

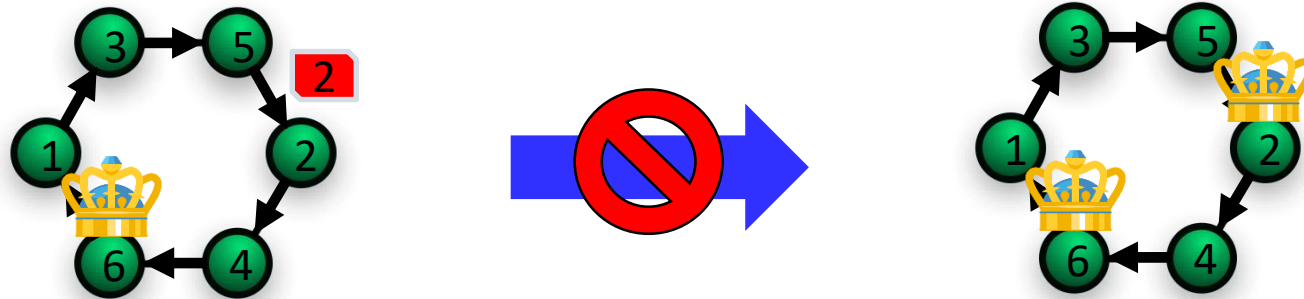
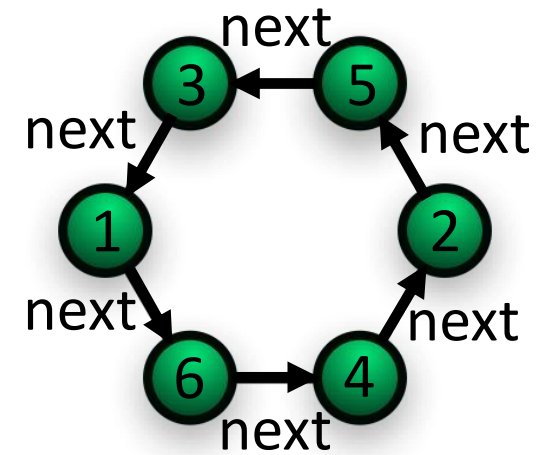
Effectively Propositional Logic

Mooly Sagiv



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**

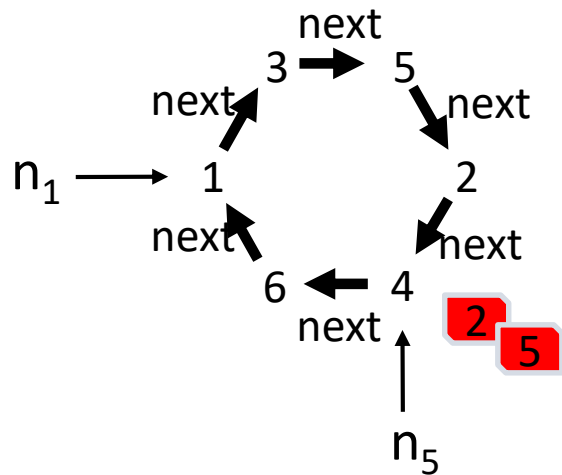


Leader election protocol – first-order logic

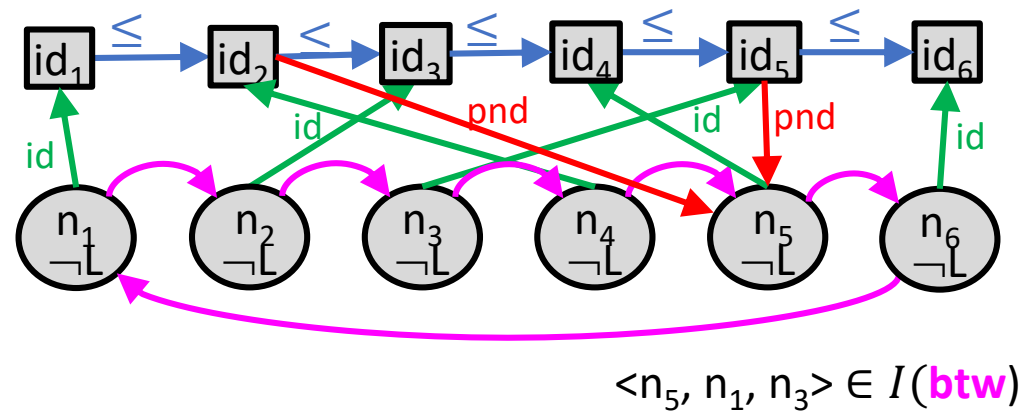
- \leq (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node \rightarrow ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



