Verification of Distributed Protocols: Decidable Modeling and Invariant Inference

Oded Padon

VMware Research

Tutorial at FMCAD 2022

18 October 2022

Verification of Distributed Protocols: Decidable Modeling and Invariant Inference

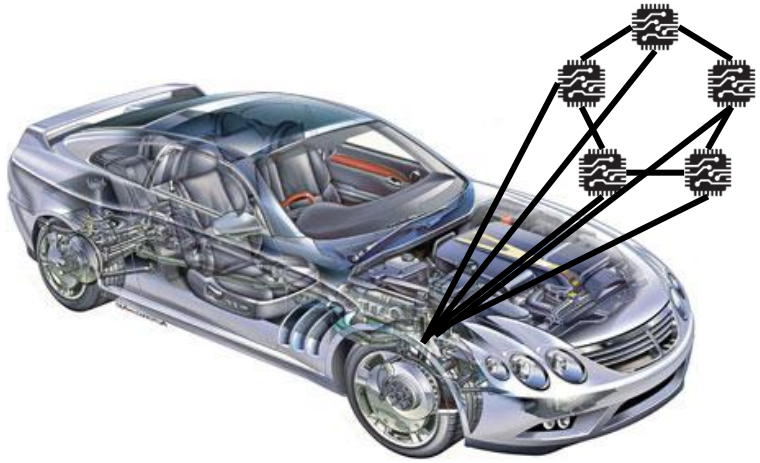
Part 1: Decidable Modeling

- The Ivy deductive verification system
- The many-sorted EPR fragment
- Main challenge: expressing interesting systems and properties in EPR
  - Expressing transitive closure
  - Expressing sets and cardinalities
  - Liveness and temporal verification

Part 2: Invariant Inference

- Problem setting

Distributed protocols – excellent opportunity for verification



- Distributed systems are everywhere
  - Safety-critical systems
  - Internet scale services
  - Cloud infrastructure

- Distributed systems are notoriously hard to get right
  - Even small protocols can be tricky
  - Bugs occur in rare scenarios
  - Testing is costly and not sufficient

SOSP'07

Best Paper Award

Zyzzyva: Speculative Byzantine Fault Tolerance

Zyzzyva is a state machine replication protocol based on three sub-protocols: (1) agreement, (2) view change, and (3) checkpoint. The *agreement* protocol orders requests for execution by the replicas. The *view change* protocol coordinates ... of a new primary when the current primary is ... we describe the basic proto- ...rectness [17]. In §4 we ...mented in our ...blic

arXiv:1712.01367 [cs.DC] 2017

Revisiting Fast Practical Byzantine Fault Tolerance

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

CACM'08

Zyzzyva: Speculative Byzanti... Fault Tolerance

DOI:10.1...

ACM Transactions on Computer Systems '0...

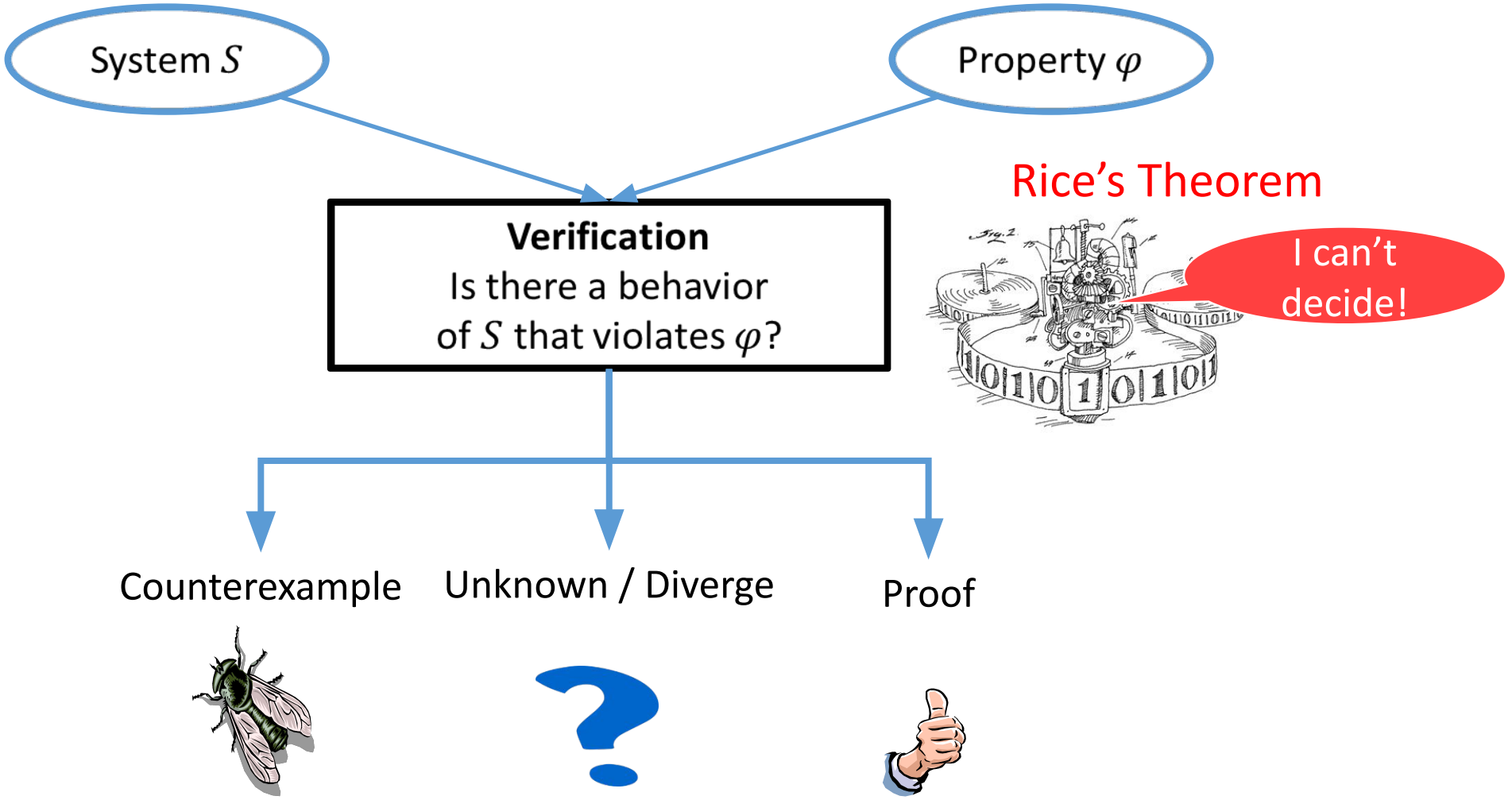
Zyzzyva: Speculative Byzantine Fault Tolerance

ASPLOS'16

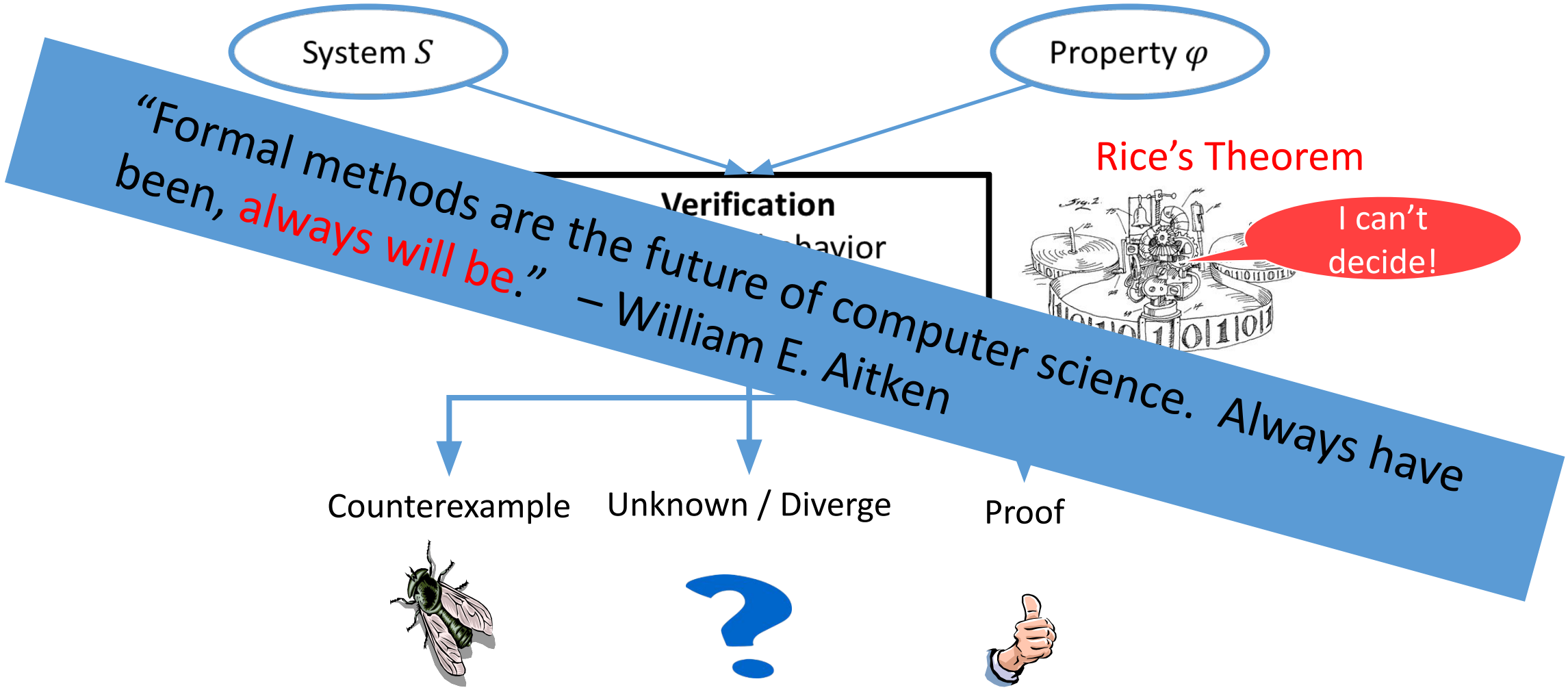
TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems

We present TaxDC, the largest and most comprehensive taxonomy of non-deterministic concurrency bugs in distributed systems. We study 104 distributed concurrency (DC) bugs from four widely-deployed cloud-scale datacenter distributed systems, Cassandra, Hadoop MapReduce, HBase and ZooKeeper. We study DC-bug characteristics along sev-

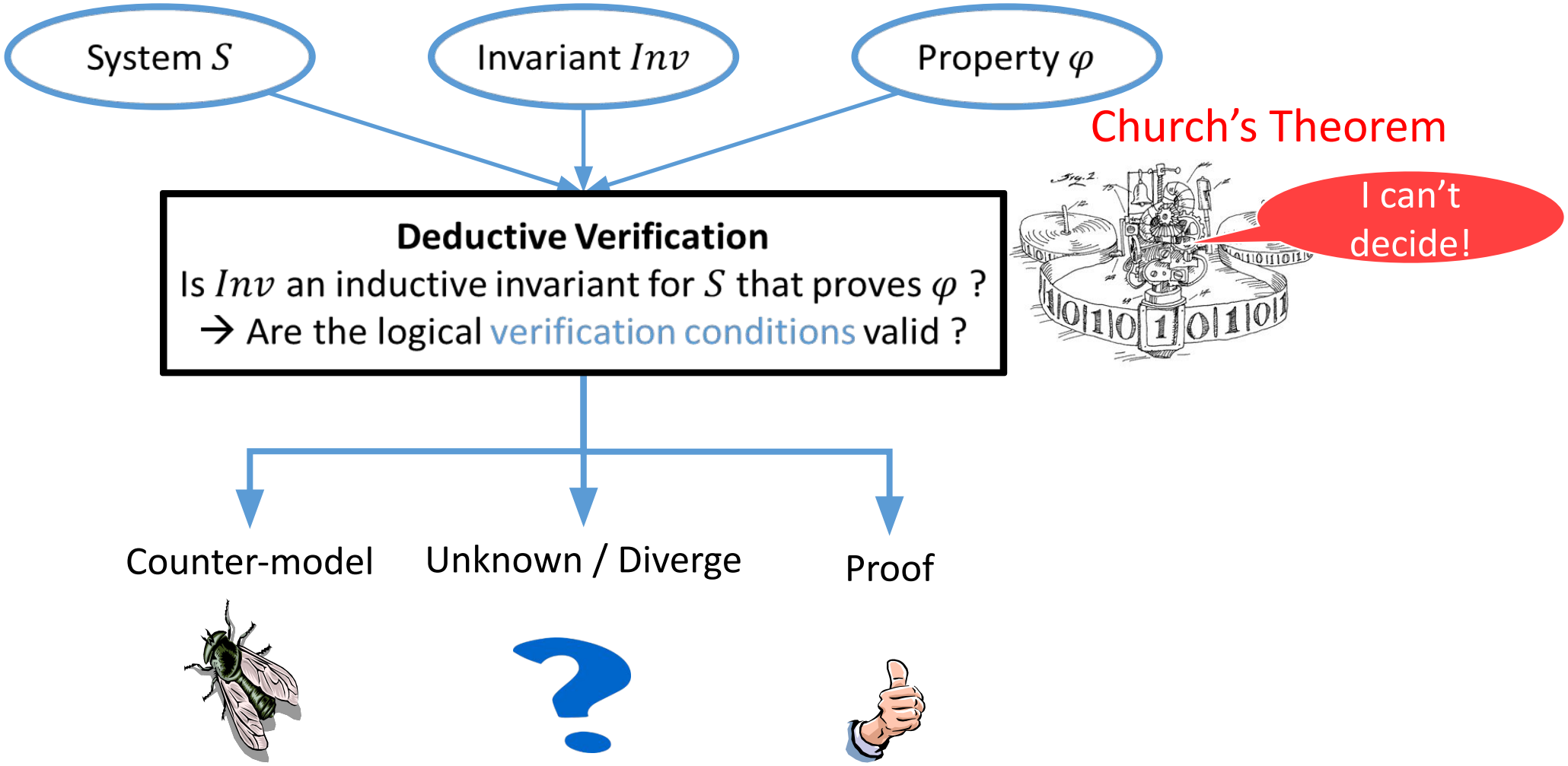
Automatic verification of infinite-state systems



