forall i, b, v : \square (\text{Chosen}(i, b, v) \Rightarrow \text{Dose2a}(i, b, v))$.
- $\forall i, b, v, w : \square ((\text{Dose2a}(i, b, v) \wedge \text{Dose2a}(i, b, w)) \Rightarrow \text{Chosen}(i, b, v) \wedge \text{Chosen}(i, b, w))$.
- $\forall i, b, v, w : \square ((\text{note}[i][b] = v) \Rightarrow \text{Dose2a}(i, b, v))$.

14

PROOF: Assume $\text{note2a}(i, b, Q) \neq \infty$. By definition of note2a , this implies $\text{note2a}(i, b, Q)$ is a command (and not \perp). Since $\text{E}(i, b, Q)$ holds by assumption (11.1), the definitions of note2a and note2a imply that some acceptor a in Q has sent a $(\text{bal}[a], b, \text{note2a}(i, b, Q), \text{note2a}(i, b, Q))$ message, which implies $\text{note}[a][\text{note2a}(i, b, Q)] = \text{note2a}(i, b, Q)$ when the message was sent. Lemma 1.3 then implies $\text{Dose2a}(i, \text{note2a}(i, b, Q), \text{note2a}(i, b, Q))$ was true when the message was sent, and is still true because $\text{Dose2a}(\dots)$ is stable.

(13) $\forall j, c < b, w : (c \leq \text{note2a}(j, b, Q)) \wedge (w \neq \text{note2a}(j, b, Q)) \Rightarrow \text{NotChosable}(j, c, w)$
PROOF: We assume $c \leq \text{note2a}(j, b, Q)$ and $w \neq \text{note2a}(j, b, Q)$, and we prove $\text{NotChosable}(j, c, w)$. Since $\text{note2a}(j, b, Q) \neq \infty$, step (1/2) implies $\text{Dose2a}(j, \text{note2a}(j, b, Q), \text{note2a}(j, b, Q))$. By assumption (11.1), this implies $\text{SafeAfter}(j, \text{note2a}(j, b, Q), \text{note2a}(j, b, Q))$. The assumption $c \leq \text{note2a}(j, b, Q)$, together with assumption (11.4) and Lemma 1.8 (which implies $\text{note2a}(j, b, Q) < b$), implies $c < b$. The assumption $w \neq \text{note2a}(j, b, Q)$ and $\text{SafeAfter}(j, \text{note2a}(j, b, Q), \text{note2a}(j, b, Q))$ then imply $\text{NotChosable}(j, c, w)$.

(14) $\forall j, c < b, w : (\text{note2a}(j, b, Q) = \perp) \Rightarrow \text{NotChosable}(j, c, w)$
PROOF: We assume $c < b$ and $\text{note2a}(j, b, Q) = \perp$ and prove $\text{NotChosable}(j, c, w)$. We split the proof into two cases.

(2.1) CASE: $\text{note2a}(j, b, Q) = \infty$
PROOF: The case assumption implies $\text{note2a}(j, b, Q) < c$, so assumption (11.4) and Lemma 3 imply $\text{NotChosable}(j, c, w)$.

(2.2) CASE: $\text{note2a}(j, b, Q) \neq \infty$
PROOF: Since $c < b$, we can split the proof into the following three cases.

(3.1) CASE: $\text{note2a}(j, b, Q) < c < b$
PROOF: By assumption (11.4), the case assumption and Lemma 3 imply $\text{NotChosable}(j, c, w)$.

(3.2) CASE: $c \leq \text{note2a}(j, b, Q)$ and $w \neq \text{note2a}(j, b, Q)$
PROOF: By (13).

(3.3) CASE: $c \leq \text{note2a}(j, b, Q)$ and $w = \text{note2a}(j, b, Q)$
 $(\exists i, \text{note2a}(i, b, Q) \in \text{StopCnd}$ and we can choose $k > j$ such that $\text{note2a}(i, b, Q) \geq \text{note2a}(j, b, Q)$.

PROOF: We define that $\text{note2a}(i, b, Q) \in \text{StopCnd}$ and such a k exists by the (2.2) case assumption, the assumption $\text{note2a}(j, b, Q) = \perp$, and the definition of note2a .