first-order structure

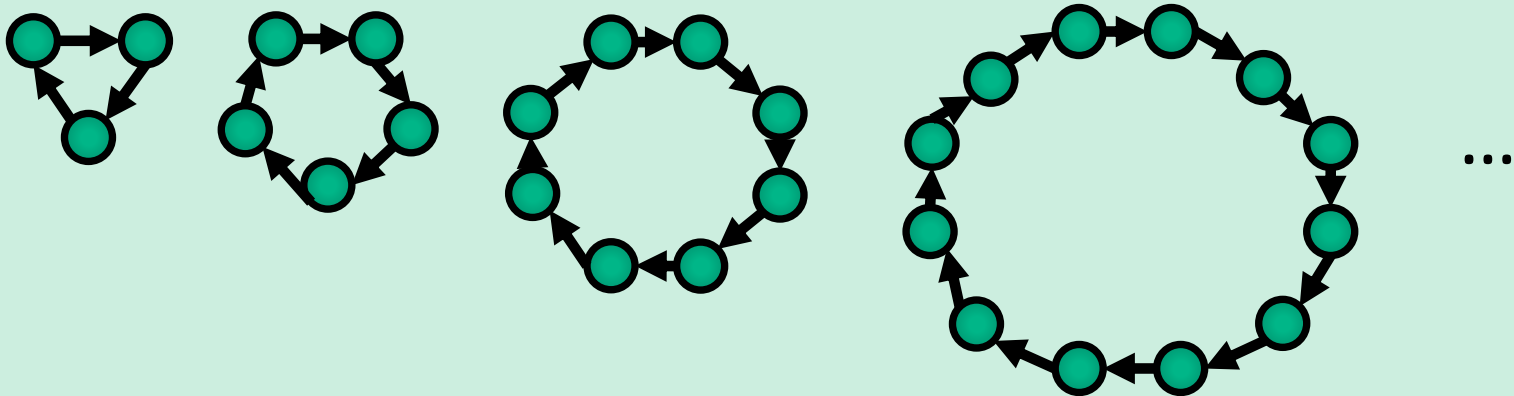


Leader election protocol – first-order logic

- \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

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



Leader election protocol – first-order logic

- \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  
}
```

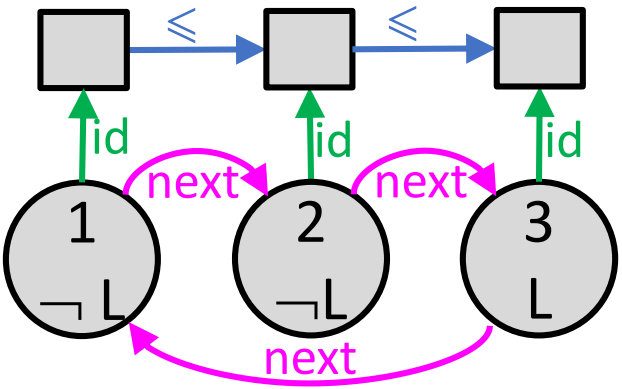
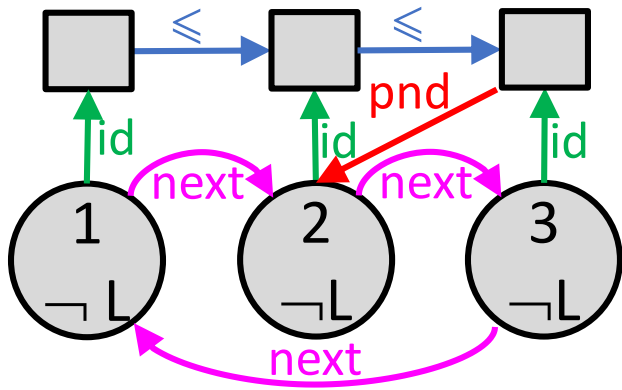
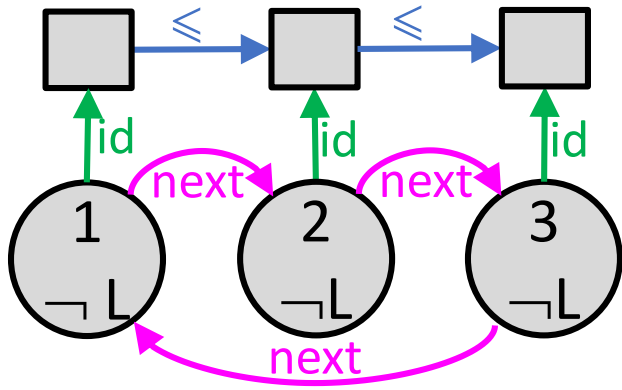
```
action receive(n: Node, m: ID) = {  
  requires pending(m, n);  
  if id(n) = m then  
    // found leader  
    leader(n) := true  
  else if id(n)  $\leq$  m then  
    // pass message  
    "s := next(n)";  
    pending(m, s) := true  
}
```

TR(send):

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

Bad:

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

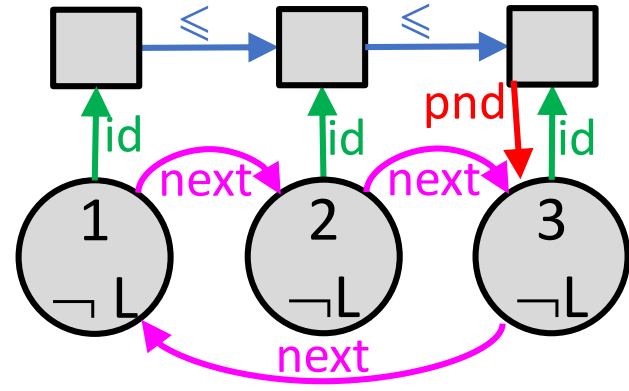
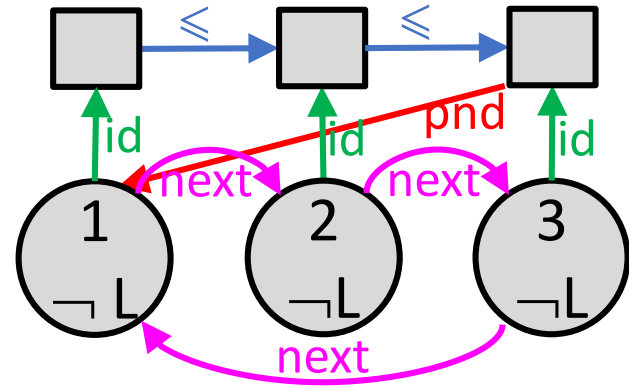


send(3)

rcv(1, id(3))

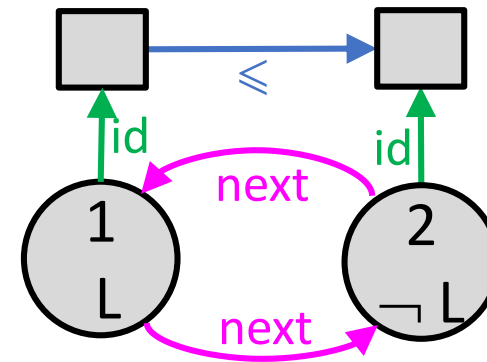
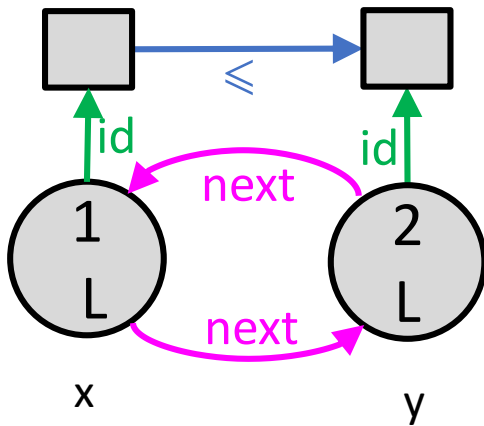
rcv(2, id(3))

rcv(3, id(3))



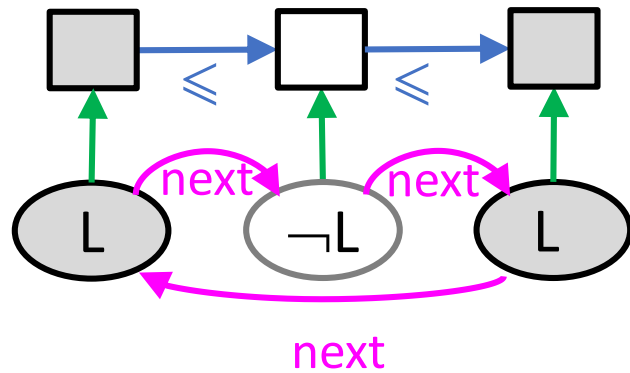
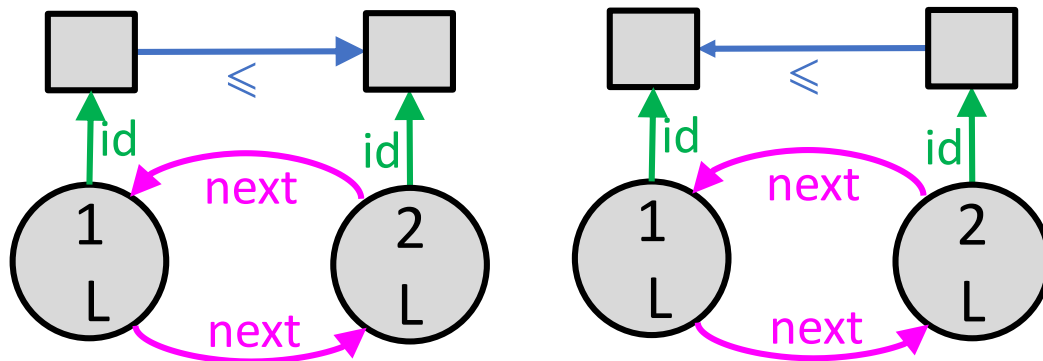
Representing Sets of States with First Order Formulas

- Configurations with at least two leaders
 - $\exists X, Y: \text{Node}. \mathbf{leader}(X) \wedge \mathbf{leader}(Y) \wedge X \neq Y$



Representing Sets of States with First Order Formulas

- Configurations with at least two leaders
 - $\exists X, Y: \text{Node}. \text{leader}(X) \wedge \text{leader}(Y) \wedge X \neq Y$



...

Leader election protocol – inductive invariant

Safety property: I_0

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

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

$$I_1 = \forall n_1$$

$$I_2 = \forall n_1$$

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

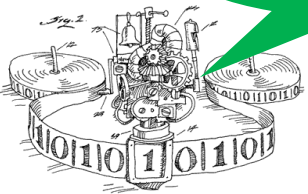
How can we find an inductive invariant without knowing it?

?

can be self-pending

I can decide EPR!

cannot bypass higher nodes