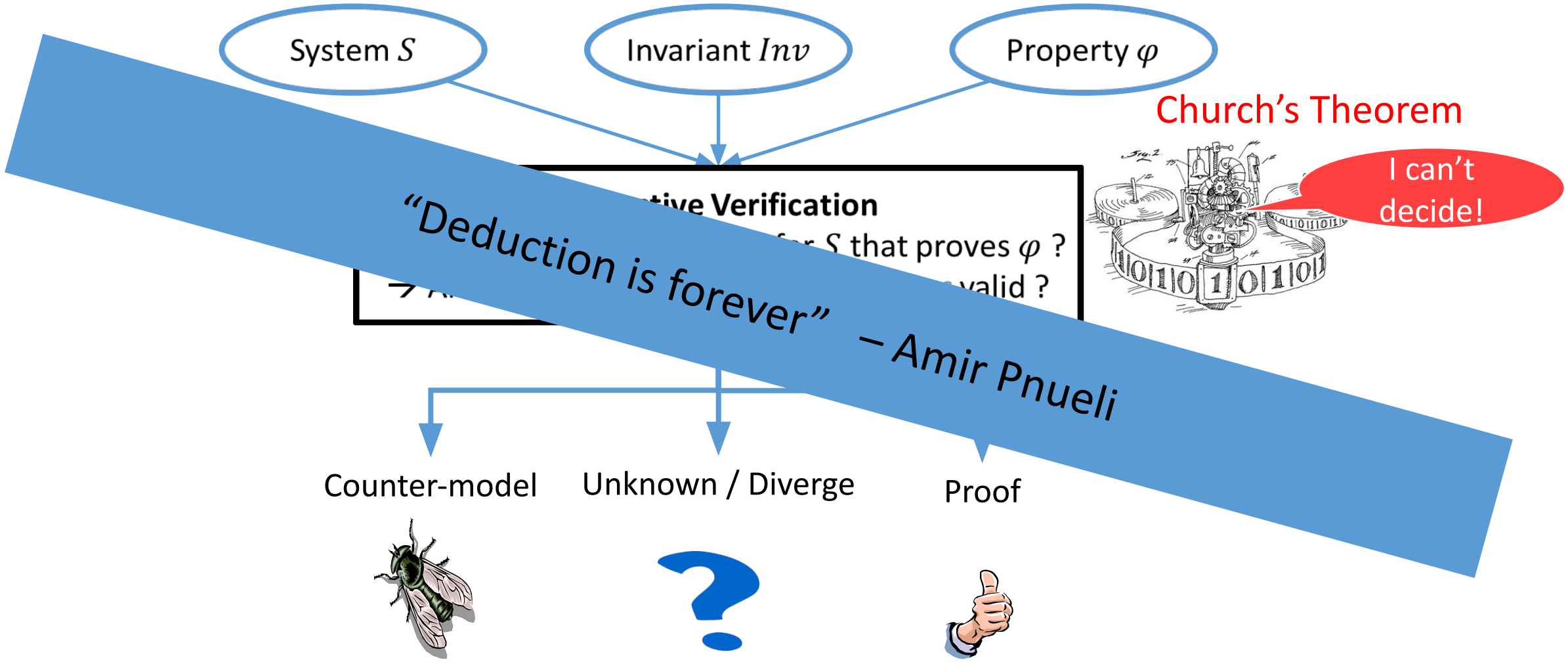
Automatic verification of infinite-state systems



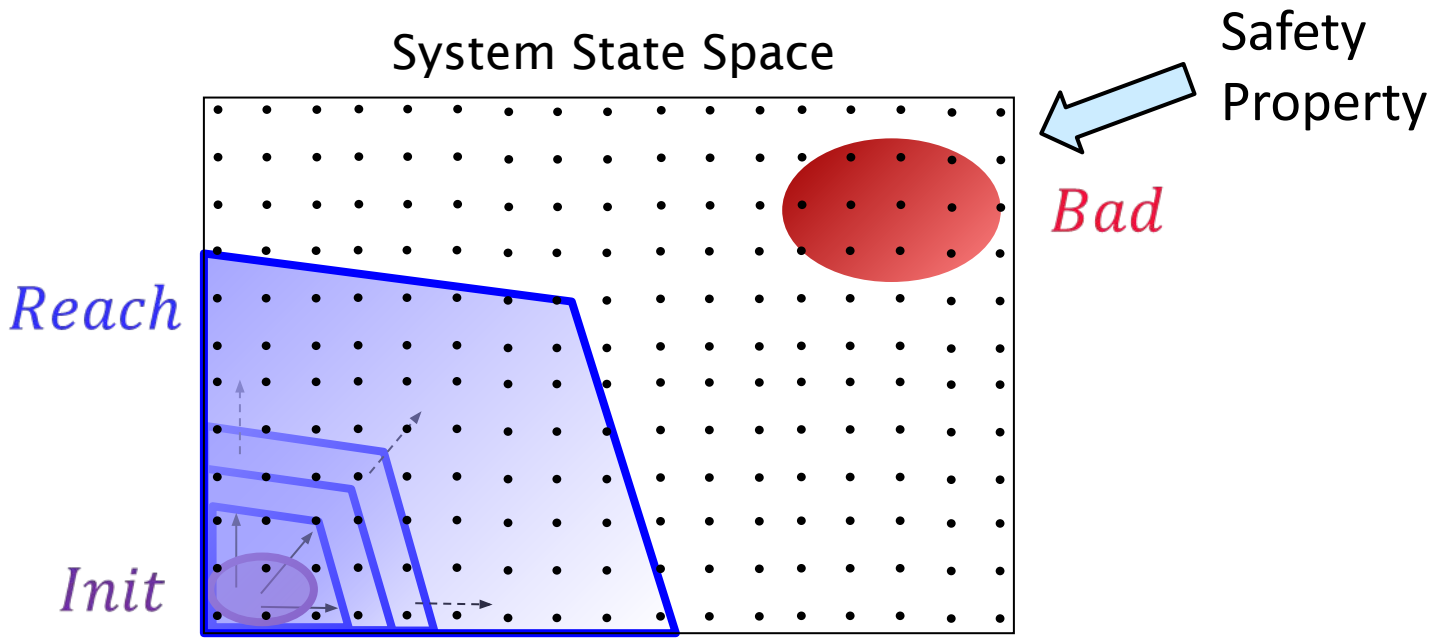
Deductive verification



Deductive verification

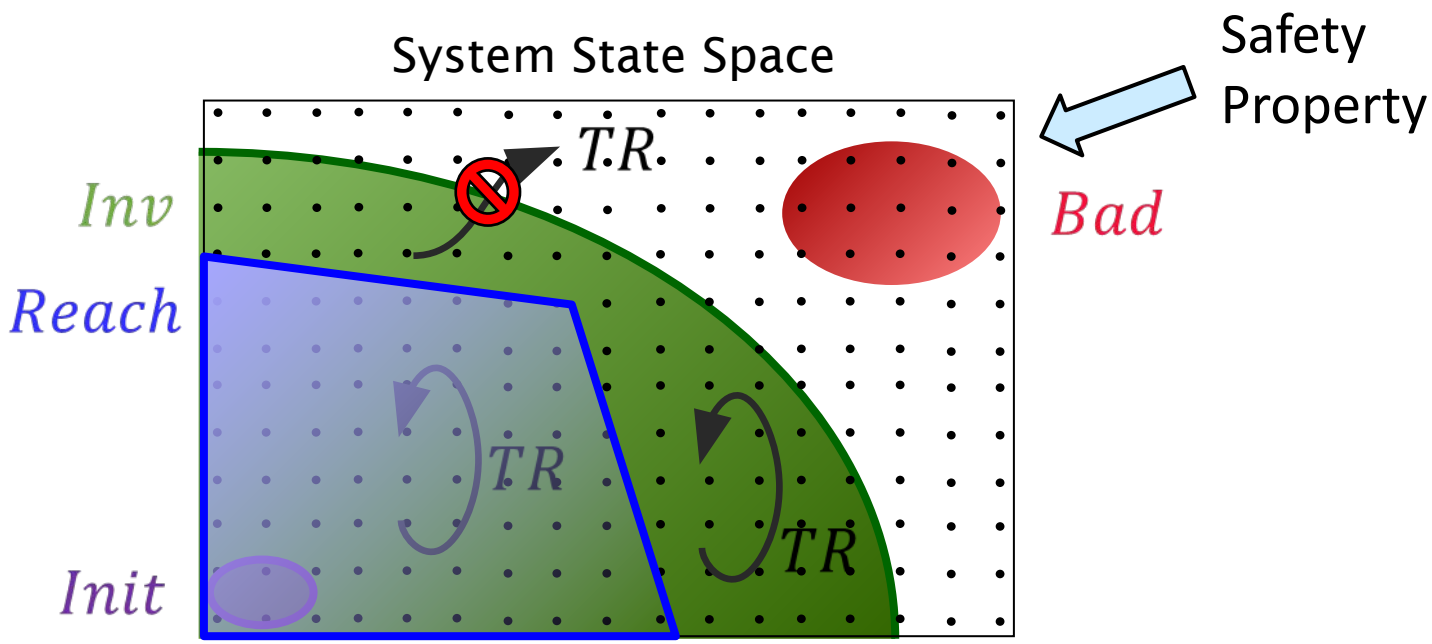


Inductive invariants



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

Inductive invariants



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

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

$Init \subseteq Inv$ (**Initiation**)

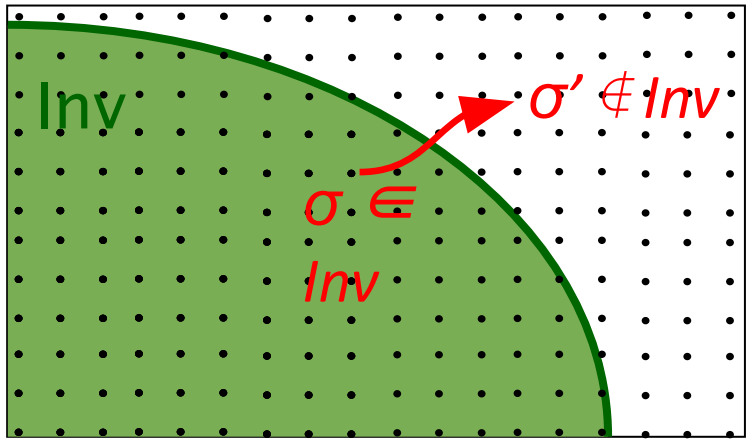
if $\sigma \in Inv$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in Inv$ (**Consecution**)

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

} translated to VCs

Counterexample To Induction (CTI)

- States σ,σ’ are a CTI of Inv if:
- σ ∈ Inv
- σ’ ∉ Inv
- σ □ σ’

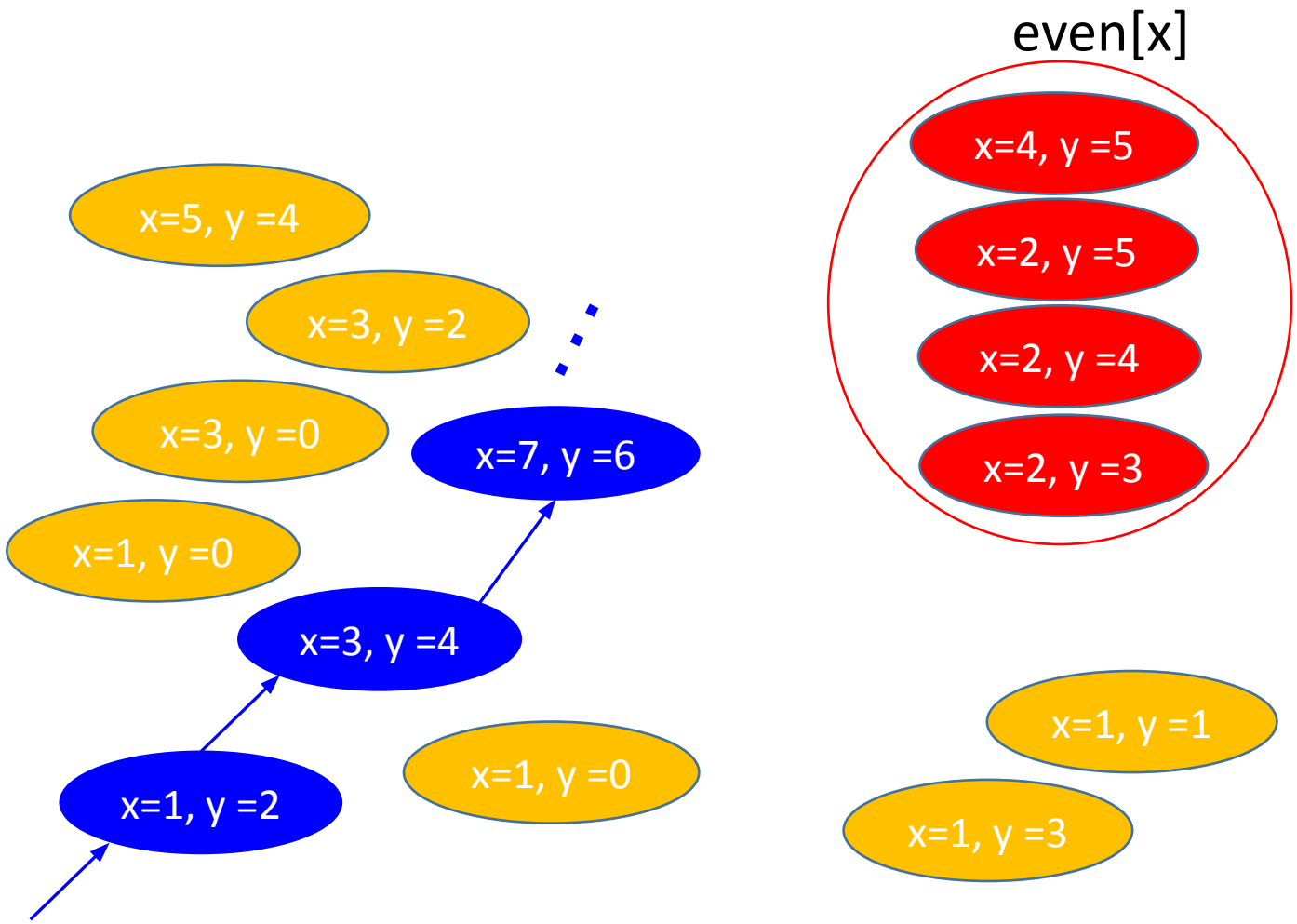


- A CTI may indicate:
  - A bug in the system
  - A bug in the safety property
  - A bug in the inductive invariant
    - Too weak
    - Too strong