(4.2) $\text{Dose2a}(i, \text{note2a}(i, b, Q), \text{note2a}(i, b, Q))$
PROOF: The (3.3) case assumption and (4.1) imply $\text{note2a}(i, b, Q) \neq \infty$. Step (1/2) then proves (4.2).

(4.3) $\text{NotChosable}(j, c, w)$
PROOF: Assumption (11.1) with $j - k, c = \text{note2a}(i, b, Q)$, and $w = \text{note2a}(i, b, Q)$ and (4.2) imply $\text{NotChosable}(j, c, w)$. By Lemma 1.3, this implies $\text{Dose2a}(i, c, w)$. By (4.1) and (4.2), case assumption (3.3) and (4.1) imply

18

Stoppable Paxos

Dahlia Malkhi

Leslie Lamport

Lidong Zhou

April 28, 2008

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

$c \leq \text{note2a}(i, b, Q)$; and (4.1) and case assumption (3.3) imply $w \in \text{StopCnd}$. Therefore, $\text{NotChosable}(j, c, w)$ implies $\text{NotChosable}(j, c, w)$.

(15) $\text{SafeAfter}(i, b, v)$
PROOF: We assume $c < b$ and $w \neq v$ and prove $\text{NotChosable}(i, c, w)$. By Lemma 1.7, it suffices to prove $\text{NotChosable}(i, c, w)$. We split the proof into two cases.

(2.1) CASE: $\text{note2a}(i, b, Q) = \perp$
PROOF: (1/4) (substituting $j = i$) implies $\text{NotChosable}(i, c, w)$.

(2.2) CASE: $\text{note2a}(i, b, Q) \neq \perp$
PROOF: Since $c < b$, we can break the proof into two sub-cases.

(3.1) CASE: $\text{note2a}(i, b, Q) < c < b$
PROOF: Assumption (11.4) and Lemma 3 imply $\text{NotChosable}(i, c, w)$.

(3.2) CASE: $c \leq \text{note2a}(i, b, Q)$
PROOF: Assumption (11.3) implies $\text{E}(i, b, Q, v)$. Case assumption (2.2) and $\text{E}(i, b, Q, v)$ imply $v = \text{note2a}(i, b, Q)$. Case assumption (2.2) and the definition of note2a then imply $v = \text{note2a}(i, b, Q)$. Case assumption (3.3), the assumption $w \neq v$, and step (1/3) (substituting $j = i$) then imply $\text{NotChosable}(i, c, w)$.

(16) $\text{NotChosable}(j, c, w)$
PROOF: We assume $j < i, w \in \text{StopCnd}$, and $c \leq b$ and we prove $\text{NotChosable}(j, c, w)$. By Lemma 1.7, it suffices to prove $\text{NotChosable}(j, c, w)$. Since $c < b$, we need consider only the following two cases.

(2.1) CASE: $b = c$
PROOF: Assumption (11.3) implies $\text{Dose2a}(i, b, v)$. Since $i > j$ and $w \in \text{StopCnd}$, this implies the third disjunct of $\text{NotChosable}(j, b, w)$ (substituting i and v for the existentially quantified variables), which by the case assumption proves $\text{NotChosable}(j, c, w)$.

(2.2) CASE: $c < b$
PROOF: We consider two sub-cases.

(3.1) CASE: $\text{note2a}(j, b, Q) = \perp$
PROOF: (1/4) and case assumption (2.2) imply $\text{NotChosable}(j, c, w)$.

(3.2) CASE: $\text{note2a}(j, b, Q) \neq \perp$
PROOF: By case assumption (2.2), we have the following two sub-cases.

(4.1) CASE: $\text{note2a}(j, b, Q) < c < b$
PROOF: Assumption (11.4), the case assumption, and Lemma 3 imply $\text{NotChosable}(j, c, w)$.

(4.2) CASE: $c \leq \text{note2a}(j, b, Q)$
PROOF: Assumption (11.3) implies $\text{E}(i, b, Q)$. The (3.2) case assumption, the assumption $j < i$, and $\text{E}(i, b, Q)$ imply $\text{note2a}(j, b, Q) \in \text{StopCnd}$. Case assumption (2.2) and the definition of note2a then imply $\text{note2a}(j, b, Q) < \text{note2a}(i, b, Q)$ for all $i > j$.