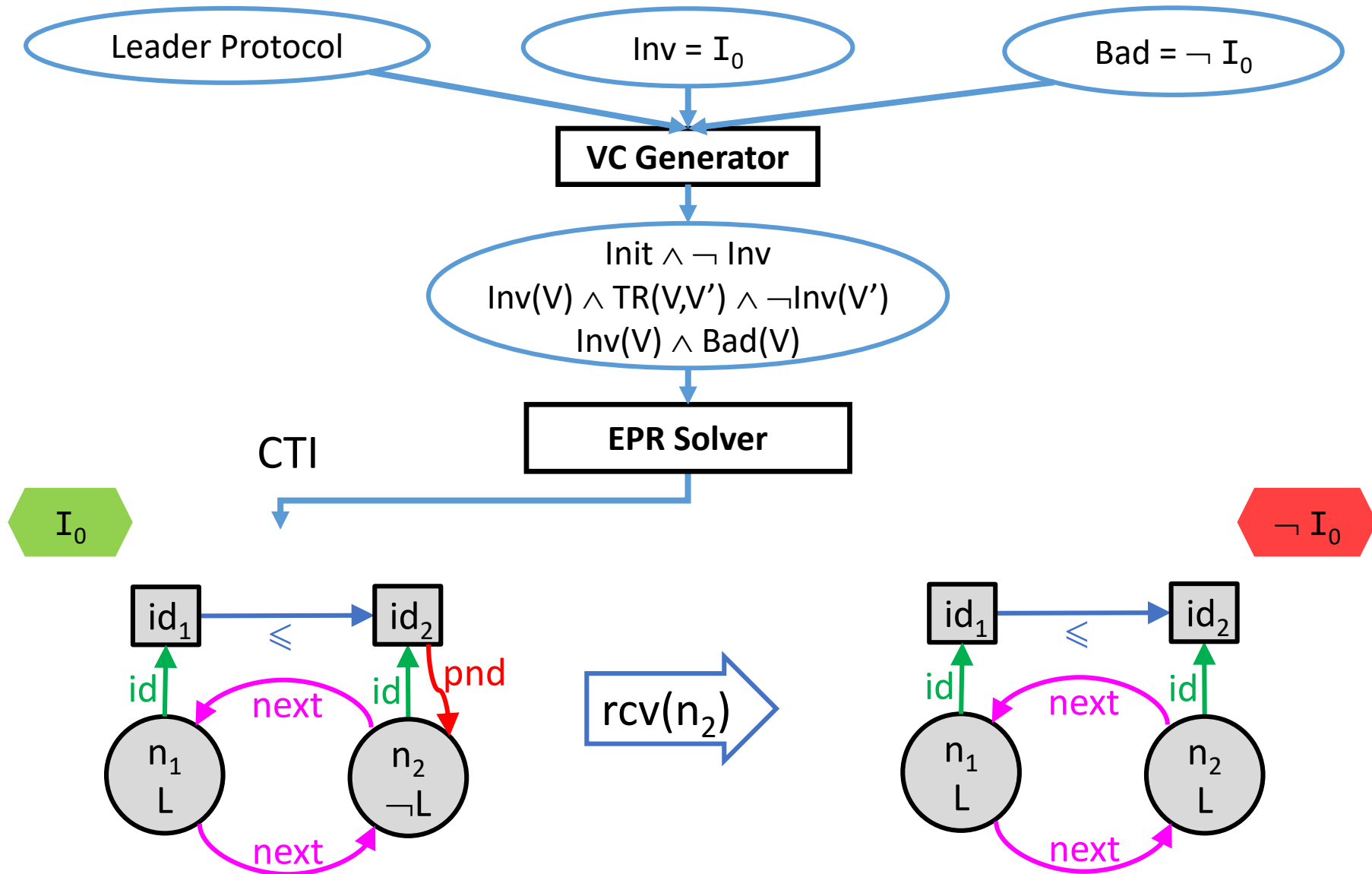
- $\leq (ID, ID)$ – total order on node id's
- **VC Generator** – $\text{Init}(V) \wedge \neg \text{Inv}(V) \rightarrow \text{Env}(V) \wedge \neg \text{Inv}(V')$
- **id**: Node \rightarrow ID – relate a node to its ID
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

EPR Solver

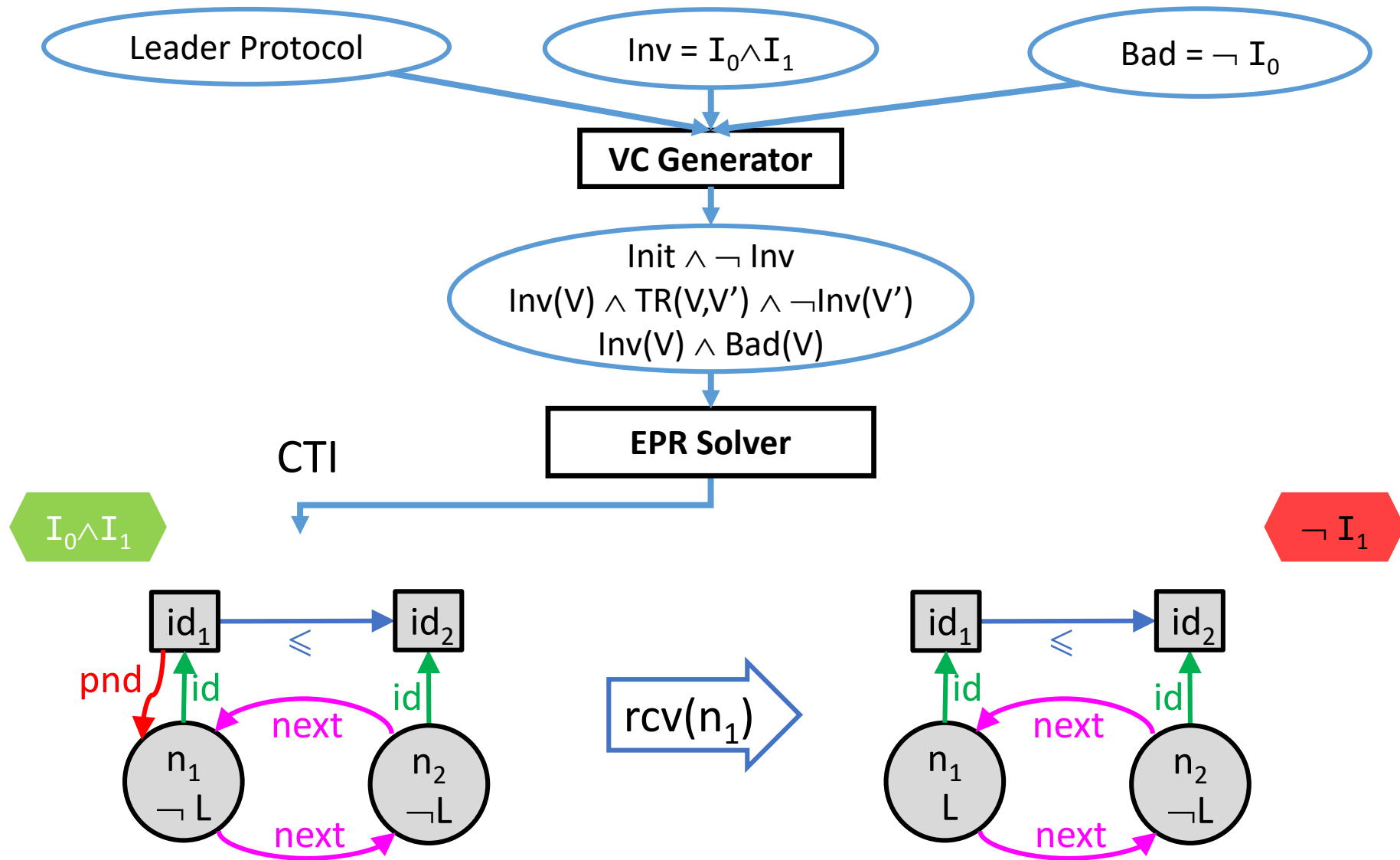


Proof

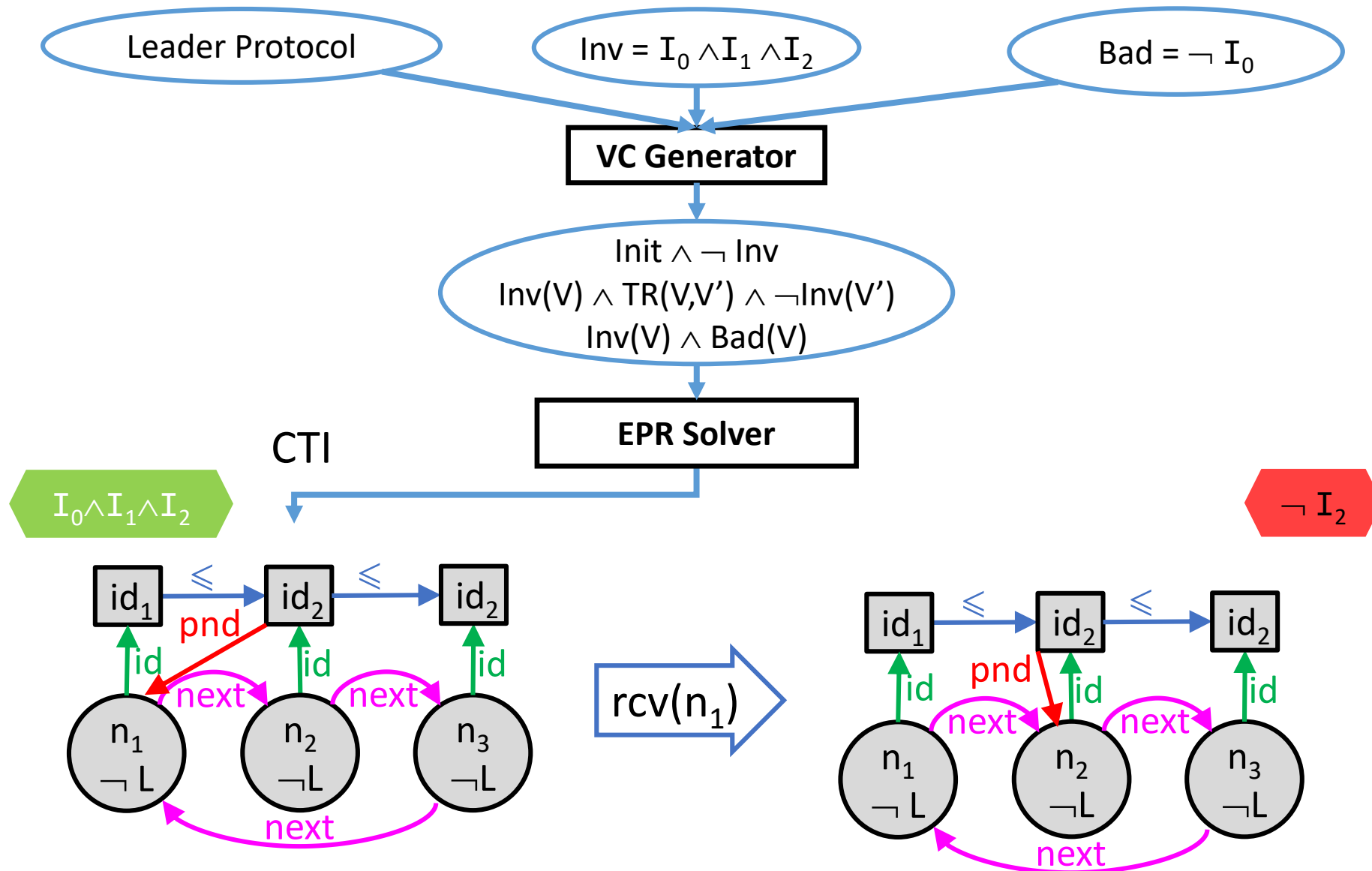
Ivy: Check Inductiveness (1)



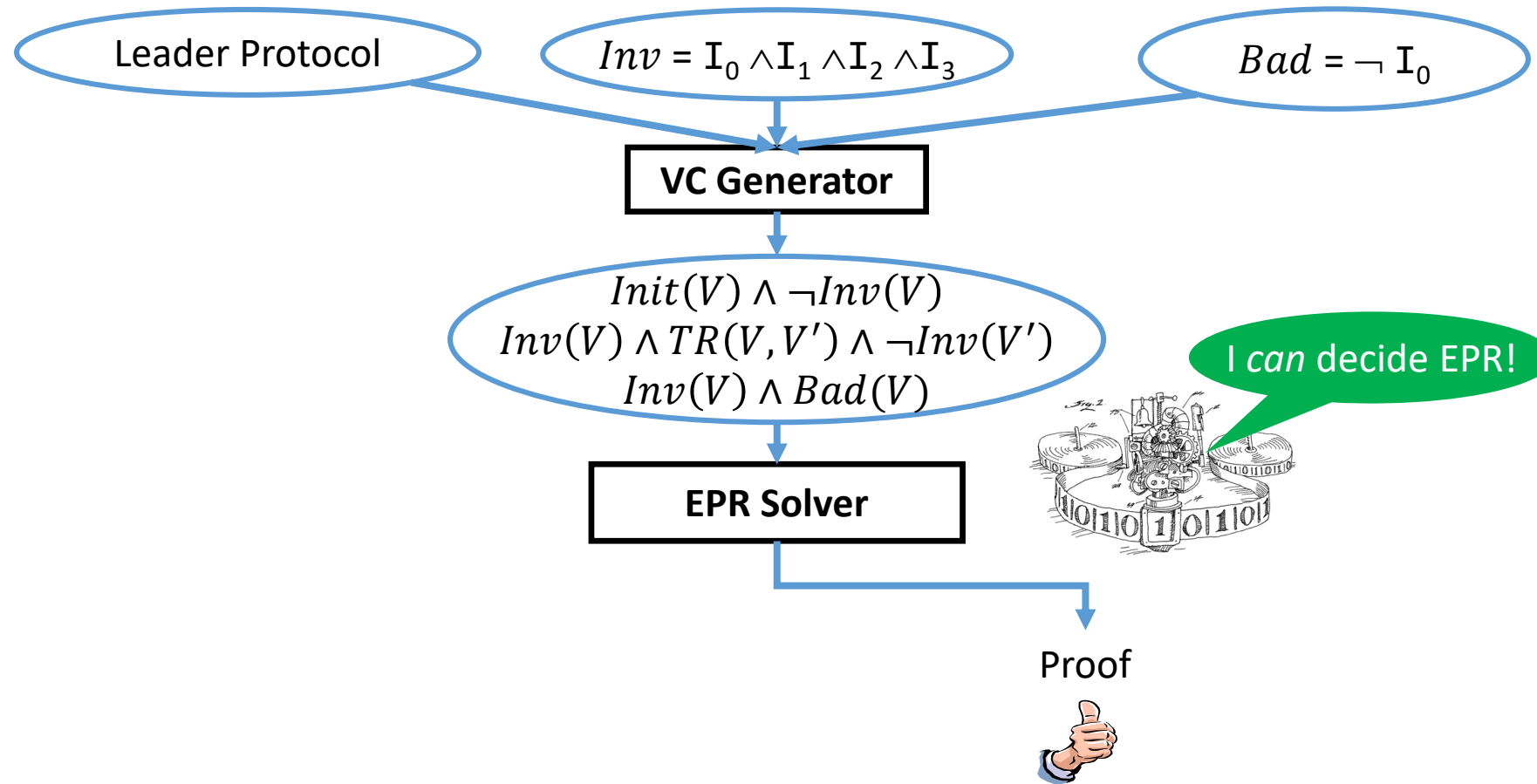
Ivy: Check Inductiveness (2)



Ivy: Check Inductiveness (3)

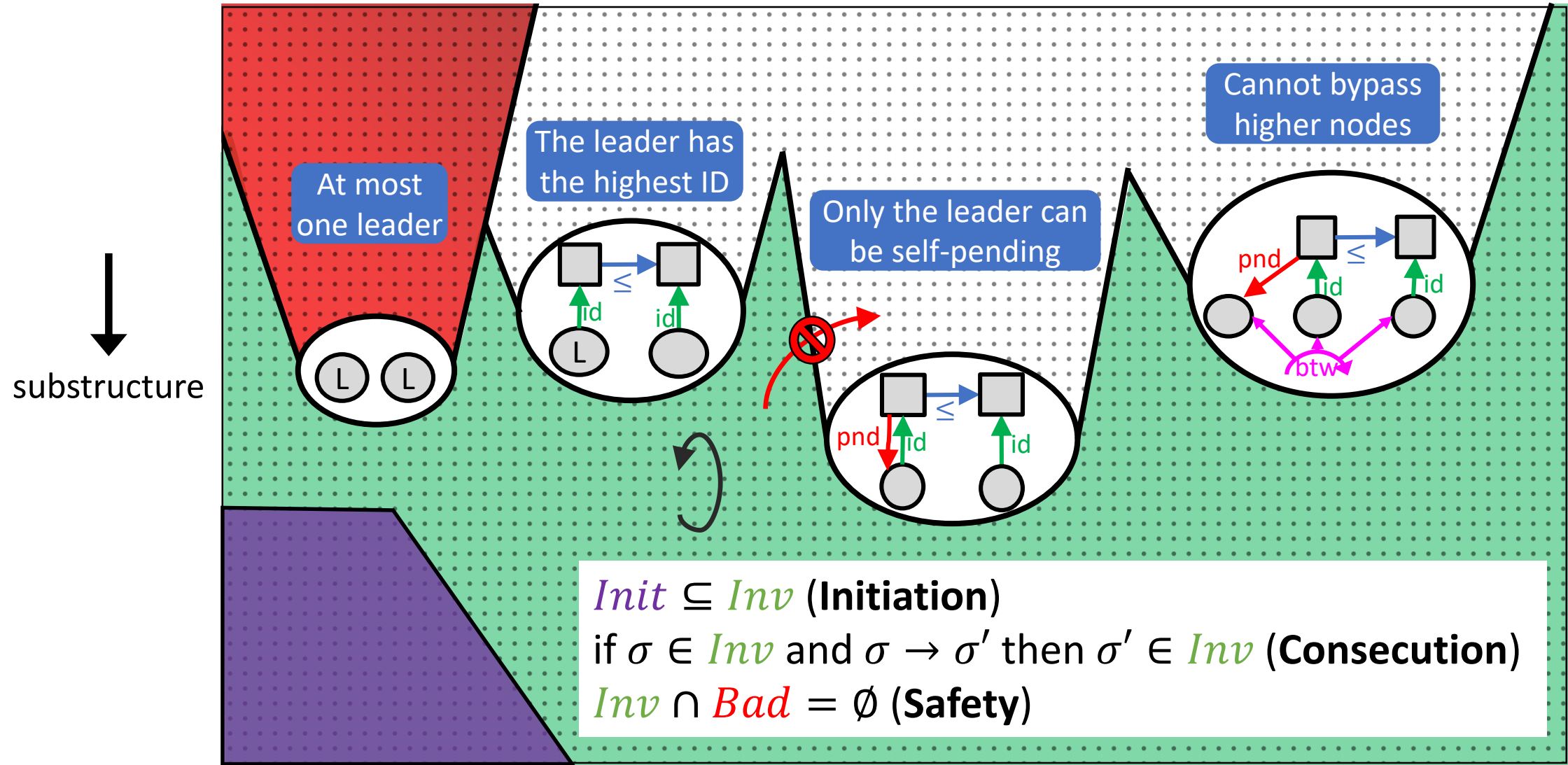


Ivy: check inductiveness



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

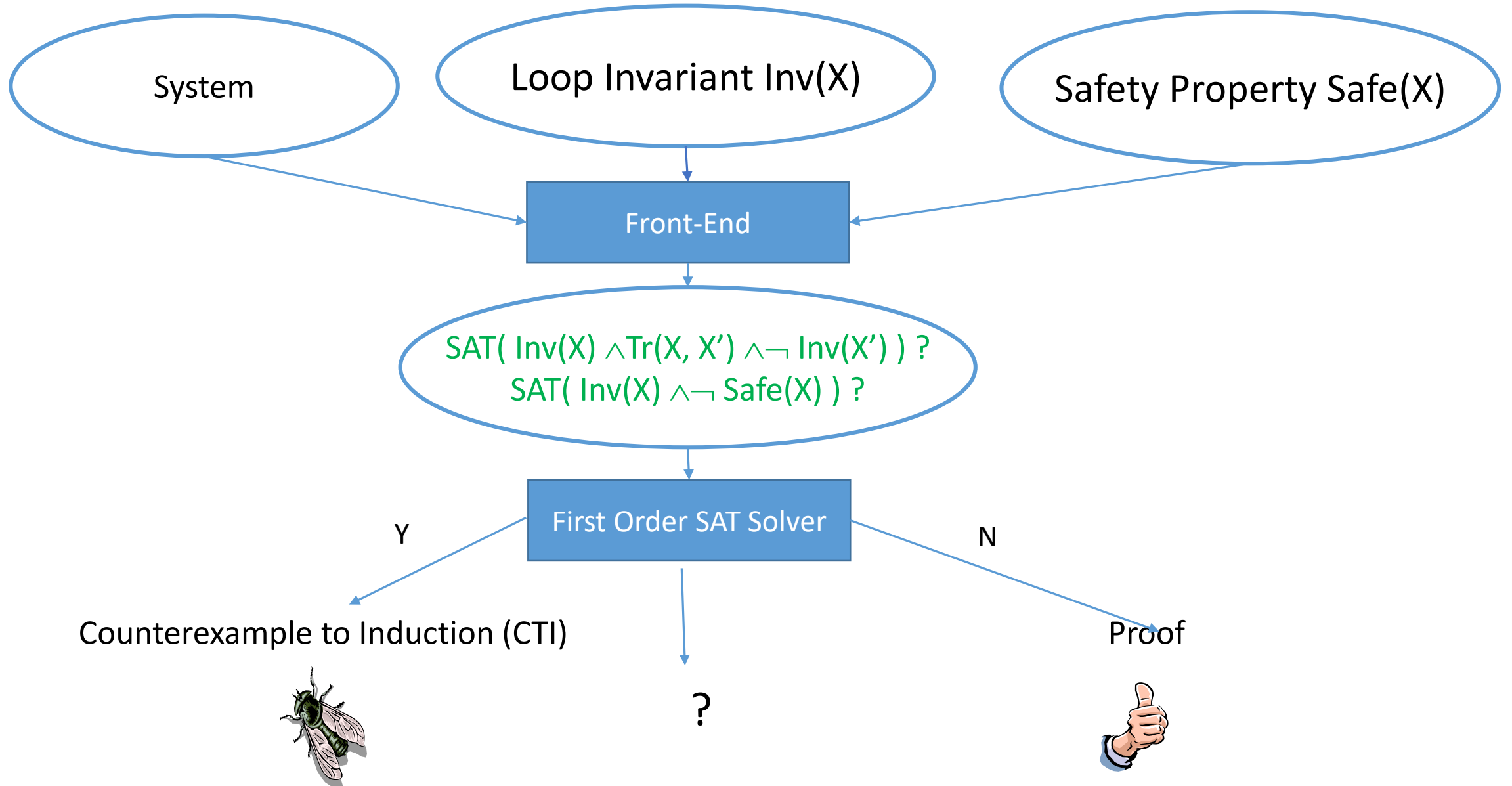
\forall^* invariant – excluded substructures



The First Order Satisfiability Problem

- Input
 - A vocabulary V
 - A formula φ over V
- Questions
 - Does there exist an interpretation \mathfrak{I} for V such that $\mathfrak{I} \models \varphi$? (SAT)
 - Does φ hold for every interpretation \mathfrak{I} for V ? (Validity)
 - $\neg \varphi$ is UNSAT
- Undecidable – validity is R.E.

Deductive verification by reductions to First Order Logic



The challenge of automated deduction

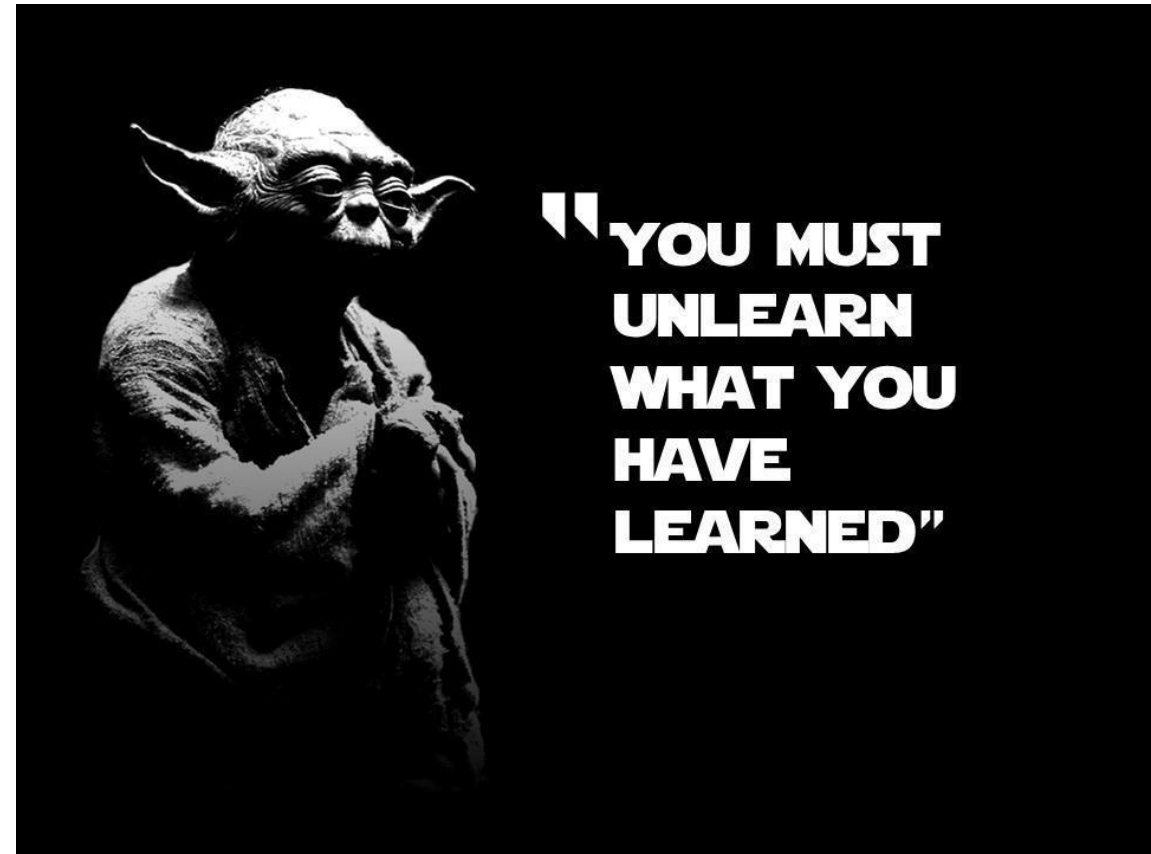
from: Ferraiulo, Baumann, Hawblitzel, Parno, “Komodo: Using Verification to Disentangle Secure-enclave Hardware from Software”, SOSPP ’17

“ The most frustrating recurring problem was proof instability [...] Timeouts are challenging to debug, because the solver generally fails to provide useful feedback [...] even once fixed, the proof may easily timeout again due to minor perturbations. Worse, minor changes can trigger timeouts in seemingly unrelated proofs ”

Gap: deductive power of automated provers is not translating into verification productivity

Rich logic, Poor logic

- SMT
 - Linear arithmetic, Bitvectors, Arrays, Strings, ...
- Great tools: Yices, Z3, CVC, Boolector, ...
- Essential in Dafny, Sage, Klee, F*,
- Hides complexity from the user