Simple example: loop invariants

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

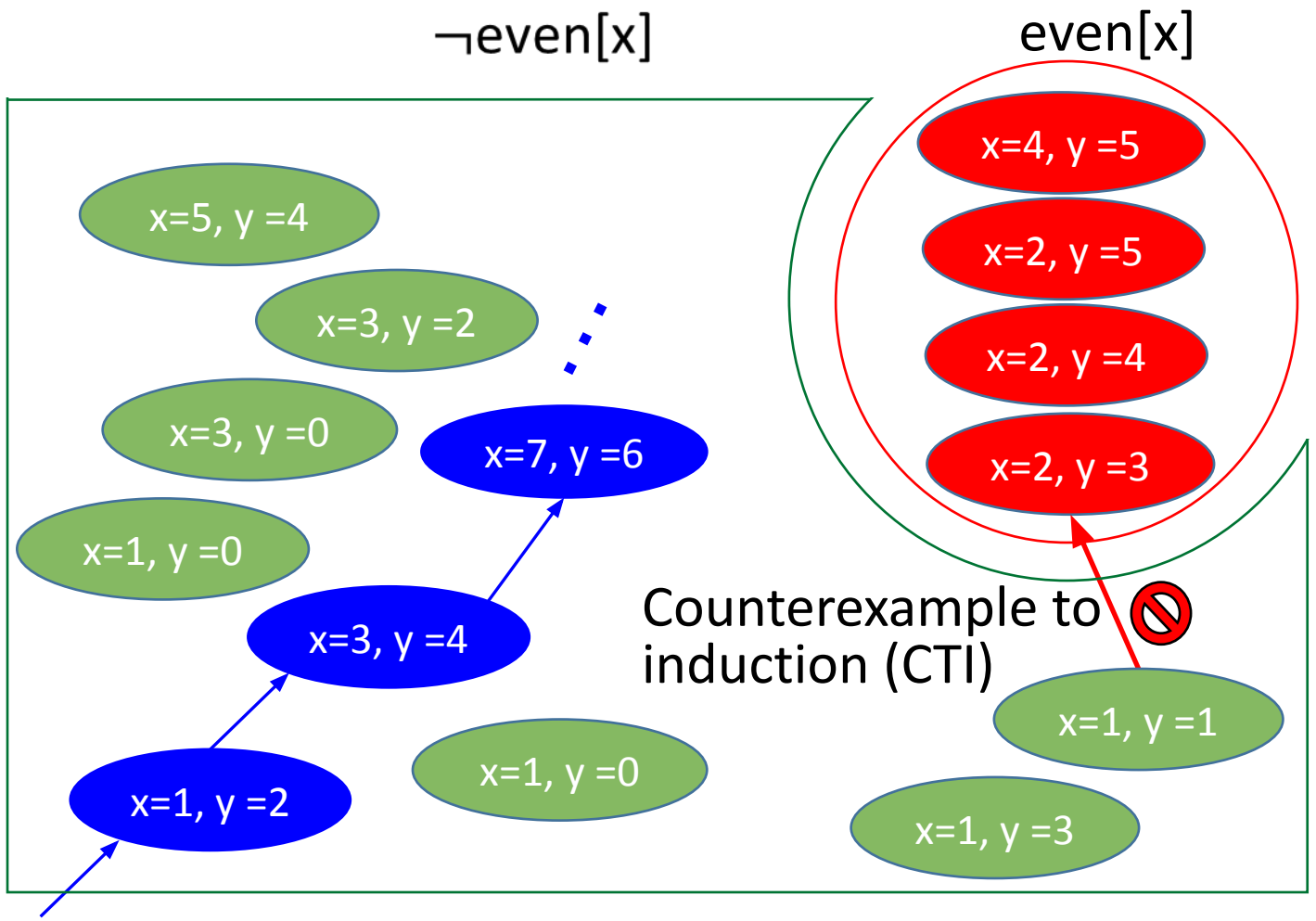
TR



Simple example: loop invariants

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

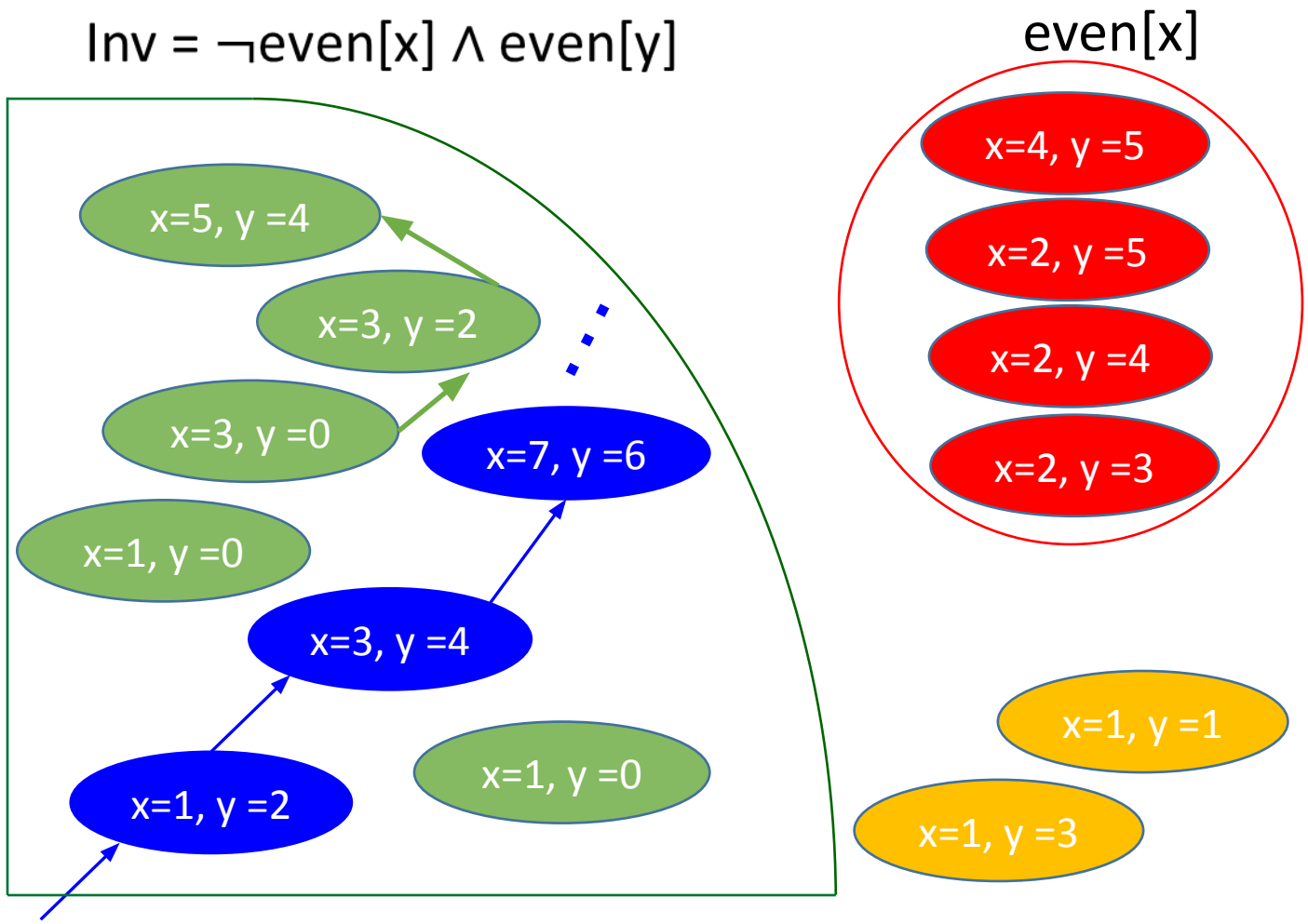
TR



Simple example: loop invariants

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

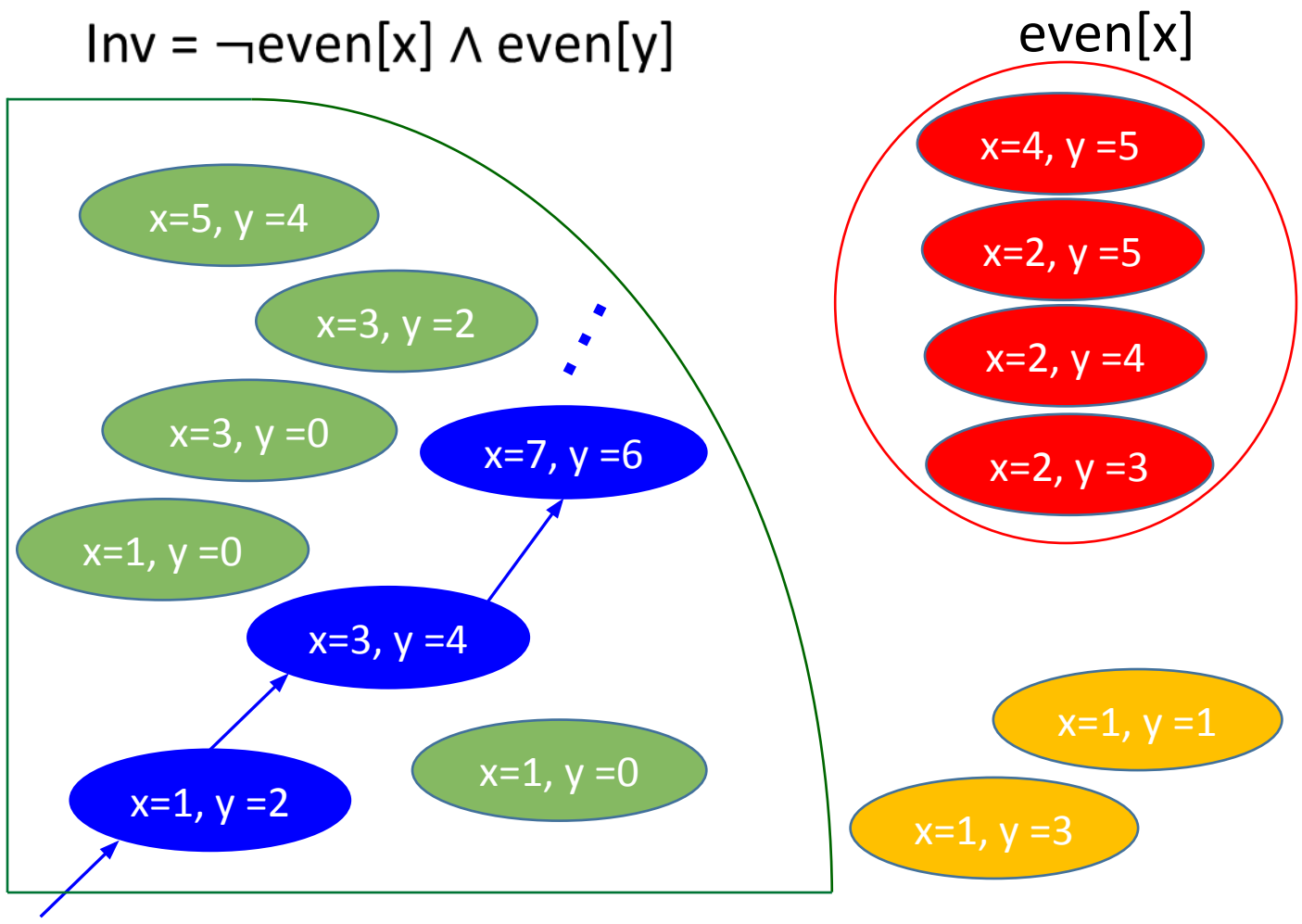
TR



Simple example: loop invariants

```
x := 1;
y := 2;
while * do {
 assert ¬even[x];
 x:=(x*x–y*y)/(x-y);
 y := y + 2;
}
```

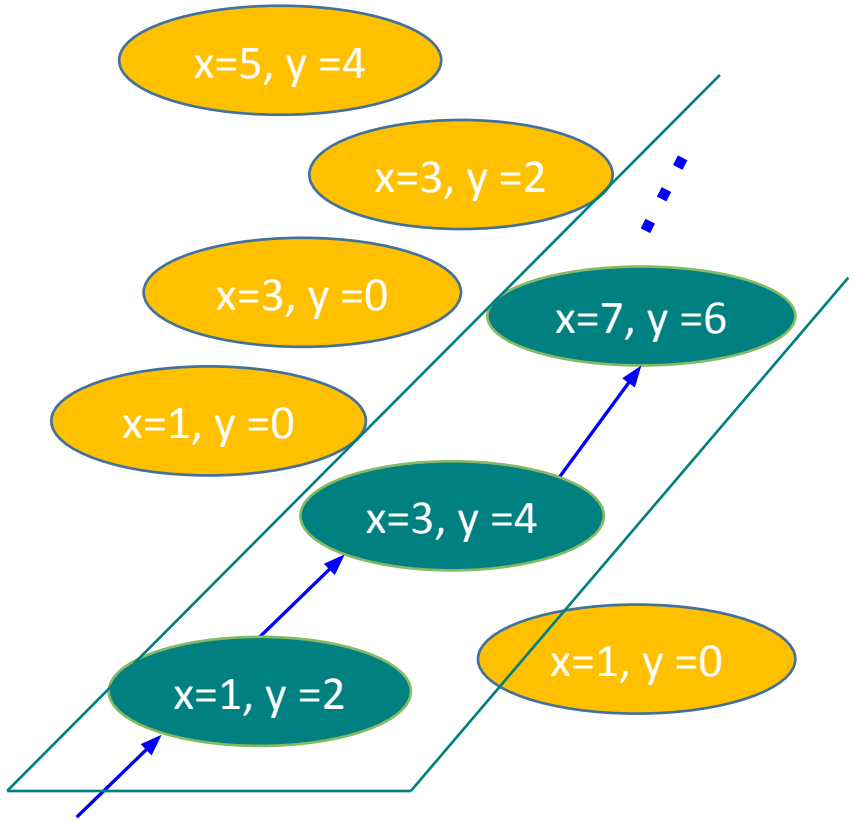
TR



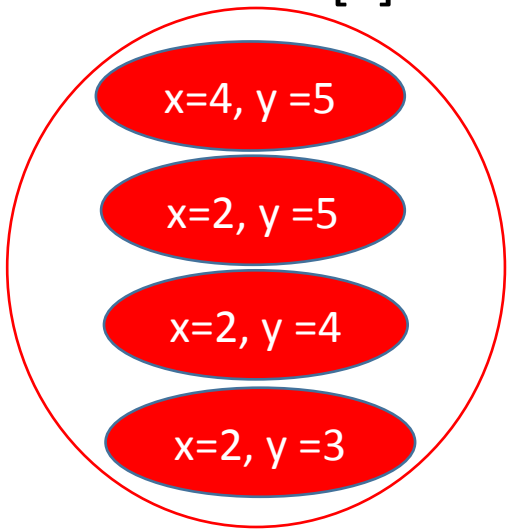
Simple example: loop invariants

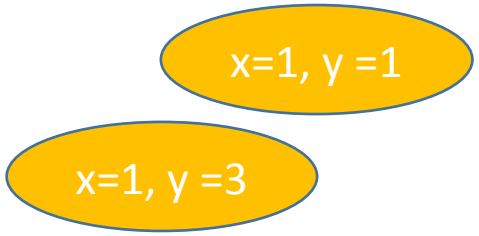
```
x := 1;
y := 2;
while * do {
 assert ¬even[x];
```
TR
```
 x := x + y;
 y := y + 2;
}
```