(17) $\text{NotChosable}(i, c, w)$
PROOF: (4.1) and case assumption (3.4) imply $\text{note2a}(j, b, Q) < c < b$. By assumption (11.4), Lemma 3 implies $\text{NotChosable}(i, c, w)$.

Theorem 1 \square *Consistency*
PROOF: By definition of *Consistency*, it suffices to assume $\text{Chosen}(i, b, v)$ and $\text{Chosen}(i, c, w)$ and to prove $v = w$. Without loss of generality, we can assume $b \leq c$. We then have two cases.

1. CASE: $b = c$
PROOF: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and $\text{Chosen}(i, c, w)$ imply $\text{Dose2a}(i, c, w)$. By Lemma 4, this implies

tion of note2a , we then have $w \neq \text{note2a}(j, b, Q)$. The (4.2) case assumption (which implies $\text{note2a}(j, b, Q) \neq \infty$) and (1/3) then imply $\text{NotChosable}(j, c, w)$.

(17) $\text{NotChosable}(i, c, w)$
PROOF: We assume $v \in \text{StopCnd}$, $j > i$, $c < b$, and w any command and we prove $\text{NotChosable}(j, c, w)$. By Lemma 1.7, it suffices to prove $\text{NotChosable}(j, c, w)$. We split the proof into two cases.

(2.1) CASE: $\text{note2a}(i, b, Q) = \perp$
PROOF: Assumption (11.3) implies $\text{E}(i, b, Q, v)$. The case assumption $v \in \text{StopCnd}$ implies $\text{E}(i, b, Q, v)$. The case assumption, the assumption $j > i$, and $\text{E}(i, b, Q, v)$ imply $\text{note2a}(j, b, Q) = \perp$. The assumption $c < b$ and step (1/4) then imply $\text{NotChosable}(j, c, w)$.

(2.2) CASE: $\text{note2a}(i, b, Q) \neq \perp$
PROOF: Assumption (11.3) implies $\text{E}(i, b, Q, v)$, which implies $\text{note2a}(i, b, Q) = v$. The case assumption and the definition of note2a then imply $\text{note2a}(i, b, Q) = v$.

(3.2) $\text{Dose2a}(i, \text{note2a}(i, b, Q), \text{note2a}(i, b, Q))$
PROOF: (2.1), assumption (11.4), and the definition of note2a imply $\text{note}[a][\text{note2a}(i, b, Q)] = v$ for some acceptor a in Q , which by Lemma 1.3 implies $\text{Dose2a}(i, \text{note2a}(i, b, Q), v)$.

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

(3.3) CASE: $c < \text{note2a}(i, b, Q)$
PROOF: Step (3/2) and assumption (11.1) imply $\text{NotChosable}(i, c, w)$. By the case assumption and the assumption $v \in \text{StopCnd}$ and $j > i$, this implies $\text{NotChosable}(j, c, w)$.

(3.4) CASE: $\text{note2a}(i, b, Q) \leq c < b$
 $(\exists i, \text{note2a}(i, b, Q) < \text{note2a}(j, b, Q))$
PROOF: The assumption $v \in \text{StopCnd}$ and (3.1) imply $\text{note2a}(i, b, Q) \in \text{StopCnd}$. Case assumption (2.2) and the definition of note2a then imply $\text{note2a}(i, b, Q) < \text{note2a}(j, b, Q)$ for all $i > j$.

(18) $\text{NotChosable}(i, c, w)$
PROOF: (4.1) and case assumption (3.4) imply $\text{note2a}(j, b, Q) < c < b$. By assumption (11.4), Lemma 3 implies $\text{NotChosable}(i, c, w)$.

Theorem 3 \square *Consistency*
PROOF: By definition of *Consistency*, it suffices to assume $\text{Chosen}(i, b, v)$ and $\text{Chosen}(i, c, w)$ and to prove $v = w$. Without loss of generality, we can assume $b \leq c$. We then have two cases.

1. CASE: $b = c$
PROOF: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and $\text{Chosen}(i, c, w)$ imply $\text{Dose2a}(i, c, w)$. By Lemma 4, this implies

some more definitions, culminating in the key invariant

$\triangleq \forall e < b, w \neq v : \text{NotChosable}(i, c, w)$
 $\triangleq v(i, b) \triangleq$
 $b, w \in \text{StopCnd} : \text{NotChosable}(j, c, w)$
 $\triangleq (\text{for}(i, b, v) \triangleq$
 $\text{nd}) \Rightarrow \forall j > i, c < b, w : \text{NotChosable}(j, c, w)$
 $\triangleq \text{Dose2a}(i, b, v) \Rightarrow \text{SafeAfter}(i, b, v)$
 $\wedge \text{NotChosable}(i, b, v)$
 $\wedge \text{NotChosable}(i, b, v)$

safety proof is the following proof that *Prophecy* is invariant.