- But unpredictable, and not transparent



Effectively Propositional Logic – EPR a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic **without theories**
 - No function symbols
 - Restricted quantifier prefix: $\exists^* \forall^* \varphi_{QF}$
 - No $\forall \exists$



EPR Satisfiability



Skolem

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

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



Herbrand

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

$$=_{\text{SAT}} (p_{11} \leftrightarrow p_{12}) \wedge (p_{12} \leftrightarrow p_{22})$$

Effectively Propositional Logic – EPR a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic **without theories**
 - No function symbols
 - Restricted quantifier prefix: $\exists^* \forall^* \varphi_{QF}$
- Finite model property
 - A formula is satisfiable iff it has a model of size:
constant symbols + # existential variables
- Complexity:
 - NEXPTIME-complete
 - Σ_2^P if relation arities are bounded by a constant
 - NP if quantifier prefix is also bounded by a constant

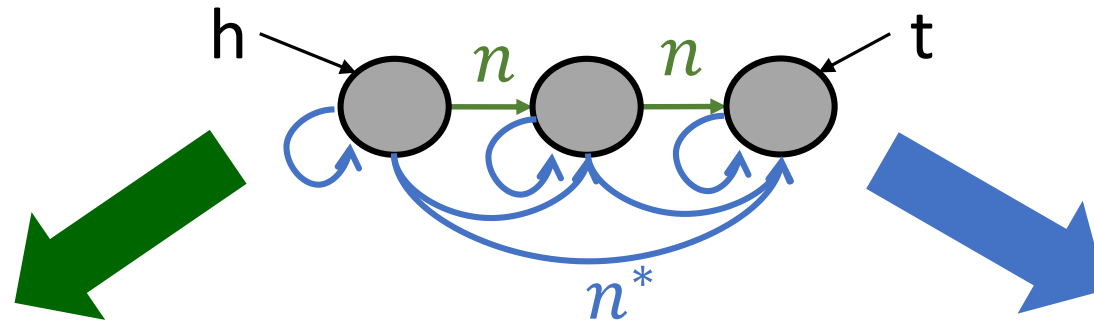


EPR++

- EPR++ allow **acyclic** function and quantifier alternations
 - E.g., $f: A \rightarrow B$ without $g: B \rightarrow A$
 - Maintains small model property of EPR
 - Finite complete instantiations
- **But what can you possibly express in such a restricted logic?**
 - **Transitive closure over deterministic paths**
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

Key idea: representing deterministic paths

[Shachar Itzhaky PhD, SIGPLAN Dissertation Award 2016]



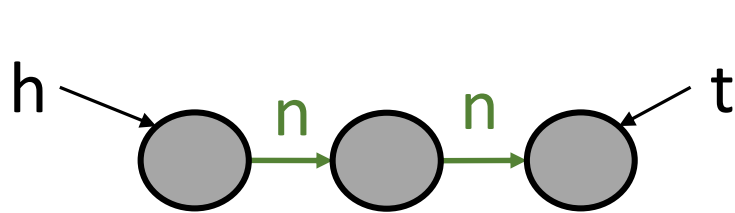
Alternative 1: maintain n

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

Alternative 2: maintain n^*

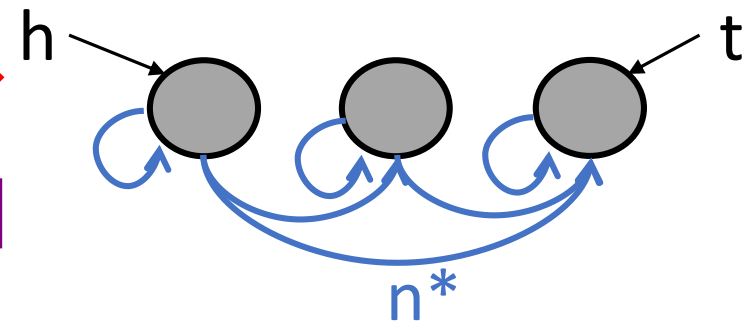
- n defined by transitive reduction of n^*
- Unique due to outdegree ≤ 1
- Definable in first order logic

$$n(x, y) \equiv n^*(x, y) \wedge x \neq y \wedge \forall z. n^*(x, z) \rightarrow n^*(y, z)$$

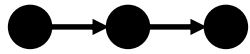