Inv = $y^2 - 2y - 4x + 4 = 0$

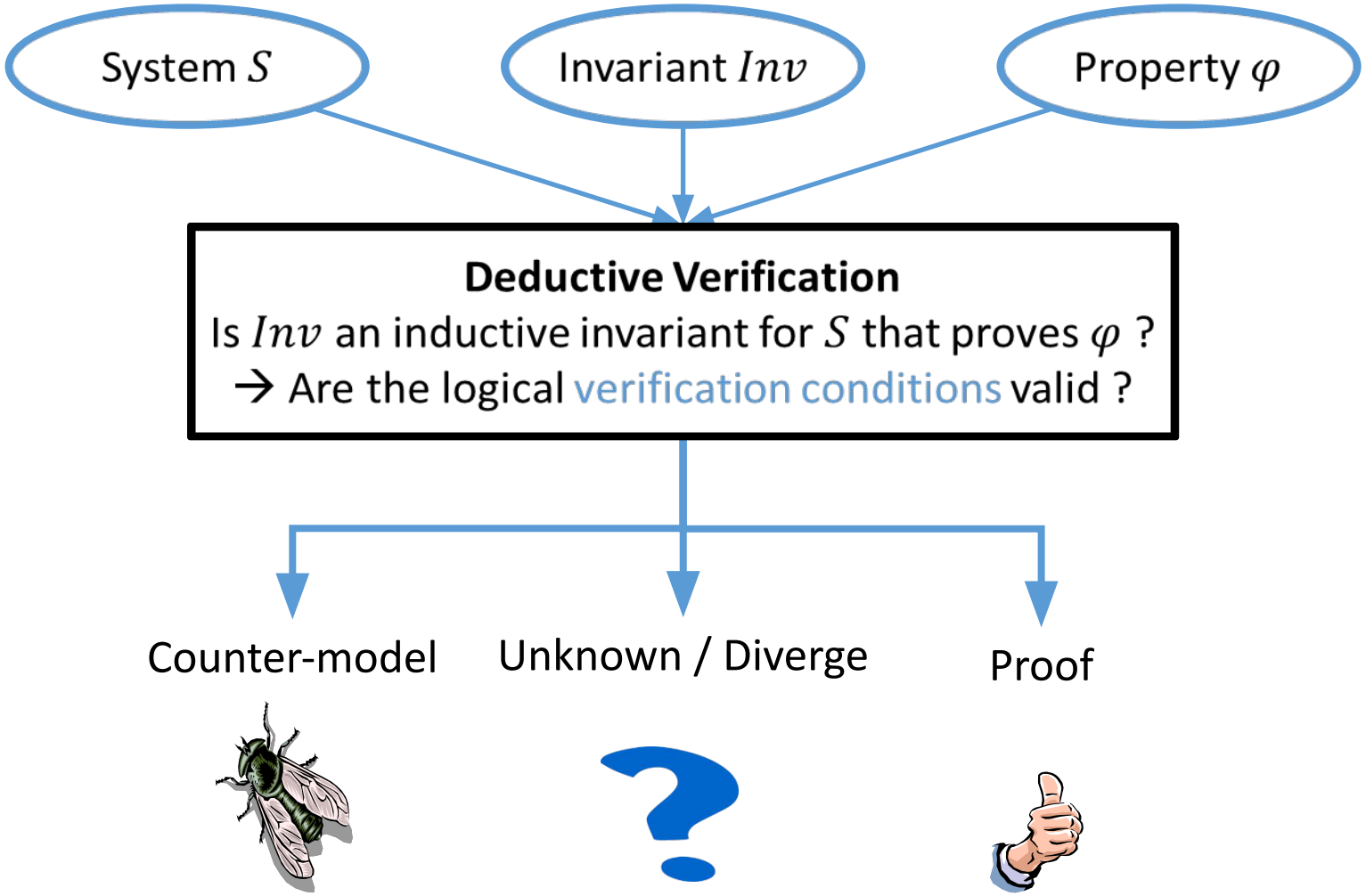


even[x]





Deductive verification

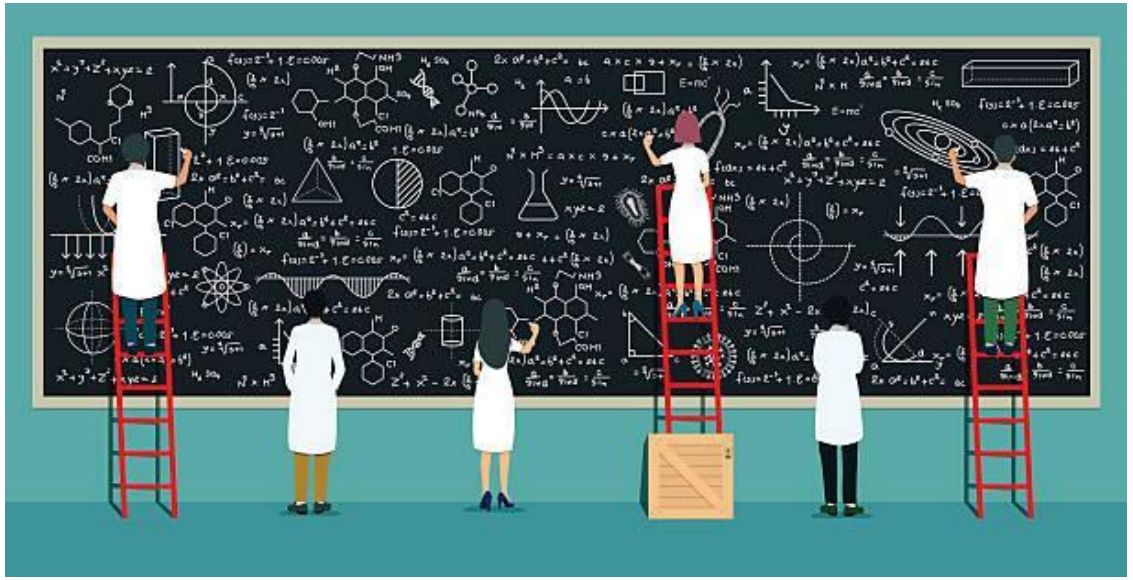


Verifying distributed systems is hard

Verdi

Verification of Raft in Coq

50,000 lines of manual proof



IronFleet

Verification of Multi-Paxos in Dafny

12,000 lines and 3.7 person-years

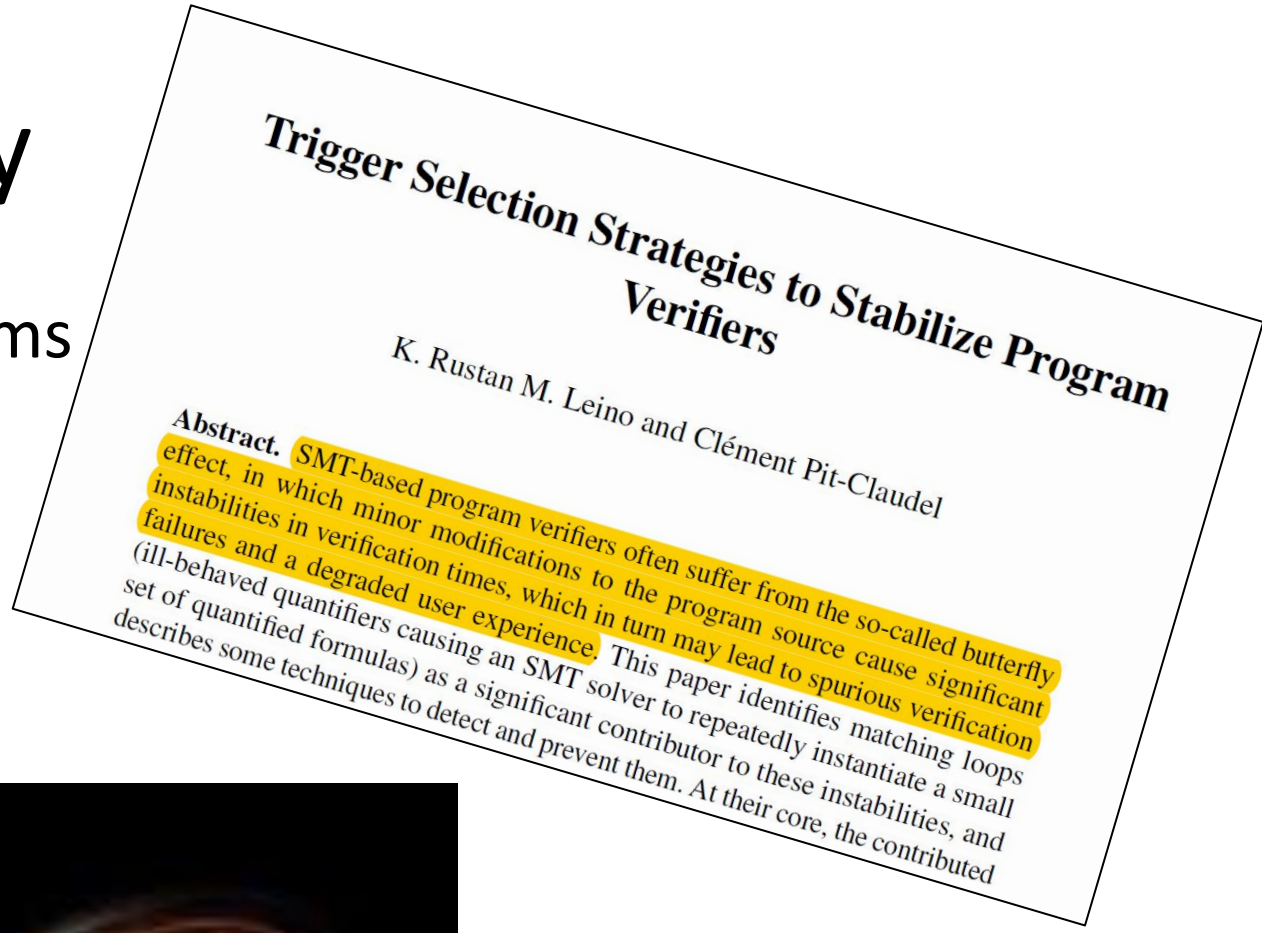
Uses solver for undecidable SMT checks

[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

Effects of undecidability

- The verifier may fail on tiny programs
- No explanation when tactics fails
  - Counterproofs
- The butterfly effect



Trigger Selection Strategies to Stabilize Program Verifiers

K. Rustan M. Leino and Clément Pit-Claudel

Abstract. SMT-based program verifiers often suffer from the so-called butterfly effect, in which minor modifications to the program source cause significant instabilities in verification times, which in turn may lead to spurious verification failures and a degraded user experience. This paper identifies matching loops (ill-behaved quantifiers causing an SMT solver to repeatedly instantiate a small set of quantified formulas) as a significant contributor to these instabilities, and describes some techniques to detect and prevent them. At their core, the contributed

The challenge of automated deduction

from: Ferraiulo, Baumann, Hawblitzel, Parno, "Komodo: Using Verification to Disentangle Secure-enclave Hardware from Software", SOSP '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

Rich SMT Theories

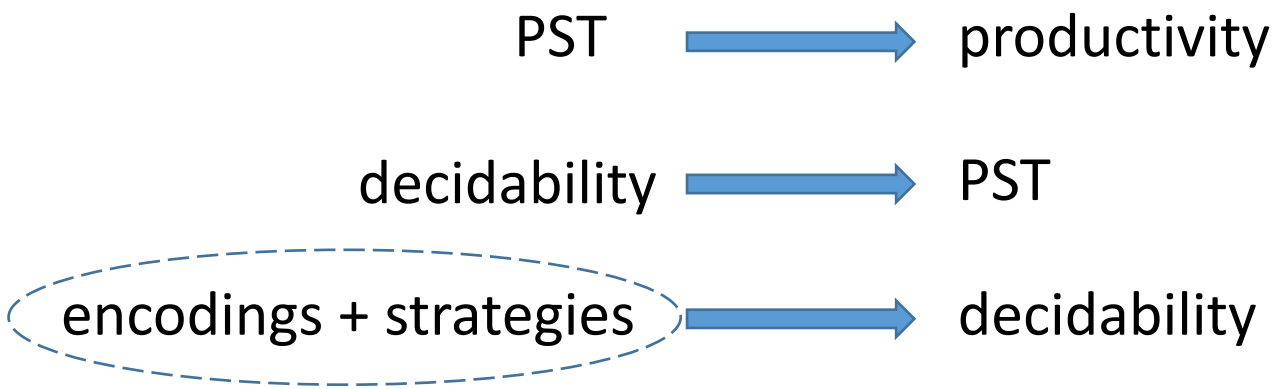
- Linear arithmetic, Bitvectors, Arrays, Strings, Datatypes, …
- Great tools: Yices, Z3, CVC4, Boolector, MathSAT, SMTInterpol, …
- Essential in Dafny, Sage, Klee, F*, ….
- Hides complexity from the user
- But solvers are heuristic, unpredictable, and not transparent

The Ivy deductive verification system – design hypotheses

PST

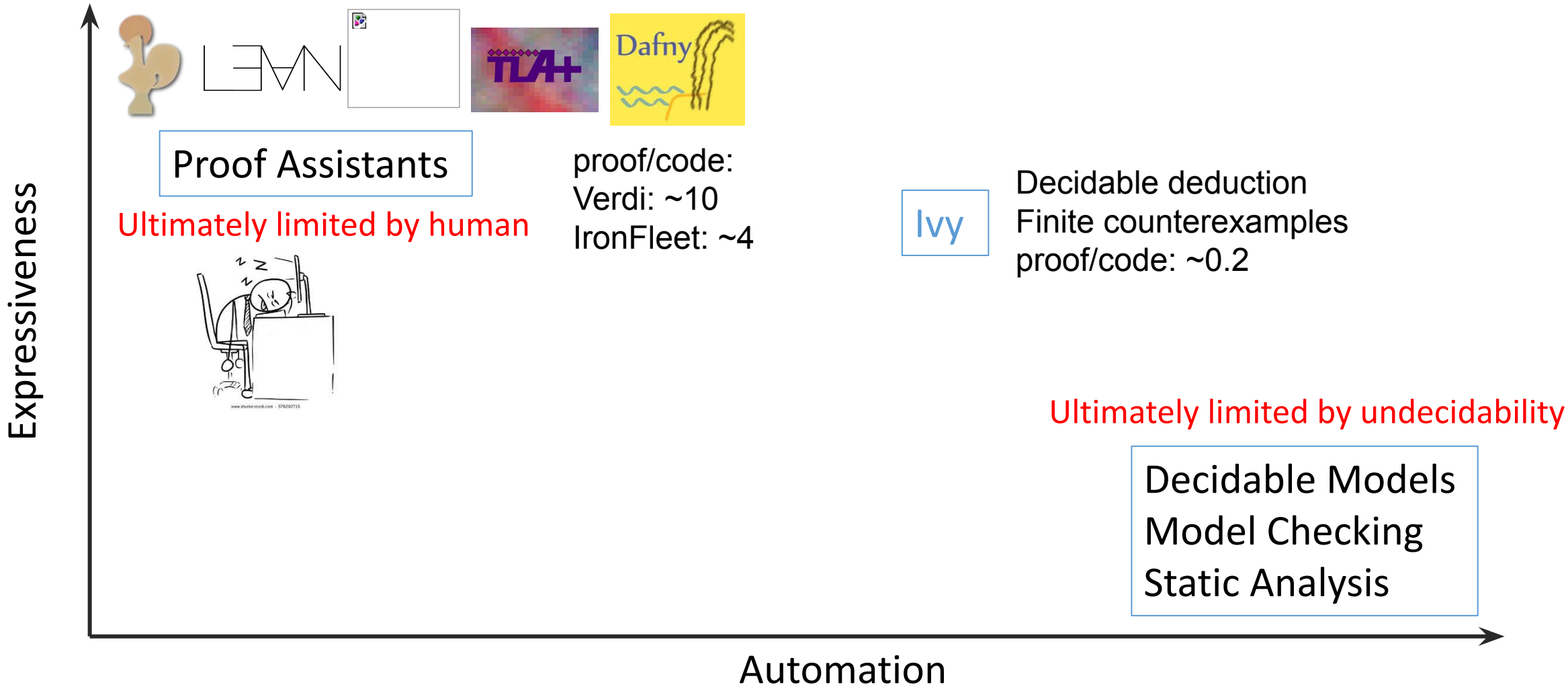
- Predictability: we can reasonably predict solver performance
- Stability (or continuity): small, irrelevant changes do not greatly affect performance
- Transparency: failures have understandable explanations