$i, b, v : \text{Prophecy}(i, b, v)$
 $\text{Prophecy}(i, b, v)$ is true in the initial state because $\text{Dose2a}(\dots)$ is true. We can therefore assume $s \rightarrow i$ is a quorum. Formulas $\text{E}(i, b, Q)$ holds because i is $\text{note2a}(i, b, v, Q)$ action.
 NOTE clause of (1) are proved as steps (1/5), e steps are used in their proofs.
 $\triangleq \text{Dose2a}(j, \text{note2a}(j, b, Q), \text{note2a}(j, b, Q))$
 $\triangleq \text{Dose2a}(j, \text{note2a}(j, b, Q), \text{note2a}(j, b, Q))$

17

$\text{SafeAfter}(i, c, w)$. The assumptions $b < c$, an $v \neq w$ then imply $\text{NotChosable}(i, c, w)$. By Lemma 2, this contradicts the assumption $\text{Chosen}(i, b, v)$.

2. CASE: $b = c$
PROOF: Lemma 1.1 implies $\text{Dose2a}(i, b, v) \wedge \text{Dose2a}(i, c, w)$, which by Lemma 1.2 implies $b = c$.

Theorem 2 \square Stopping

PROOF: By definition of *Stopping*, it suffices to assume $\text{Chosen}(i, b, v)$, $\text{Chosen}(j, c, w)$, $v \in \text{StopCnd}$, and $j > i$ and to obtain a contradiction. We split the proof into two cases.

1. CASE: $c < b$
PROOF: $\text{Chosen}(i, b, v)$ and Lemma 1.1 imply $\text{Dose2a}(i, b, v)$. This and Lemma 4 imply $\text{NotChosable}(i, b, v)$, which by the case assumption and the assumption $v \in \text{StopCnd}$ and $j > i$ implies $\text{NotChosable}(j, c, w)$. The assumption $\text{Chosen}(j, c, w)$ and Lemma 2 then provide the required contradiction.

2. CASE: $c \geq b$
PROOF: $\text{Chosen}(j, c, w)$ and Lemma 1.1 imply $\text{Dose2a}(j, c, w)$. Lemma 4 then implies $\text{NotChosable}(j, c, w)$. The case assumption, the assumption $v \in \text{StopCnd}$ and $j > i$ implies $\text{NotChosable}(i, b, v)$. The assumption $\text{Chosen}(i, b, v)$ and Lemma 2 then provide the required contradiction.

A.2 The Proof of Progress.

Theorem 3 $\forall b, Q : \text{Progress}(b, Q)$

PROOF: We assume $\text{P}(b, Q)$, $\text{P}(b, Q)$ and $\text{P}(b, Q)$ and we must prove that there exists a v such that either $\square \text{Chosen}(i, b, v)$ or $v \in \text{StopCnd} \wedge \square \text{Chosen}(i, b, v)$ for some $j < i$.

(11) $\square \text{Chosen}(i, b, v)$
PROOF: $\text{P}(b, Q)$ implies that the ballot b leader eventually executes a $\text{Phase}(b)$ action. By $\text{P}(b, Q)$, every acceptor a in Q eventually receives the $\text{Phase}(b)$ message. Because $\text{bal}[a]$ is set to a value v only by receiving a ballot v message, assumption $\text{P}(b)$ implies $\text{bal}[a] \leq b$. Hence, a must eventually receive the $\text{Phase}(b)$ message and execute $\text{Phase}(b)$. By $\text{P}(b, Q)$, the $\text{Phase}(b)$ message it sends is eventually received by the leader.