Not first order expressible

First order expressible

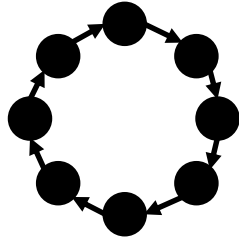


Sound and complete* axiomatization of deterministic paths



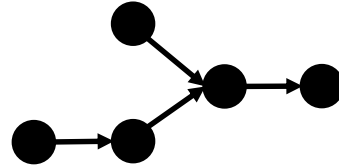
Line

$\leq (x, y)$



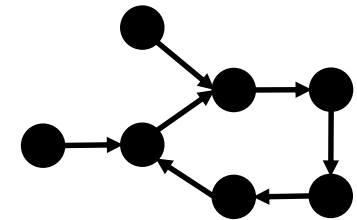
Ring

$btw(x, y, z)$



Forest, Tree,
Acyclic partial function

$\leq (x, y)$



Graph with out degree 1,
General partial function

$p(x, y, z)$

For every class C of finite graphs above:

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

Soundness Theorem

*Every graph of class C satisfies the axioms of C
Edges agree with successor formula*

Completeness Theorem

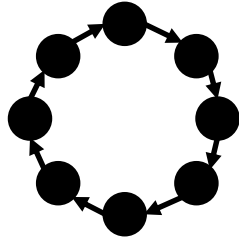
*Every finite structure satisfying the axioms of C is
isomorphic (paths and edges) to a graph of class C*

Sound and complete* axiomatization of deterministic paths



Line

$\leq (x, y)$



Ring

$btw(x, y, z)$

Ac

Universally quantified formulas
 \rightarrow finite model property
 + Completeness Thm. for finite structures

Sound and complete automatic
 deductive verification

For every class C of finite graphs above

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

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

• **Completeness Theorem** *Every finite structure satisfying the axioms of C is
 isomorphic (paths and edges) to a graph of class C*

EPR++

- But what can you possibly express in such a restricted logic?
 - Transtive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

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
- Active research and extensive industry use



Sets and cardinalities in EPR

- Consensus algorithms use set cardinalities
 - Wait for messages from **more than $N / 2$ nodes**
- Set Cardinalities + Arithmetic + Uninterpreted \gg EPR ?
- Insight: set cardinalities are used to get a simple effect
Can be modeled in first-order logic and EPR!
- 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) {  
  require ">N/2 join_msg's"  
  ...  
}
```