Hypotheses:



[PLDI'16] O.P., Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, Sharon Shoham.
Ivy: Safety Verification by Interactive Generalization
[CAV'20] O.P., Kenneth L. McMillan. Ivy: A Multi-modal Verification Tool for Distributed Algorithms

Design space for formal verification



Ivy's principles

- Specify systems and properties in decidable fragment of first-order logic
  - Allows quantifiers to reason about unbounded sets
  - Decidable to check inductiveness
  - Finite counterexamples to induction, display graphically or textually
- Interact with the user to find inductive invariants
- Challenge: use restricted logic to verify interesting systems
  - Transitive closure: network topology
  - Sets and cardinalities: Paxos, Byzantine Fault Tolerance, Reconfiguration
  - Liveness and temporal properties

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

- Limited fragment of first-order logic without theories
  - Relations, but no functions
  - Restricted quantifier prefix: $\exists^*\forall^*\varphi_{QF}$

$r(a,b)$ ✓ $r(a,f(a))$ ⊘

$\exists x \forall y. r(x,y)$ ✓

$\forall x \exists y. r(x,y)$ ⊘

- 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

F. Ramsey. On a problem in formal logic. Proc. London Math. Soc. 1930

EPR Satisfiability

Skolem

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

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

Herbrand

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

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

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

- Limited fragment of first-order logic without theories
  - Relations, but no functions
  - 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

F. Ramsey. On a problem in formal logic. Proc. London Math. Soc. 1930

Decidable Fragments in Ivy

- Many-sorted EPR: allows acyclic functions and quantifier alternations
  - E.g., $f: A \rightarrow \mathrm{B}$ but not $g: B \rightarrow A$
  - Maintains finite model property
  - Finite complete instantiations, supported by Z3

Protocol Designs

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

System Implementations

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

Logic-based deductive verification in Ivy

System S: $Init, TR$

Inductive invariant: Inv

Property $\varphi = \neg Bad$

Verification Condition (VC) Generator
VCs ≡ Is Inv an inductive invariant for S that proves φ ?

Fragment Checker
Are the logical VCs in the decidable fragment ?

Clear Explanation

Transparency

Automated Solver
Are the logical VCs valid ?

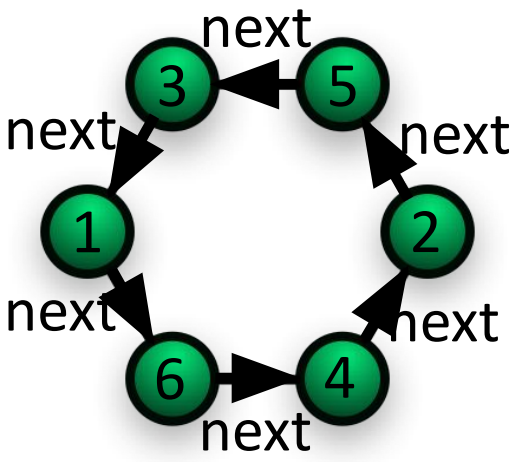
Proof

Counterexample to Induction

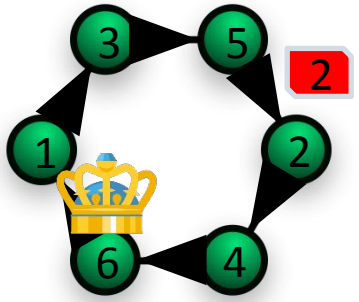
Challenge: encoding into EPR

- Many-sorted EPR: allows acyclic functions and quantifier alternations
  - E.g., $f: A \to B$ without $g: B \to A$
  - 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

Example: Leader election in a ring

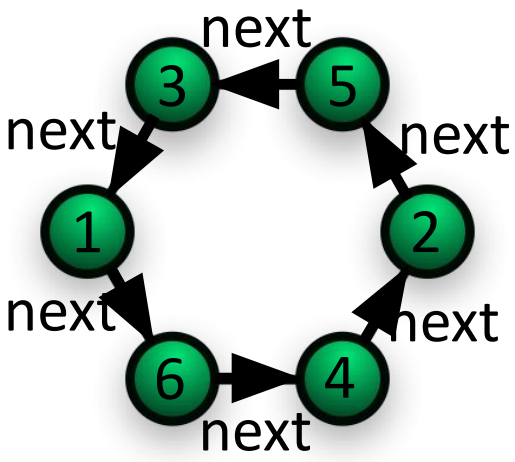


- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node's own id
  - A node that receives its own id becomes a leader
- Theorem: at any given time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

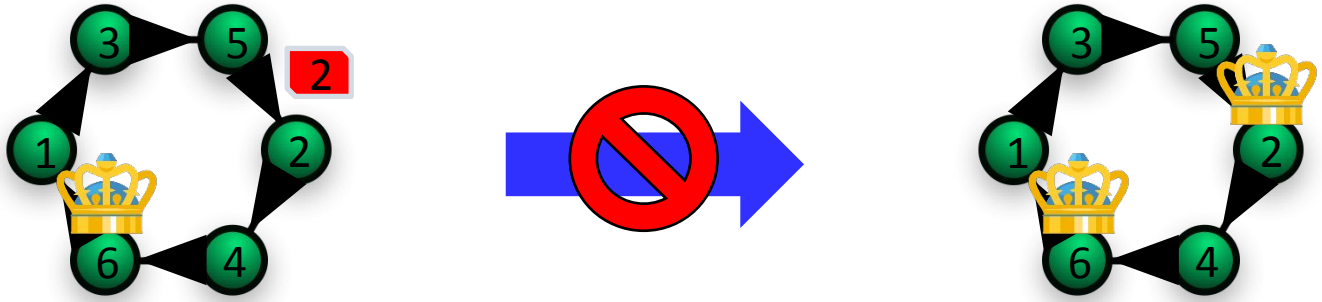


[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

Example: Leader election in a ring

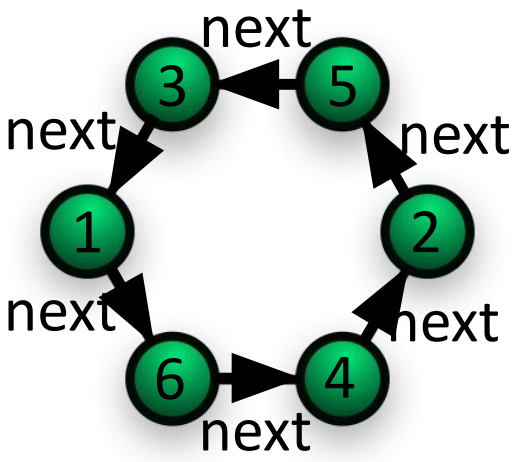


- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node's own id
  - A node that receives its own id becomes a leader
- Theorem: at any given time there is at most one leader
  - Inductive? NO



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

Example: Leader election in a ring



- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the ic
  - A noc
- Theorem

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 initiator. Only one message, that with the highest number, will not encounter a higher number on its way around. Thus, the only process getting its own message back is the one with the highest number.

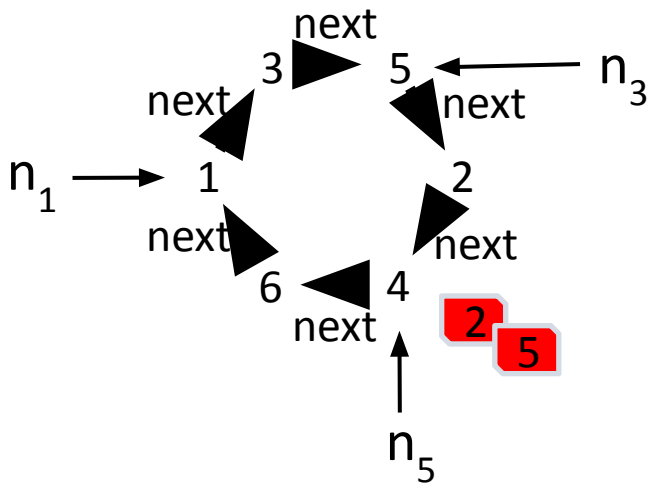
[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

Leader election protocol – first-order logic

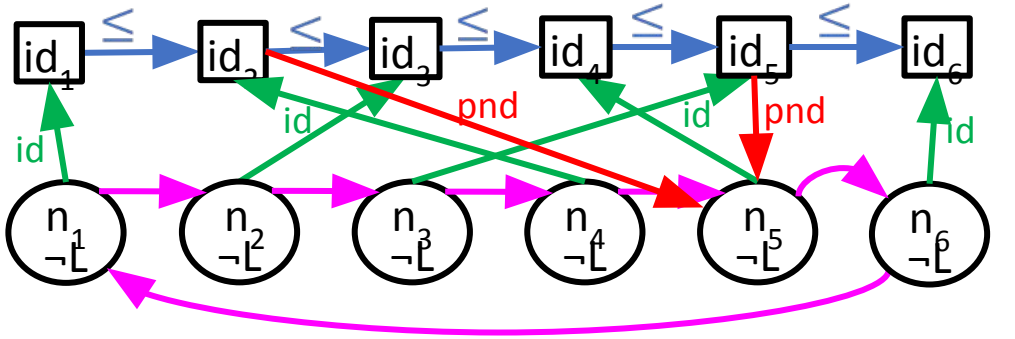
- **≤** (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



first-order structure



$\langle n_5, n_1, n_3 \rangle \in I(\mathbf{btw})$

Leader election protocol – first-order logic

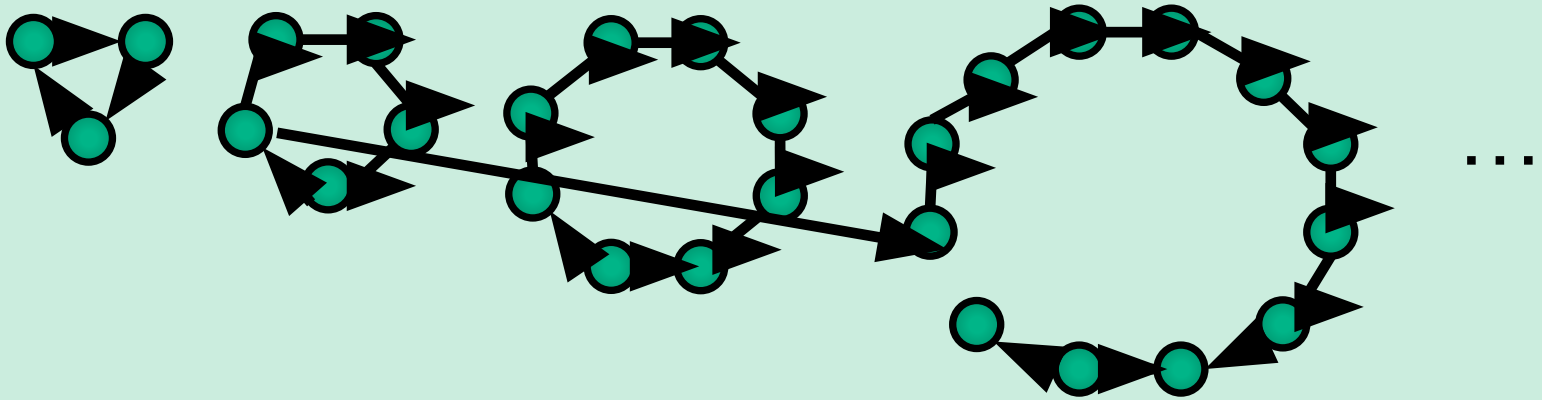
- ≤ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id:** Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state

first-order structure

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



…

Leader election protocol – first-order logic

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

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

```
action receive(n: Node, m: ID)
  requires pending(m, n)
  if * then
    pending(m, n) := false
  if id(n) = m then
    leader(n) := true
  else if id(n) ≤  m then
    "s := next(n)"
    pending(m, s) := true
```

TR:

```
∃n,s: Node. "s = next(n)" ∧ ∀x:ID,y:Node. pending'(x,y)↔(pending(x,y)∨(x=id(n)∧y=s))
```

Bad:

```
assert I0 = ∀ x,y: Node. leader(x)∧leader(y) → x = y
```

Leader election protocol – inductive invariant

Safety property: I_0

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

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

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

The leader has the highest ID

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

Only the leader can be self-pending

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

Cannot bypass higher nodes

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

Leader election protocol – inductive invariant

Safety property: I_0

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

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

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

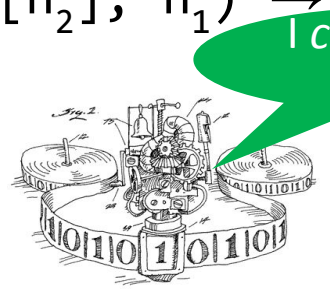
The leader has the highest ID

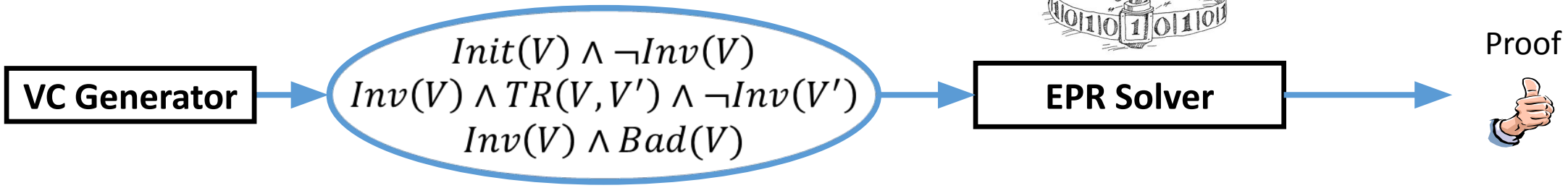
$$I_2 = \forall n_1, n_2: \text{Node}.\ pending(id[n_2], n_2) \rightarrow id[n_1] \leq id[n_2]$$