(12) $\forall i, v : \square (\text{Dose2a}(i, b, v) \Rightarrow \square \text{Chosen}(i, b, v))$
PROOF: $\text{Dose2a}(i, b, v)$ means that a $\text{Phase}(b)$ action has been executed sending a $(\text{bal}[i], b, v)$ message to every acceptor a . If a is in Q , then assumption $\text{P}(b)$ implies that it eventually receives that message. Assumption $\text{P}(b)$ implies $\text{bal}[a] \leq b$. By $\text{P}(b, Q)$ implies that every a in Q eventually executes $\text{Phase}(b)$, a, b, v , setting $\text{note}[a][b]$ to v . Hence, eventually $\text{Chosen}(i, b, v)$ becomes true.

21

(1)7. *NoneChoosableAfter*(i, b, v)'

PROOF: We assume $v \in \text{StopCmd}$, $j > i$, $c < b$, 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: $\text{sval2a}(i, b, Q) = \top$

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

(2)2. CASE: $\text{sval2a}(i, b, Q) \neq \top$

(3)1. $\text{sval2a}(i, b, Q) = \text{val2a}(i, b, Q) = v$

PROOF: Assumption (1)1.3 implies $E3(i, b, Q, v)$, which implies $\text{sval2a}(i, b, Q) = v$. The case assumption and the definition of sval2a then implies $\text{val2a}(i, b, Q) = v$.

(3)2. *Done2a*($i, \text{mbal2a}(i, b, Q), v$)

PROOF: (3)1, assumption (1)1.4, and the definition of val2a imply $\text{vote}_i[a][\text{mbal2a}(i, b, Q)] = v$ for some acceptor a in Q , which by Lemma 1.3 implies *Done2a*($i, \text{mbal2a}(i, b, Q), v$).

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

(3)3. CASE: $c < \text{mbal2a}(i, b, Q)$

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

(3)4. CASE: $\text{mbal2a}(i, b, Q) \leq c < b$

(4)1. $\text{mbal2a}(j, b, Q) < \text{mbal2a}(i, b, Q)$

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

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

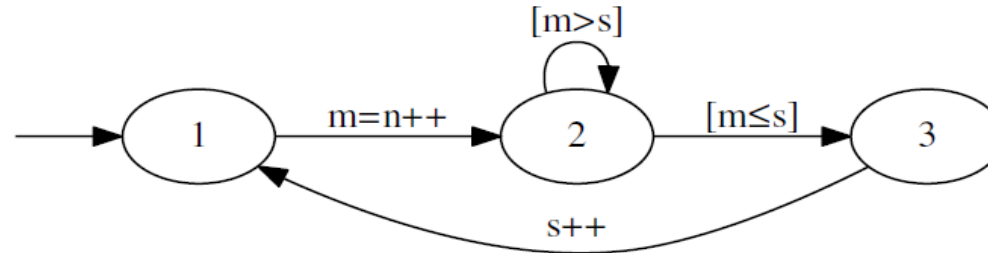
PROOF: (4)1 and case assumption (3)4 imply $\text{mbal2a}(j, b, Q) < c < b$. By assumption (1)1.4, Lemma 3 implies *NotChoosable*(j, c, w). \square

Verification of Temporal Properties

```

global nat s, n
local nat m
1: while (true) {
    m=n++; // Acquire a ticket
2:   while (m>s) { // Busy wait
        skip;
    }
    // Critical section
3:   s++; // Exit critical
    }

```



Liveness Property	$\forall x : \text{thread. } \Box (pc_2(x) \rightarrow \Diamond pc_3(x))$
Fairness Assumption	$\forall x : \text{thread. } \Box \Diamond \text{ scheduled}(x)$
Temporal Spec. (<i>spec</i>)	$(\forall x : \text{thread. } \Box \Diamond \text{ scheduled}(x)) \rightarrow \forall x : \text{thread. } \Box (pc_2(x) \rightarrow \Diamond pc_3(x))$

Possible Projects

- Verify any distributed / shared memory algorithm
- Paxos variants
 - Disk Paxos, Generalised Paxos, EPaxos (see <http://paxos.systems/variants.html> for ideas)
 - Prove reconfiguration / failure recovery / log truncation / liveness
- Mutual Exclusion Algorithms
 - Knuth's Algorithm, Lamport's Bakery, Patterson, ...
 - Prove safety and liveness
- Blockchain algorithms
 - Algorand, HoneyBadgerBFT, Bitcoin-NG, ...
- Improve Ivy
 - Experiment with other SMT solvers (e.g. iProver, CVC4, Vampire, SPASS)