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

EPR++

- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

Quantifier alternation cycles

- Axiom

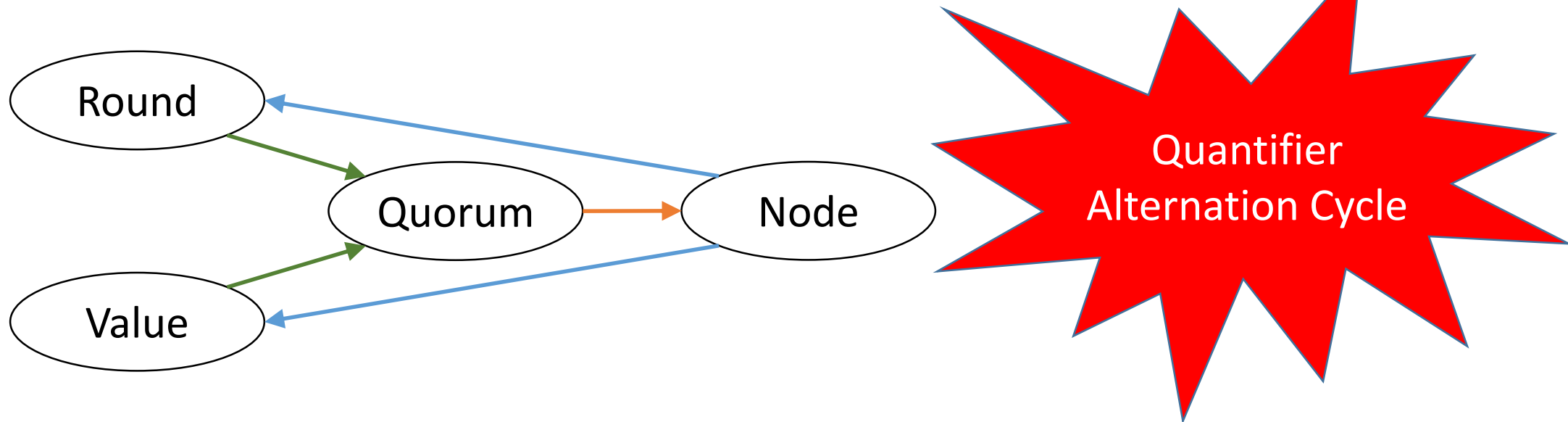
$\forall q_1, q_2: \text{Quorum}. \exists n: \text{Node}. \text{member}(n, q_1) \wedge \text{member}(n, q_2)$

- 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

$\forall r: \text{Round}, v: \text{Value}. \text{decision}(r, v) \rightarrow \exists q: \text{Quorum}. \forall n: \text{Node}. \text{member}(n, q) \rightarrow \text{vote_msg}(n, r, v)$

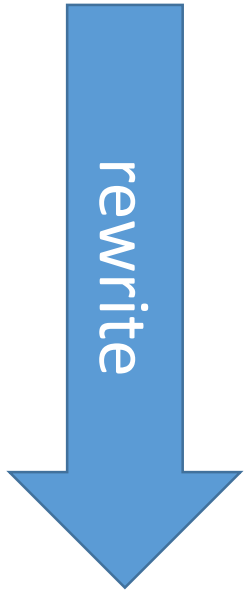


Derived relations

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

Derived relations

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



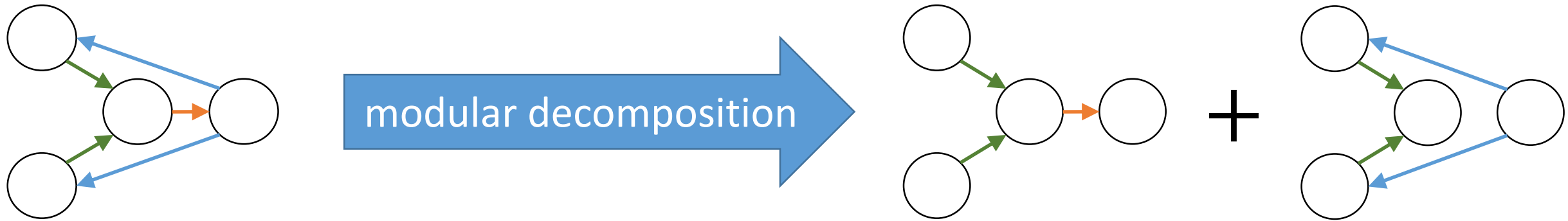
new relation: $\text{joined}(n:\text{node}, r:\text{round}) \equiv \exists r':\text{round}, v':\text{value}. \text{join_msg}(n,r,r',v')$