Only the leader can be self-pending

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

Cannot bypass higher nodes



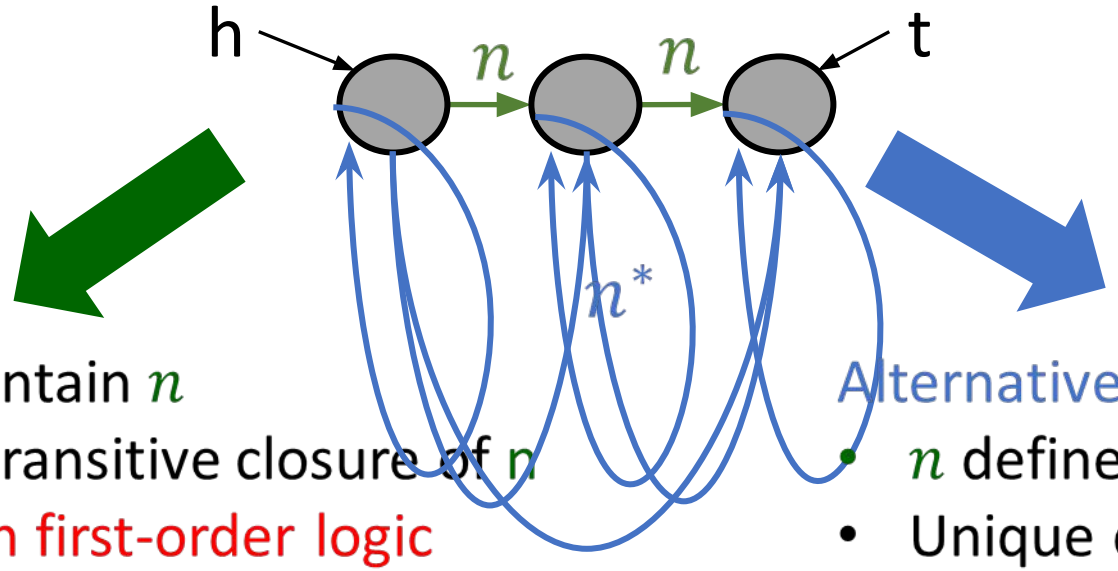


Principle: first-order abstractions

Concept	Intention	First-order abstraction
Node ID’s	Integers	
Ring Topology	Next edges + Transitive closure	

Key idea: representing deterministic paths

[Shachar Itzhaky PhD, SIGPLAN Dissertation Award 2016]



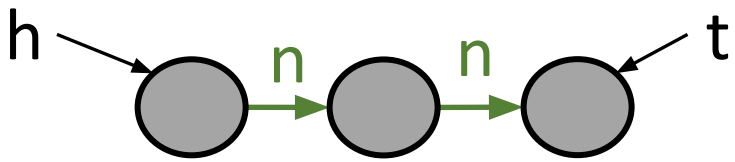
Alternative 1: maintain n

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

Alternative 2: maintain n^*

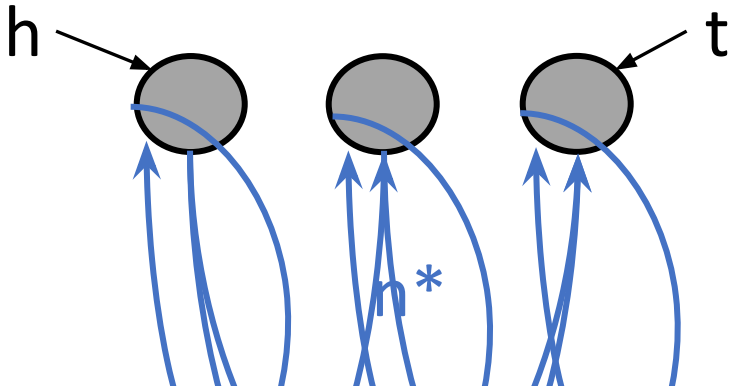
- n defined by transitive reduction of n^*
- Unique due to outdegree ≤ 1
- Definable in first order logic

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



Not first order expressible

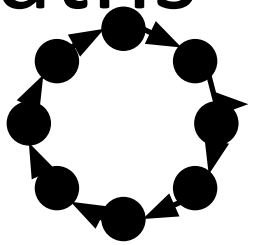
First order expressible



Sound and complete* axiomatization of deterministic paths

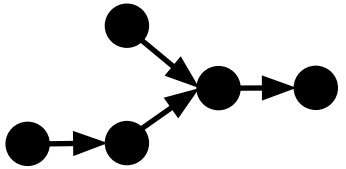
Line

$\leq (x, y)$



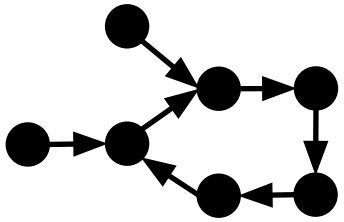
Ring

$btw(x, y, z)$



Forest, Tree,
Acyclic partial function

$\leq (x, y)$



Graph with out degree 1,
General partial function

$p(x, y, z)$

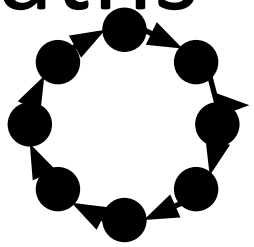
For every class C of finite graphs above:

- 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
 finite model property
+ Completeness Thm. for finite structures

Sound and complete automatic deductive verification

For every class C of finite graphs above:

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

Challenge: encoding into 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

Paxos

- Family of distributed consensus protocols
  - let nodes maintain consistent state under crashes and packet loss
- Variants for different tradeoffs and extra features
- Active research and extensive industry use
  - Pervasive approach to fault-tolerant distributed computing

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!
- Axiomatize quorums in first-order logic:

  sort Quorum

  relation $\boldsymbol{m}$(Node, Quorum)

  – set membership (2nd-order logic in first-order)

  axiom $\forall q_1, q_2$: Quorum. $\exists n$: Node. $\boldsymbol{m}(n, q_1) \wedge \boldsymbol{m}(n, q_2)$

```
action propose(r:Round) {
 require |{n | φ(n)}| > N/2
 …
}
```

```
action propose(r:Round) {
 require ∃q. ∀n. m(n, q) → φ(n)
 …
}
```

✓ EPR

[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

Principle: first-order abstractions

Concept	Intention	First-order abstraction
Messages	Network semantics: dropping duplication reordering	**relation** start_msg(Round) **relation** join_msg(Node, Round, Round, Value) **relation** propose_msg(Round, Value) **relation** vote_msg(Node, Round, Value)
Quorums	Majority sets	
Byzantine Quorums	2/3 Majority	

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 }
```

$$\forall n_1, n_2 : \text{node}, r_1, r_2 : \text{round}, v_1, v_2 : \text{value}.\ decision(n_1, r_1, v_1) \wedge decision(n_2, r_2, v_2) \rightarrow v_1 = v_2$$

$$\forall r : \text{round}, v_1, v_2 : \text{value}.\ propose\_msg(r, v_1) \wedge propose\_msg(r, v_2) \rightarrow v_1 = v_2$$

$$\forall n : \text{node}, r : \text{round}, v : \text{value}.\ vote\_msg(n, r, v) \rightarrow propose\_msg(r, v)$$

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

$$\forall n : \text{node}, r, r' : \text{round}, v, v' : \text{value}.\ join\_ack\_msg(n, r, \bot, v) \wedge r' < r \rightarrow \neg vote\_msg(n, r', v')$$

$$\forall n : \text{node}, r, r' : \text{round}, v : \text{value}.\ join\_ack\_msg(n, r, r', v) \wedge r' \neq \bot \rightarrow r' < r \wedge vote\_msg(n, r', v)$$

$$\forall n : \text{node}, r, r', r'' : \text{round}, v, v' : \text{value}.join\_ack\_msg(n, r, r', v) \wedge r' \neq \bot \wedge r' < r'' < r \rightarrow \neg vote\_msg(n, r'', v')$$

$$\forall n : \text{node}, v : \text{value}.\ \neg vote\_msg(n, \bot, v)$$

$$\forall r_1, r_2 : \text{round}, v_1, v_2 : \text{value}, q : \text{quorum}.\ propose\_msg(r_2, v_2) \wedge r_1 < r_2 \wedge v_1 \neq v_2 \rightarrow$$
$$\exists n : \text{node}, r', r'' : \text{round}, v : \text{value}.\ member(n, q) \wedge \neg vote\_msg(n, r_1, v_1) \wedge r' > r_1 \wedge join\_ack\_msg(n, r', r'', v)$$

VCs in first-order logic

Challenge: encoding into 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

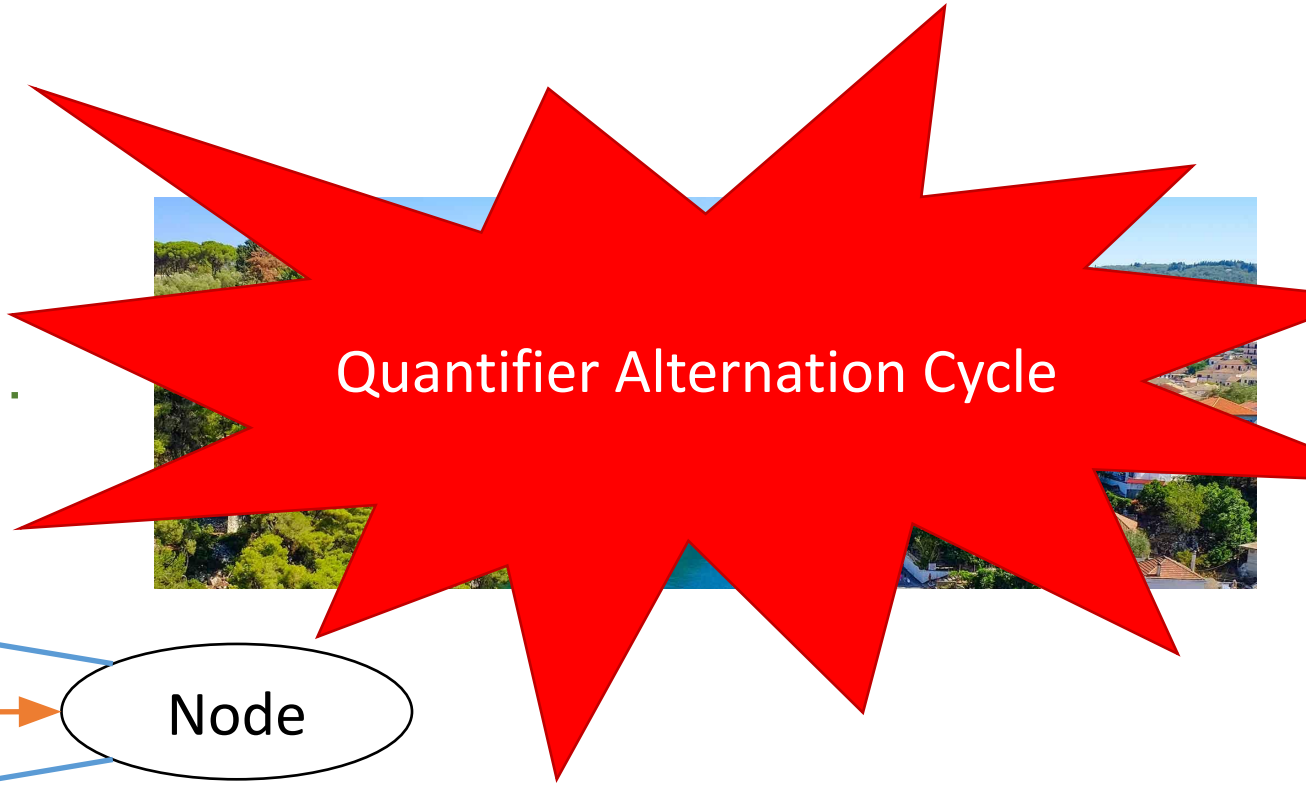
  $\forall q_1, q_2$: Quorum. $\exists$ n: Node. …

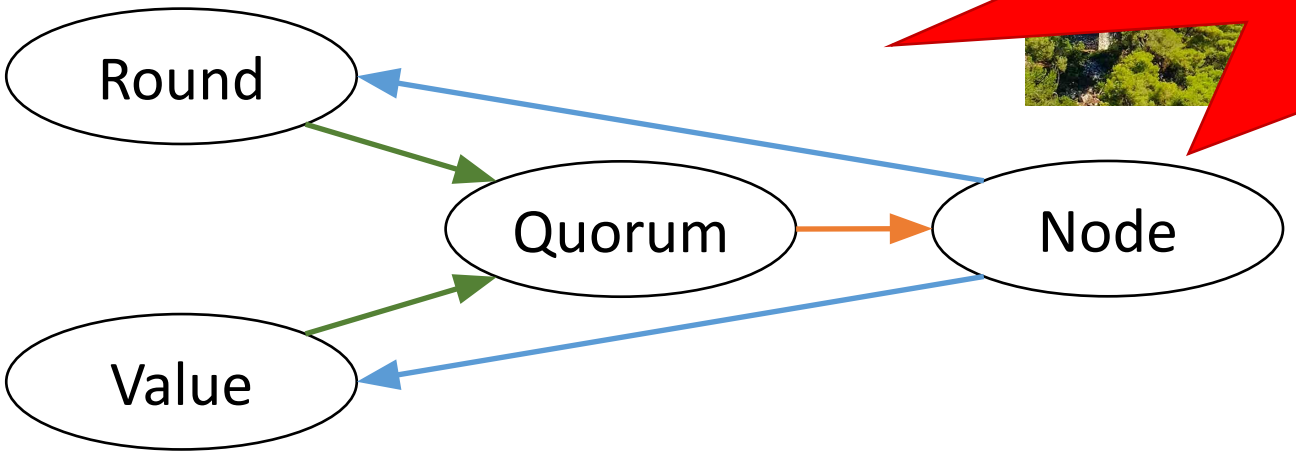
- Action precondition

  … $\forall$ n:Node. … $\rightarrow$ $\exists$ r:Round,v:Value. …

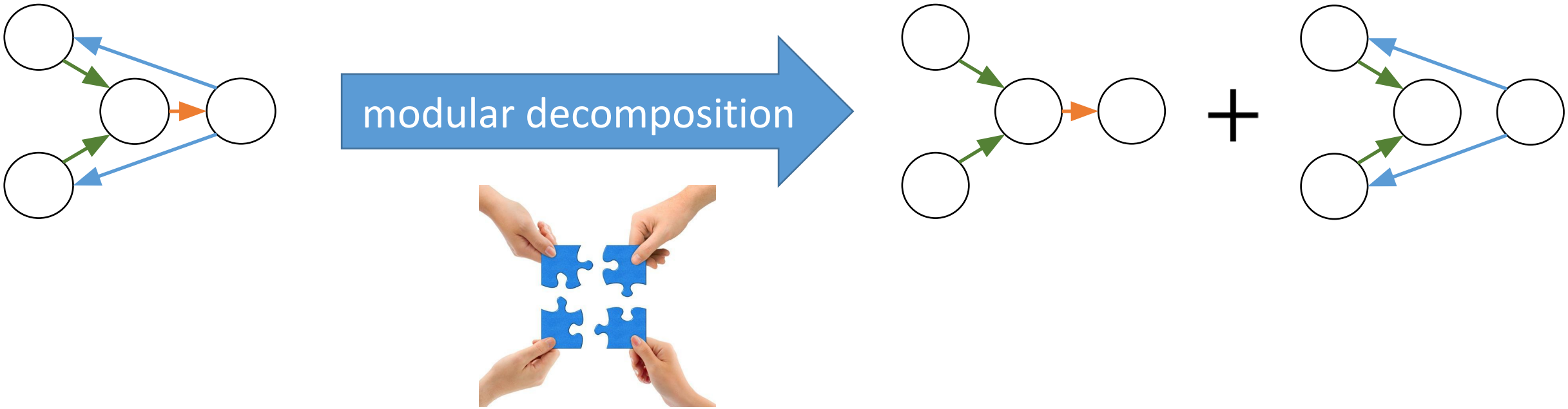
- Inductive invariant

  $\forall$ r:Round, v:Value. … $\rightarrow$ $\exists$ q:Quorum. …





Modularity



[PLDI 2018] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, O. P., Mooly Sagiv, Sharon Shoham, James R. Wilcox, Doug Woos

Modularity for decidability of deductive verification with applications to distributed systems

Inductive invariant of Paxos

```
# safety property
invariant decision(N1,R1,V1) & decision(N2,R2,V2) -> V1 = V2
# proposals are unique per round
invariant proposal(R,V1) & proposal(R,V2) -> V1 = V2
# only vote for proposed values
invariant vote(N,R,V) -> proposal(R,V)
# decisions come from quorums of votes:
invariant forall R, V. (exists N. decision(N,R,V)) -> exists Q. forall N. member(N, Q) -> vote(N,R,V)
# properties of one_b_max_vote
invariant one_b_max_vote(N,R2,none,V1) & ~le(R2,R1) -> ~vote(N,R1,V2)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none -> ~le(R,RM) & vote(N,RM,V)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none & ~le(R,RO) & ~le(RO,RM) -> ~vote(N,RO,VO)
# property of choosable and proposal
invariant ~le(R2,R1) & proposal(R2,V2) & V1 ~= V2 -> exists N. member(N,Q) & left_rnd(N,R1) & ~vote(N,R1,V1)
# property of one_b, left_rnd
invariant one_b(N,R2) & ~le(R2,R1) -> left_rnd(N,R1)
```

Paxos made EPR: experimental evaluation

Protocol	Model [LOC]	Invariant [Conjectures]		
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos*	123	18	2.2	0.2
Fast Paxos*	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos*	132	16	5.4	0.9

*first mechanized verification

Transformation to EPR reusable across all variants!

[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

Paxos made EPR: experimental evaluation

Protocol	Model [LOC]	Invariant [Conjectures]		
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos*	123	18	2.2	0.2
Fast Paxos*	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos*	132	16	5.4	0.9

Proof / code ratio:

IronFleet: ~4

Verdi: ~10

Ivy: ~0.2

*first mechanized verification

Transformation to EPR reusable across all variants!

[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

Paxos made EPR: experimental evaluation

Protocol	Model [LOC]	Invariant [Conjectures]		
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos*	123	18	2.2	0.2
Fast Paxos*	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos*	132	16	5.4	0.9

μ – mean

σ – std. deviation

*first mechanized verification

Transformation to EPR reusable across all variants!

[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

Paxos made EPR: experimental evaluation

Protocol	Model [LOC]	Invariant [Conjectures]		
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos*	123	18	2.2	0.2
Fast Paxos*	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos*	132	16	5.4	0.9

Rounds			T.O.
2	1.2	0.1	0
4	1.8	0.4	0
8	107	129	30%
16	229	110	70%

Multi-Paxos in FOL

*first mechanized verification

Transformation to EPR reusable across all variants!

[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

Paxos made EPR: experimental evaluation

Protocol	Model [LOC]	Invariant [Conjectures]		
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos*	123	18	2.2	0.2
Fast Paxos*	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos*	132	16	5.4	0.9

Rounds			T.O.
2	186	123	50%
4	300	0	100%
8	300	0	100%
16	300	0	100%

Stoppable Paxos in FOL

*first mechanized verification

Transformation to EPR reusable across all variants!

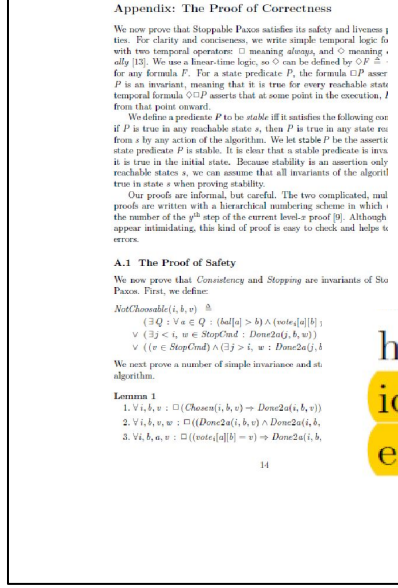
[OOPSLA'17] O.P., Giuliano Losa, Mooly Sagiv, Sharon Shoham. Paxos made EPR: Decidable Reasoning about Distributed Protocols

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.


Appendix: The Proof of Correctness

We now prove that Stoppable Paxos satisfies its safety and liveness p ties. For clarity and conciseness, we write simple temporal logic fo with two temporal operators: $\Box$ meaning *always*, and $\Diamond$ meaning *ally* [13]. We use a linear-time logic, so $\Diamond$ can be defined by $\Diamond P \triangleq$ for any formula F. For a state predicate P, the formula $\Box P$ asser P is an invariant, meaning that it is true for every reachable state temporal formula $\Diamond\Box P$ asserts that at some point in the execution, I from that point onward.

We define a predicate P to be *stable* iff it satisfies the following con if P is true in any reachable state s, then P is true in any state re from s by any action of the algorithm. We let stable P be the asserti state predicate P is stable. It is clear that a stable predicate is inva it is true in the initial state. Because stability is an assertion only reachable states s, we can assume that all invariants of the algorit true in state s when proving stability.

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

A.1 The Proof of Safety

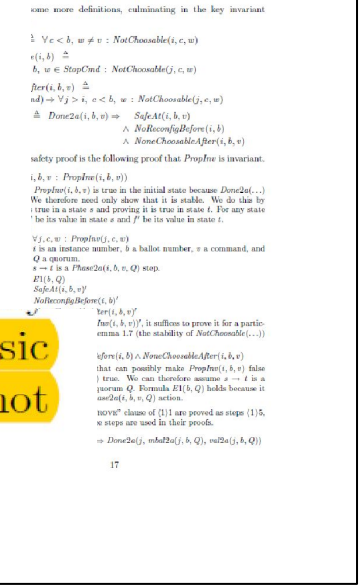
We now prove that *Consistency* and *Stopping* are invariants of Sto Paxos. First, we define:

$NotChoosable(i, b, v) \triangleq$
$(\exists Q : \forall a \in Q : (bal[a] > b) \wedge (vote_i[a][b] \dots$
$\vee (\exists j < i, w \in StopCmd : Done2a(j, b, w))$
$\vee (\{v \in StopCmd\} \wedge (\exists j > i, w : Done2a(j, b$

We next prove a number of simple invariance and st algorithm.

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

14


some more definitions, culminating in the key invariant

$\forall c < b, w \neq v : NotChoosable(i, c, w)$

$e(i, b) \triangleq$
$b, w \in StopCmd : NotChoosable(j, c, w)$

$fter(i, b, v) \triangleq$
$nd) \Rightarrow \forall j > i, c < b, w : NotChoosable(j, c, w)$

$\triangleq Done2a(i, b, v) \Rightarrow SafeAt(i, b, v)$
$\wedge NoReconfigBefore(i, b)$
$\wedge NoneChoosableAfter(i, b, v)$

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

$i, b, v : PropInv(i, b, v))$
PropInv(i, b, v) is true in the initial state because *Done2a*(...) We therefore need only show that it is stable. We do this by true in a state s and proving it is true in state t. For any state be its value in state s and f' be its value in state t.

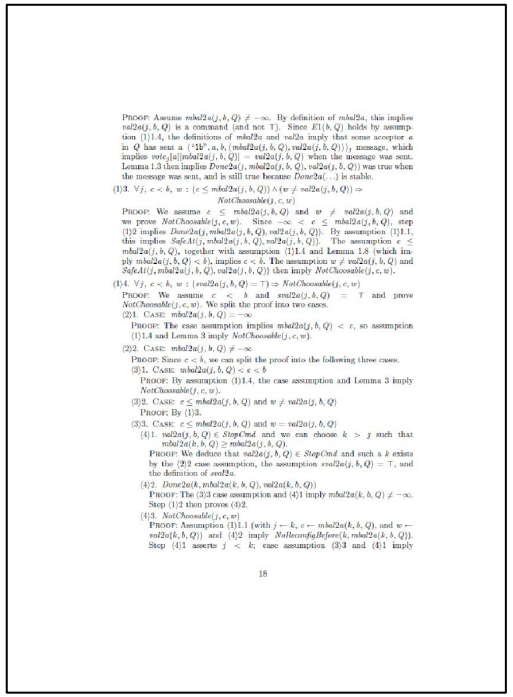
$\forall j, c, w : PropInv(j, c, w)$
c is an instance number, b a ballot number, v a command, and Q a quorum.
$s \rightarrow t$ is a $Phase2a(b, v, Q)$ step.
$E1(b, Q)$
$SafeAt(i, b, v)'$
$NoReconfigBefore(i, b)'$
$\dots$ $Ier(i, b, v)'$
$Inv(i, b, v))'$, it suffices to prove it for a partic- emma 1.7 (the stability of $NotChoosable(\dots)$)

$nfore(i, b) \wedge NoneChoosableAfter(i, b, v)$
that can possibly make $PropInv(i, b, v)$ false) true. We can therefore assume $s \rightarrow t$ is a quorum Q. Formula $E1(b, Q)$ holds because it $ose2a(i, b, v, Q)$ action.

move" clause of (1)1 are proved as steps (1)5, se steps are used in their proofs.

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

17


Proof: Assume $mbal2a(j, b, Q) \neq -\infty$. By definition of $mbal2a$, this implies $val2a(j, b, Q)$ is a command (and not $\top$). Since $E1(b, Q)$ holds by assumption (1)1.4, the definitions of $mbal2a$ and $val2a$ imply that some acceptor a in Q has sent a $(\text{"2b"}, a, b, \langle mbal2a(j, b, Q), val2a(j, b, Q)\rangle_1)$ message, which implies $vote_i[a][mbal2a(j, b, Q)] = val2a(j, b, Q)$ when the message was sent. Lemma 1.3 then implies $Done2a(i, mbal2a(j, b, Q), val2a(j, b, Q))$ was true when the message was sent, and is still true because $Done2a(\dots)$ is stable.

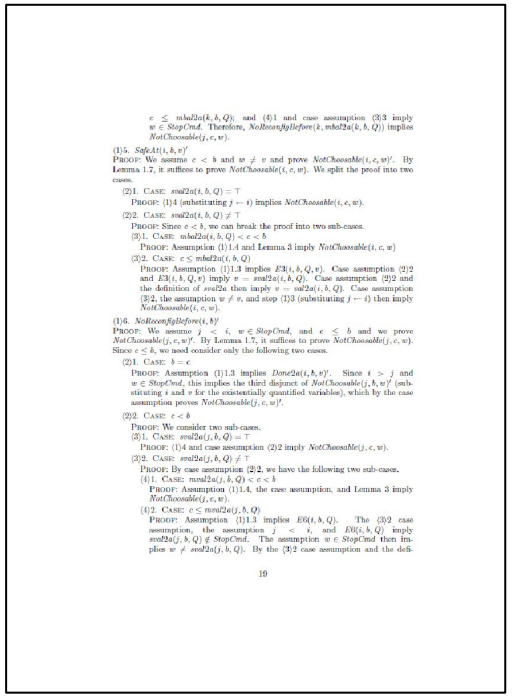
(1)3. $\forall j, c < b, w : (c \leq mbal2a(j, b, Q)) \wedge (w \neq val2a(j, b, Q)) \Rightarrow NotChoosable(j, c, w)$

Proof: We assume $c \leq mbal2a(j, b, Q)$ and $w \neq val2a(j, b, Q)$ and we prove $NotChoosable(j, c, w)$. Since $-\infty < c \leq mbal2a(j, b, Q)$, step (1)2 implies $Done2a(j, mbal2a(j, b, Q), val2a(j, b, Q))$. By assumption (1)1.1, this implies $SafeAt(j, mbal2a(j, b, Q), val2a(j, b, Q))$. The assumption $c < mbal2a(j, b, Q)$, together with assumption (1)1.4 and Lemma 1.8 (which implies $mbal2a(j, b, Q) < b$), implies $c < b$. The assumption $w \neq val2a(j, b, Q)$ and $SafeAt(j, mbal2a(j, b, Q), val2a(j, b, Q))$ then imply $NotChoosable(j, c, w)$.

(1)4. $\forall j, c < b, w : (val2a(j, b, Q) = \top) \Rightarrow NotChoosable(j, c, w)$

Proof: We assume $c < b$ and $val2a(j, b, Q) = \top$ and prove $NotChoosable(j, c, w)$. We split the proof into two cases.
(2)1. Case: $mbal2a(j, b, Q) = -\infty$
Proof: The case assumption implies $mbal2a(j, b, Q) < c$, so assumption (1)1.4 and Lemma 3 imply $NotChoosable(j, c, w)$.
(2)2. Case: $mbal2a(j, b, Q) \neq -\infty$
Proof: Since $c < b$, we can split the proof into the following three cases.
(3)1. Case: $mbal2a(j, b, Q) < c < b$
Proof: By assumption (1)1.4, the case assumption and Lemma 3 imply $NotChoosable(j, c, w)$.
(3)2. Case: $c \leq mbal2a(j, b, Q)$ and $w \neq val2a(j, b, Q)$
Proof: By (1)3.
(3)3. Case: $c \leq mbal2a(j, b, Q)$ and $w = val2a(j, b, Q)$