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  
}
```

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

Modularity



Proof Length

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

Verification Effort

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

EPR++

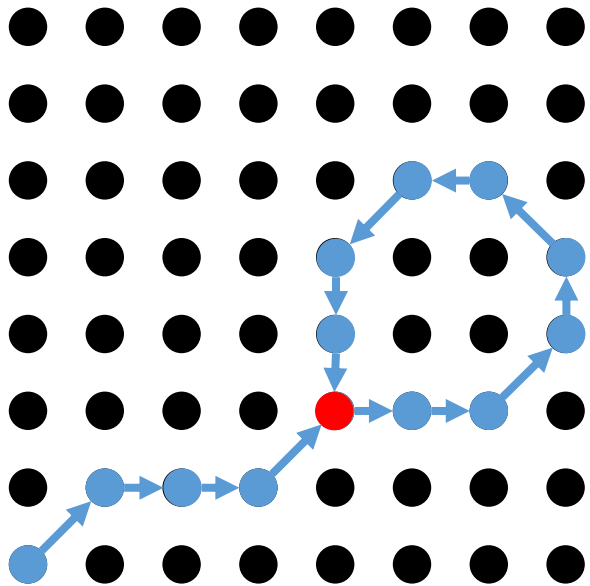
- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL

Liveness properties

- Liveness property: “something good eventually happens”
- Often depend on fairness assumptions
- Typically proven by ranking functions, well-founded relations
 - Well beyond EPR, or not?

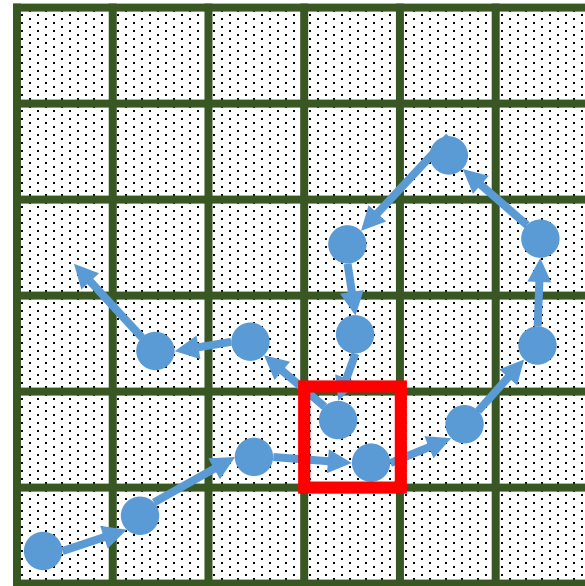
Lasso & Dynamic Abstraction

Finite State
Parameterized



Liveness \Leftrightarrow No Lasso

Infinite State
Finite Abstraction

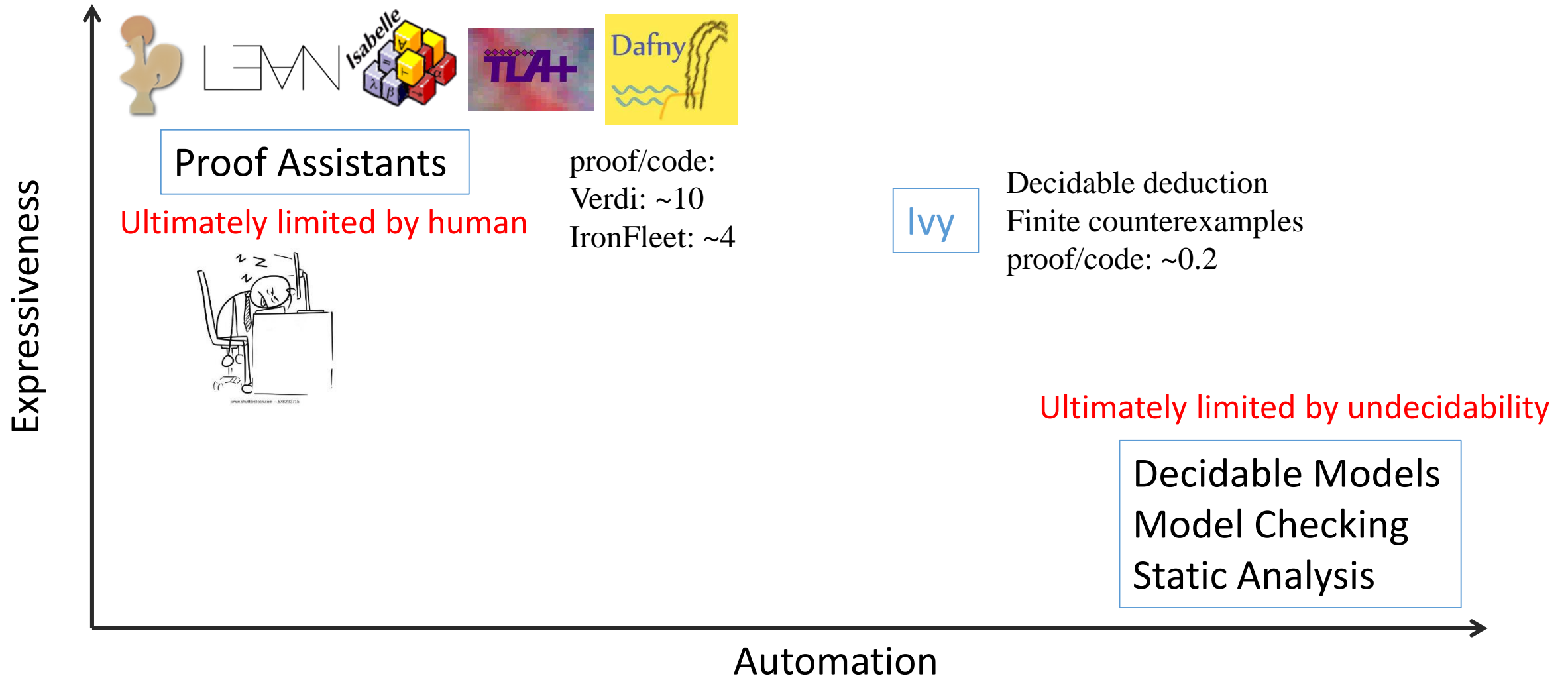


Liveness \Leftarrow No Lasso
Problem: Spurious Lasso

EPR++

- But what can you possibly express in such a restricted logic?
 - Transitive closure over deterministic paths
 - Set cardinalities
 - Avoiding quantifier alternations
 - Encoding liveness and LTL
- Implemented in Ivy

Ivy in the design space for formal verification



Decidable Fragments in Ivy

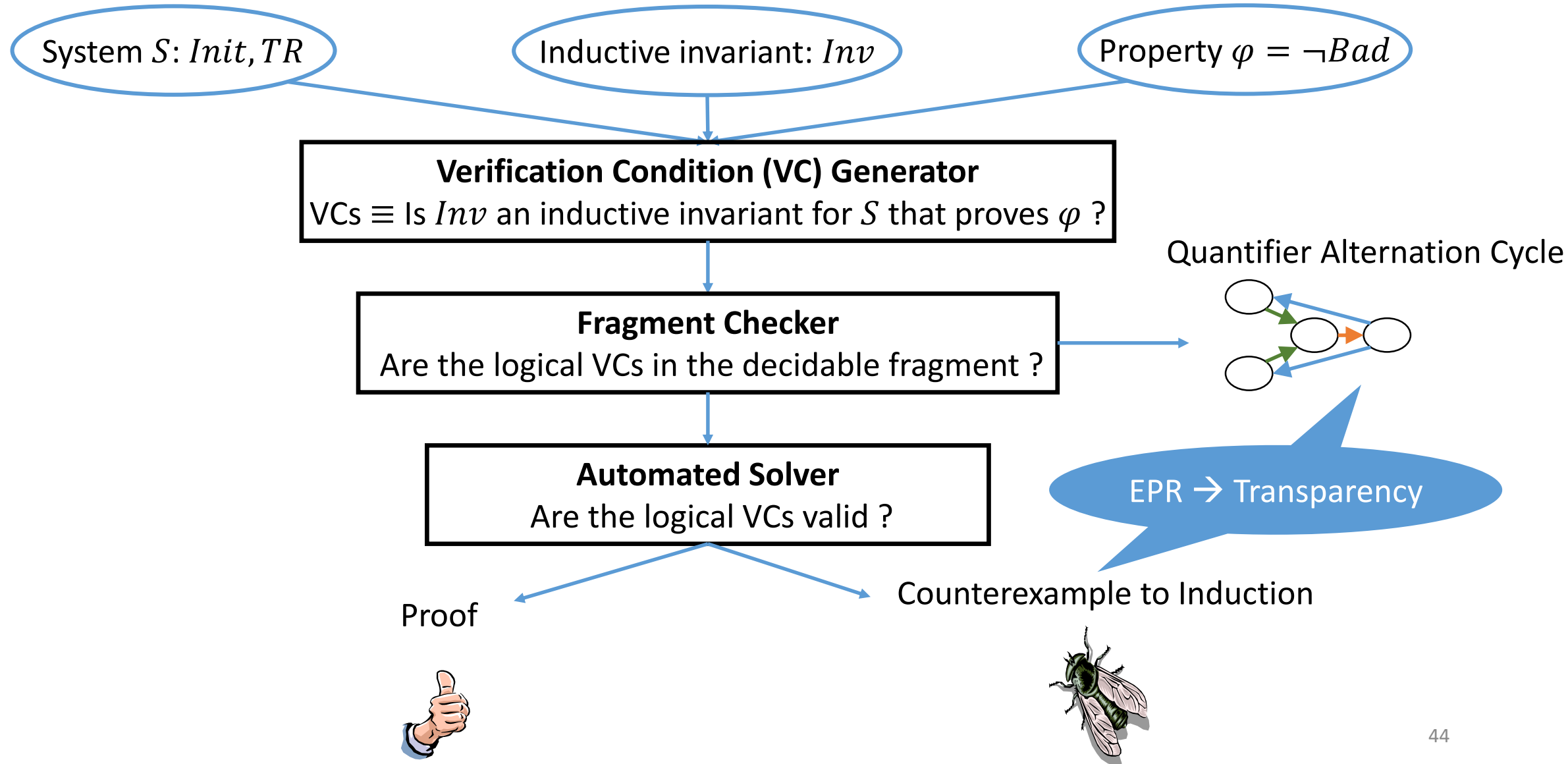
- EPR
- EPR++

Designs

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

Low Level
Implementations

Logic-based deductive verification in Ivy



Protocol	Model [LOC]	Invariants	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
Lock Server (500 Coq lines [Verdi])	122	9	2
Distributed Lock (1 week [IronFleet])	41	7	1.4
Single Decree Paxos (+liveness)	85	40	10.7
Multi-Paxos (+liveness)	98	55	14.6
Vertical Paxos*	123	18	2.2
Fast Paxos	117	17	6.2
Flexible Paxos	88	11	2.2
Stoppable Paxos (+liveness) *	132	60	18.4
Ticket Protocol (+liveness)	86	45	6
Ticket w/ Task Queues (+liveness)	99	60	9.4
Alternating Bit Protocol (+liveness)	161	70	32
TLB Shutdown (+liveness) *	385	102	283

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

* first mechanized verification

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 easy.

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 \bigcirc meaning *eventually*. [13]. We use a linear-time logic, so \bigcirc can be defined by $\bigcirc F \triangleq \neg \square \neg F$, for any formula F . For a state predicate P , the formula $\square P$ asserts that P is an invariant, meaning that it is true for every reachable state. The temporal formula $\bigcirc \square P$ asserts that at some point in the execution, P holds 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 reachable from s by any action of the algorithm. We let $\text{stable } P$ be the assertion that state predicate P is stable. It is clear that a stable predicate is invariant if it is true in the initial state. Because stability is an assertion only about reachable states s , 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-page proofs are written with a hierarchical numbering scheme in which τ_{ij} is the number of the j^{th} step of the current level- i proof [9]. Although it may appear intimidating, this kind of proof is easy to check and helps to avoid errors.

A.1 The Proof of Safety

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

$\text{NotChosen}(i, k, v) \triangleq$
 $(\exists Q : \forall a \in Q : (\text{ball}[a] > k) \wedge (\text{vote}[a][k] \neq v))$
 $\vee (\exists j < i : w \in \text{StopCmd} : \text{Done2}(j, k, w))$
 $\vee (\exists (v \in \text{StopCmd}) : \exists j > i, w : \text{Done2}(j, k, w))$

We next prove a number of simple invariance and stability properties of the algorithm.

Lemma 1

- $\forall i, k, v : \square (\text{Chosen}(i, k, v) \rightarrow \text{Done2}(i, k, v))$.
- $\forall i, k, v, w : \square (\text{Done2}(i, k, v) \wedge \text{Done2}(i, k, w) \rightarrow (v = w))$
- $\forall i, k, v : \square (\text{vote}[i][k] = v \rightarrow \text{Done2}(i, k, v))$

14

We now prove some less obvious invariants.

Lemma 2 $\forall i, k, v : \square (\text{NotChosen}(i, k, v) \rightarrow \neg \text{Chosen}(i, k, v))$

15

Lemma 3 $\forall i, k, v : \square (\text{NotChosen}(i, k, v) \rightarrow \neg \text{Chosen}(i, k, v))$

16

Lemma 4 $\forall i, k, v : \square (\text{NotChosen}(i, k, v) \rightarrow \neg \text{Chosen}(i, k, v))$

17

Proof: Assume $\text{vote}[i][k, Q] \neq v$. By definition of $\text{vote}[i]$, this implies $\text{vote}[i][k, Q]$ is a command (and not \perp). Since $\text{E}1(i, k)$ holds by assumption (11), the definition of $\text{vote}[i]$ and $\text{vote}[i]$ imply that some acceptor a in Q has sent a $(\text{ID}, a, k, \text{vote}[a][k, Q], \text{vote}[a][k, Q])$ message, which implies $\text{vote}[a][\text{ball}[\text{vote}[a][k, Q]] = \text{vote}[a][k, Q]$ when the message was sent. Lemma 1.7 then implies $\text{Done2}(i, \text{ball}[\text{vote}[a][k, Q]], \text{vote}[a][k, Q])$ was true when the message was sent, and is still true because $\text{Done2}(\dots)$ is stable.

(12) $\forall i, c, v, w : (\text{vote}[i][c, v] = v \wedge \text{vote}[i][c, w] = w) \rightarrow v = w$
Proof: We assume $c < k$ and $\text{vote}[i][c, v] = v$ and $\text{vote}[i][c, w] = w$ and prove $v = w$. We split the proof into two cases.
(21) Case: $\text{vote}[i][c, v] = \perp$
Proof: The case assumption implies $\text{vote}[i][c, v] < c$, so assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, v)$.

(22) Case: $\text{vote}[i][c, v] \neq \perp$
Proof: Since $c < k$, we can split the proof into the following three cases.
(21) Case: $\text{vote}[i][c, v] < c < k$
Proof: By assumption (11.4), the case assumption and Lemma 3 imply $\text{NotChosen}(i, c, v)$.

(22) Case: $\text{vote}[i][c, v] < c < k$
Proof: By (11.3).
(23) Case: $c < \text{ball}[\text{vote}[i][c, v]]$ and $w = \text{vote}[i][c, v]$
(i) $\text{vote}[i][c, v] \in \text{StopCmd}$ and $w = \text{vote}[i][c, v]$ and we can choose $j > j$ such that $\text{ball}[\text{vote}[i][c, v]] < \text{ball}[\text{vote}[i][c, v]]$.
Proof: We deduce that $\text{vote}[i][c, v] \in \text{StopCmd}$ and that $c < k$ by the (22) case assumption, the assumption $\text{vote}[i][c, v] < c < k$, and the definition of $\text{vote}[i]$.

(ii) $\text{vote}[i][c, v] \in \text{StopCmd}$ and $\text{ball}[\text{vote}[i][c, v]] < \text{ball}[\text{vote}[i][c, v]]$
Proof: The (22) case assumption and (i) imply $\text{ball}[\text{vote}[i][c, v]] < \text{ball}[\text{vote}[i][c, v]]$.
Step (12) then proves (22).

(23) $\text{NotChosen}(i, c, w)$
Proof: Assumption (11.4) with $v = k$, $c = \text{ball}[\text{vote}[i][c, v]]$, and $w = \text{vote}[i][c, v]$ and (i2) imply $\text{NotChosen}(i, c, w)$.

Step (12) asserts $j < k$; case assumption (23) and (i1) imply

18

$c < \text{ball}[\text{vote}[i][c, v]]$ and (i1) and case assumption (23) imply $\text{vote}[i][c, v] < c < k$ and (11.4) and case assumption (23) imply $\text{NotChosen}(i, c, w)$. We split the proof into two cases.
(21) Case: $\text{vote}[i][c, v] = \perp$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(22) Case: $\text{vote}[i][c, v] \neq \perp$
Proof: Since $c < k$, we can break the proof into two sub-cases.
(21) Case: $\text{vote}[i][c, v] < c < k$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(22) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.3) implies $\text{E}1(i, k, Q, v)$. Case assumption (22) and $\text{E}1(i, k, Q, v)$ imply $v = \text{vote}[i][k, Q]$. Case assumption (22) and the assumption $v \neq w$ and step (13) substituting $v \rightarrow w$ then imply $\text{NotChosen}(i, c, w)$.

(23) $\text{NotChosen}(i, c, w)$
Proof: We assume $v = c$, $w \in \text{StopCmd}$, and $c < k$ and we prove $\text{NotChosen}(i, c, w)$. By Lemma 1.7, it suffices to prove $\text{NotChosen}(i, c, w)$. Since $c < k$, we need consider only the following two cases.
(21) Case: $c < k$
Proof: Step (3/2) and assumption (11.1) imply $\text{NotChosen}(i, c, w)$.
(22) Case: $c < k$
Proof: Assumption (11.4) implies $\text{Done2}(i, k, v)$. Since $v > j$ and $w \in \text{StopCmd}$, this implies the third disjunct of $\text{NotChosen}(i, k, w)$ (substituting v and w for the existentially quantified variables), which by the case assumption proves $\text{NotChosen}(i, c, w)$.

(23) Case: $c < k$
Proof: We consider two sub-cases.
(21) Case: $\text{vote}[i][c, v] < c < k$
Proof: (11) and case assumption (22) imply $\text{NotChosen}(i, c, w)$.

(22) Case: $\text{vote}[i][c, v] < c < k$
Proof: (11) and case assumption (22) imply $\text{NotChosen}(i, c, w)$.

(23) $\text{NotChosen}(i, c, w)$
Proof: (11) and case assumption (22) imply $\text{NotChosen}(i, c, w)$.

(24) Case: $\text{vote}[i][c, v] < c < k$
Proof: Assumption (11.4), the case assumption, and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(25) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(26) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(27) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(28) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(29) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

(30) Case: $c < \text{ball}[\text{vote}[i][c, v]]$
Proof: Assumption (11.4) and Lemma 3 imply $\text{NotChosen}(i, c, w)$.

19

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

20

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and Chosen(i, k, v) imply $\text{Done2}(i, k, v)$. By Lemma 1, this implies

21

Since $\neg \text{Stop}$ is equivalent to D-Stop , for any formula F , we can split the proof into the following two cases.

(13) Case: $\exists k > i, w : \square \text{Done2}(k, k, w)$
(21) $\square \text{E}1(i, k, v)$
Proof: By definition of $\text{E}1(i, k, v)$, it suffices to assume $j < i, v \in \text{StopCmd}$, and $\text{Done2}(i, k, v)$ and obtain a contradiction. By the (13) case assumption, we have $\text{Done2}(k, k, w)$ for $k > i > j > j$. Since $k \neq j$, the following two cases are exhaustive.

(31) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(j, k, v)$
Proof: This is impossible because the enabling condition $\text{E}1(i, k, v)$ of $\text{Phase2}(i, k, v)$ implies $\neg \text{Done2}(j, k, v)$.

(32) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(33) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(34) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(35) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(36) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(37) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(38) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(39) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(40) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(41) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(42) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(43) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(44) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(45) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(46) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(47) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(48) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(49) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(50) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(51) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(52) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(53) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(54) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(55) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(56) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(57) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(58) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(59) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(60) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(61) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(62) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(63) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(64) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(65) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$
Proof: This is impossible because $\text{E}1(i, k, v)$ implies $\neg \text{Done2}(k, k, w)$.

(66) Case: $\text{Phase2}(i, k, v)$ is executed after $\text{Phase2}(k, k, w)$

Stoppable Paxos

Appendix: The Proof of Correctness

We now prove that Stoppable Paxos satisfies its safety and liveness properties. For clarity and concision, we write simple temporal logic with two temporal operators: \square meaning always, and \diamond meaning eventually [15]. We use a linear-time logic, so \square can be defined for any formula F . For a state predicate P , $\square P$ means that P is an invariant, meaning that it is true in all reachable states from that point onward.

We define a predicate P to be true in any reachable state s by any action of the state predicate P is true in state s if it is true in the last reachable state s' of s .

Our proof proceeds by showing that the assumptions imply the properties.

\dots imply $val2a(j, b, v)$.
then imply $NotChoosable(j, c, w)$.

$val2a(i, b, Q) \neq \top$
 $val2a(i, b, Q) = val2a(i, b, Q) = v$

PROOF: Assumption (1)1.3 implies $E3(i, b, v)$.
 $val2a(i, b, Q) = v$. The case assumption and the
implies $val2a(i, b, Q) = v$.

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

PROOF: (3)1, assumption (1)1.4, and the
 $vote_i[a][mbal2a(i, b, Q)] = v$ for some acceptor
implies $Done2a(i, mbal2a(i, b, Q), v)$.

By the assumption $c < b$, it suffices to consider

(3)3. CASE: $c < mbal2a(i, b, Q)$

PROOF: Step (3)2 and assumption
 $NoneChoosableAfter(i, mbal2a(i, b, Q), v)$. By the
assumptions $v \in StopCmd$ and $j > i$, this implies

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

(1)1. $mbal2a(j, b, Q) < mbal2a(i, b, Q)$

PROOF: The assumption $v \in StopCmd$ and
 $(i, b, Q) \in StopCmd$. Case assumption
then imply $mbal2a(k, b, Q)$

an instance number, v_h a command in

$\diamond Done2a(h, b, v_h)$

2. $\forall j > h, v : \square \neg Done2a(j, b, v)$

PROVE: $\exists v : \diamond Done2a(h + 1, b, v)$

PROOF: (2)1 and case assumption (1)4 implies that there is a largest instance number h and a command v_h such that $\diamond Done2a(h, b, v_h)$, and that $h \leq i$. If $v_h \in StopCmd$, then (1)2 implies $\diamond Chosen(h, b, v_h)$, and $h \leq i$ then implies we are done. Therefore, it suffices to assume $v_h \notin StopCmd$ and obtain a contradiction, which we do by proving that the assumptions imply the PROVE clause.

(2)3. $\diamond \square E5(h + 1, b)$

PROOF: Assumption (2)2.1 asserts $\diamond Done2a(h, b, v_h)$, which implies $\diamond E5(h, b)$. Since $Done2a(h, b, v_h)$ implies $\forall j < h, w \in StopCmd \neg E4a(j, b, w)$, it implies that $Phase2a(j, b, w, U)$ is not enabled for any $j < h, w \in StopCmd$, and quorum U , which implies that $E5(h, b)$ is stable, proving $\diamond \square E5(h, b)$. Assumption (2)2.1 and Lemma 1.2 imply $\forall v \in StopCmd \square \neg Done2a(h, b, v)$, which together with $\diamond \square E5(h, b)$ implies $\diamond \square E5(h + 1, b)$.

Choose a quorum U such that $Phase2a(h, b, v_h, U)$ is eventually created.

PROOF: U exists by assumption (2)2.1.

and U eventually created.

and the stability of $E1(b, U)$.

Summary

- Verification in EPR
 - Deduction is decidable
 - Finite counterexamples
 - Transparent failures
- Powerful encodings
 - Transitive closure for deterministic paths
 - Set cardinalities
 - Derived relations and modularity to battle $\forall\exists$ cycles
 - Liveness and temporal properties

Open Questions

- What cannot be proven with EPR?
 - Proof theory
- Why are solvers stable on EPR?
 - Can it be generalized?
 - Part of why CDCL and works
- Can it be used beyond a small group of fans?
 - Transitive closure 😊, LTL 😊, Cardinality 😊, Breaking $\forall\exists$ cycles 😞
- Can you get the benefit of EPR without the costs?
 - Bounded quantifier instantiations
 - Special case of “simple proofs”?

**YOU MUST
UNLEARN
WHAT YOU
HAVE
LEARNED**



Summary

- Distributed protocols are interesting for verification
 - But real distributed systems are more complex
- Can be naturally modeled in pure first order logic
- Decidable logics can be used to reason about interesting systems
 - **No more butterfly effects**
 - But some jagged corners
 - Details on Wednesday