(4)1. $val2a(j, b, Q) \in StopCmd$ and we can choose $k > j$ such that $mbal2a(k, b, Q) \geq mbal2a(j, b, Q)$.
Proof: We deduce that $val2a(j, b, Q) \in StopCmd$ and such a k exists by the (2)2 case assumption, the assumption $val2a(j, b, Q) = \top$, and the definition of $val2a$.
(4)2. $Done2a(k, mbal2a(k, b, Q), val2a(k, b, Q))$
Proof: The (3)3 case assumption and (4)1 imply $mbal2a(k, b, Q) \neq -\infty$. Step (1)2 then proves (4)2.
(4)3. $NotChoosable(j, c, w)$
Proof: Assumption (1)1.1 (with $j \leftarrow k$, $c \leftarrow mbal2a(k, b, Q)$, and $w \leftarrow val2a(k, b, Q)$) and (4)2 imply $NoReconfigBefore(k, mbal2a(k, b, Q))$. Step (4)1 asserts $j < k$; case assumption (3)3 and (4)1 imply

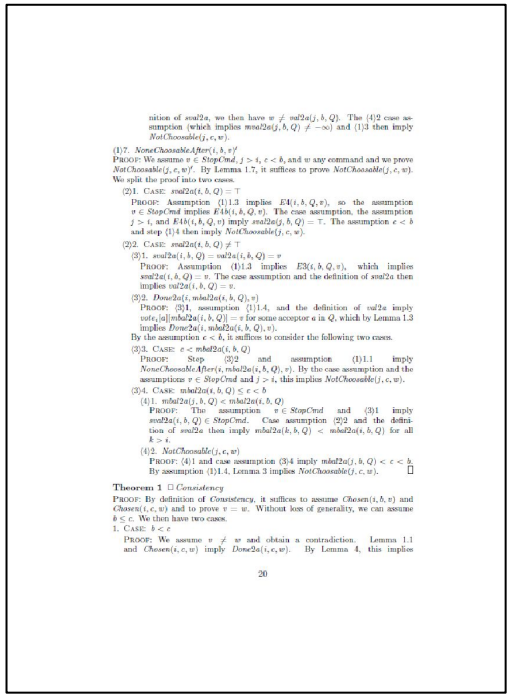
18


$c \leq mbal2a(k, b, Q)$; and (4)1 and case assumption (3)3 imply $w \in StopCmd$. Therefore, $NoReconfigBefore(k, mbal2a(k, b, Q))$ implies $NotChoosable(j, c, w)$.

(1)5. $SafeAt(i, b, v)'$
Proof: We assume $c < b$ and $w \neq v$ and prove $NotChoosable(i, c, w)'$. By Lemma 1.7, it suffices to prove $NotChoosable(i, c, w)$. We split the proof into two cases.
(2)1. Case: $val2a(i, b, Q) = \top$
Proof: Since $c < b$, we can break the proof into two sub-cases.
(3)1. Case: $mbal2a(i, b, Q) < c < b$
Proof: Assumption (1)1.4 and Lemma 3 imply $NotChoosable(i, c, w)$
(3)2. Case: $c \leq mbal2a(i, b, Q)$
Proof: Assumption (1)1.3 implies $E3(i, b, Q, v)$. Case assumption (2)2 and $E3(i, b, Q, v)$ imply $v = val2a(i, b, Q)$. Case assumption (2)2 and the definition of $val2a$ then imply $v = val2a(i, b, Q)$. Case assumption (3)2, the assumption $w \neq v$, and step (1)3 (substituting $j \leftarrow i$) then imply $NotChoosable(i, c, w)$.

(1)6. $NoReconfigBefore(i, b)'$
Proof: We assume $j < i$, $w \in StopCmd$, and $c \leq b$ and we prove $NotChoosable(j, c, w)'$. By Lemma 1.7, it suffices to prove $NotChoosable(j, c, w)$. Since $c \leq b$, we need consider only the following two cases.
(2)1. Case: $b = c$
Proof: Assumption (1)1.3 implies $Done2a(i, b, v)'$. Since $i > j$ and $w \in StopCmd$, this implies the third disjunct of $NotChoosable(j, b, w)'$ (substituting i and v for the existentially quantified variables), which by the case assumption proves $NotChoosable(j, c, w)'$.
(2)2. Case: $c < b$
Proof: We consider two sub-cases.
(3)1. Case: $val2a(j, b, Q) = \top$
Proof: (1)4 and case assumption (2)2 imply $NotChoosable(j, c, w)$.
(3)2. Case: $val2a(j, b, Q) \neq \top$
Proof: By case assumption (2)2, we have the following two sub-cases.
(4)1. Case: $mval2a(j, b, Q) < c < b$
Proof: Assumption (1)1.4, the case assumption, and Lemma 3 imply $NotChoosable(j, c, w)$.
(4)2. Case: $c \leq mval2a(j, b, Q)$
Proof: Assumption (1)1.3 implies $E6(i, b, Q)$. The (3)2 case assumption, the assumption $j < i$, and $E6(i, b, Q)$ imply $val2a(j, b, Q) \notin StopCmd$. The assumption $w \in StopCmd$ then implies $w \neq val2a(j, b, Q)$. By the (3)2 case assumption and the defi-

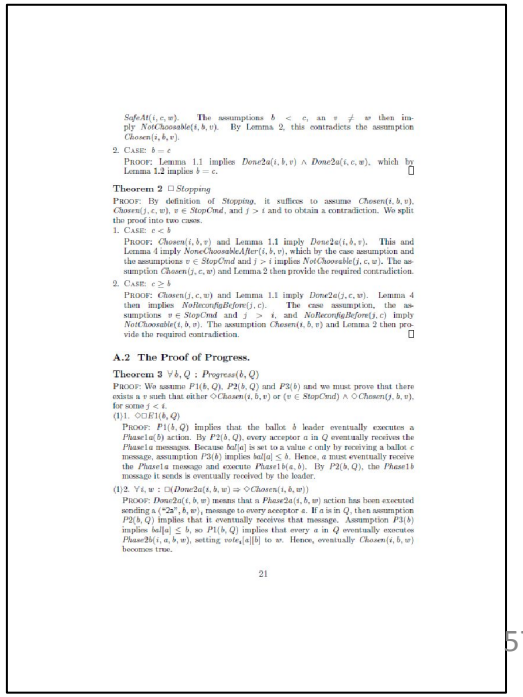
19


nition of $val2a$, we then have $w \neq val2a(j, b, Q)$. The (4)2 case assumption (which implies $mval2a(j, b, Q) \neq -\infty$) and (1)3 then imply $NotChoosable(j, c, w)$.

(1)7. $NoneChoosableAfter(i, b, v)'$
Proof: We assume $v \in StopCmd$, $j > i$, $c < b$, and w any command and we prove $NotChoosable(j, c, w)'$. By Lemma 1.7, it suffices to prove $NotChoosable(j, c, w)$. We split the proof into two cases.
(2)1. Case: $val2a(i, b, Q) = \top$
Proof: Assumption (1)1.3 implies $E4(i, b, Q, v)$, so the assumption $v \in StopCmd$ implies $E4b(i, b, Q, v)$. The case assumption, the assumption $j > i$, and $E4b(i, b, Q, v)$ imply $val2a(j, b, Q) = \top$. The assumption $c < b$ and step (1)4 then imply $NotChoosable(j, c, w)$.
(2)2. Case: $val2a(i, b, Q) \neq \top$
(3)1. $val2a(i, b, Q) = val2a(i, b, Q) = v$
Proof: Assumption (1)1.3 implies $E3(i, b, Q, v)$, which implies $val2a(i, b, Q) = v$. The case assumption and the definition of $val2a$ then implies $val2a(i, b, Q) = w$.
(3)2. $Done2a(i, mbal2a(i, b, Q), v)$
Proof: (3)1, assumption (1)1.4, and the definition of $val2a$ imply $vote_i[a][mbal2a(i, b, Q)] = v$ for some acceptor a in Q, which by Lemma 1.3 implies $Done2a(i, mbal2a(i, b, Q), v)$.
By the assumption $c < b$, it suffices to consider the following two cases.
(3)3. Case: $c < mbal2a(i, b, Q)$
Proof: Step (3)2 and assumption (1)1.1 imply $NoneChoosableAfter(i, mbal2a(i, b, Q), v)$. By the case assumption and the assumptions $v \in StopCmd$ and $j > i$, this implies $NotChoosable(j, c, w)$.
(3)4. Case: $mbal2a(i, b, Q) \leq c < b$
(4)1. $mbal2a(j, b, Q) < mbal2a(i, b, Q)$
Proof: The assumption $v \in StopCmd$ and (3)1 imply $val2a(j, b, Q) \in StopCmd$. Case assumption (2)2 and the definition of $val2a$ then imply $mbal2a(k, b, Q) < mbal2a(i, b, Q)$ for all $k > i$.
(4)2. $NotChoosable(j, c, w)$
Proof: (4)1 and case assumption (3)4 imply $mbal2a(j, b, Q) < c < b$. By assumption (1)1.4, Lemma 3 implies $NotChoosable(j, c, w)$. □

Theorem 1 □ *Consistency*
Proof: By definition of *Consistency*, it suffices to assume $Chosen(i, b, v)$ and $Chosen(i, c, w)$ and to prove $v = w$. Without loss of generality, we can assume $b \leq c$. We then have two cases.
1. Case: $b < c$
Proof: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and $Chosen(i, c, w)$ imply $Done2a(i, c, w)$. By Lemma 4, this implies

20


$SafeAt(i, c, w)$. The assumptions $b < c$, an $v \neq w$ then imply $NotChoosable(i, b, v)$. By Lemma 2, this contradicts the assumption $Chosen(i, b, v)$.
2. Case: $b = c$
Proof: Lemma 1.1 implies $Done2a(i, b, v) \wedge Done2a(i, c, w)$, which by Lemma 1.2 implies $b = c$. □

Theorem 2 □ *Stopping*
Proof: By definition of *Stopping*, it suffices to assume $Chosen(i, b, v)$, $Chosen(j, c, w)$, $v \in StopCmd$, and $j > i$ and to obtain a contradiction. We split the proof into two cases.
1. Case: $c < b$
Proof: $Chosen(i, b, v)$ and Lemma 1.1 imply $Done2a(i, b, v)$. This and Lemma 4 imply $NoneChoosableAfter(i, b, v)$, which by the case assumption and the assumptions $v \in StopCmd$ and $j > i$ implies $NotChoosable(j, c, w)$. The assumption $Chosen(j, c, w)$ and Lemma 2 then provide the required contradiction.
2. Case: $c \geq b$
Proof: $Chosen(j, c, w)$ and Lemma 1.1 imply $Done2a(j, c, w)$. Lemma 4 then implies $NoReconfigBefore(j, c)$. The case assumption, the assumptions $v \in StopCmd$ and $j > i$, and $NoReconfigBefore(j, c)$ imply $NotChoosable(i, b, v)$. The assumption $Chosen(i, b, v)$ and Lemma 2 then provide the required contradiction. □

A.2 The Proof of Progress.

Theorem 3 $\forall b, Q : Progress(b, Q)$
Proof: We assume $P1(b, Q)$, $P2(b, Q)$ and $P3(b)$ and we must prove that there exists a v such that either $\Diamond Chosen(i, b, v)$ or $(v \in StopCmd) \wedge \Diamond Chosen(i, b, v)$, for some $j < i$.
(1)1. $\Diamond\Box E1(b, Q)$
Proof: $P1(b, Q)$ implies that the ballot b leader eventually executes a $Phase1a(b)$ action. By $P2(b, Q)$, every acceptor a in Q eventually receives the $Phase1a$ message. Because $bal[a]$ is set to a value c only by receiving a ballot c message, assumption $P3(b)$ implies $bal[a] \leq b$. Hence, a must eventually receive the $Phase1a$ message and execute $Phase1b(a, b)$. By $P2(b, Q)$, the $Phase1b$ message it sends is eventually received by the leader.
(1)2. $\forall i, w : \Box(Done2a(i, b, w) \Rightarrow \Diamond Chosen(i, b, w))$
Proof: $Done2a(i, b, w)$ means that a $Phase2a(i, b, w)$ action has been executed sending a $(\text{"2a"}, b, w)$, message to every acceptor a. If a is in Q, then assumption $P2(b, Q)$ implies that it eventually receives that message. Assumption $P3(b)$ implies $bal[a] \leq b$, so $P1(b, Q)$ implies that every a in Q eventually executes $Phase2b(i, a, b, w)$, setting $vote_i[a][b]$ to w. Hence, eventually $Chosen(i, b, w)$ becomes true.

21

Challenge: encoding into 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

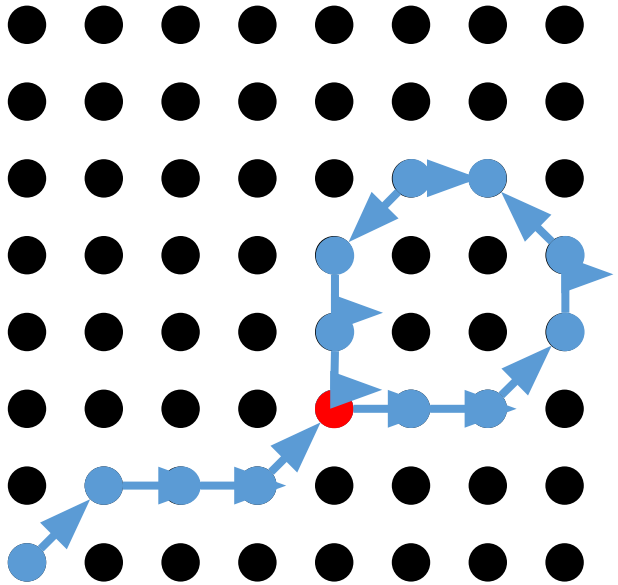
- Liveness property: “something good eventually happens”
- Often depend on fairness assumptions
- Typically proven by ranking functions, well-founded relations
  - Well beyond EPR, or not?

[POPL’18] O. P., Jochen Hoenicke, Giuliano Losa, Andreas Podelski, Mooly Sagiv, Sharon Shoham.
Reducing Liveness to Safety in First-Order Logic
[FMCAD’18] O. P., Jochen Hoenicke, Kenneth L. McMillan, Andreas Podelski, Mooly Sagiv, Sharon Shoham.
Temporal Prophecy for Proving Temporal Properties of Infinite-State Systems.

Lasso & Dynamic Abstraction

Finite State

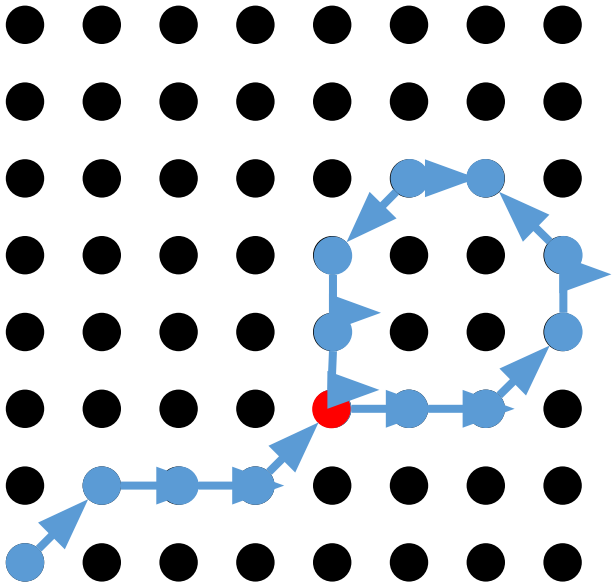
Parameterized



Liveness ⇔ No Lasso

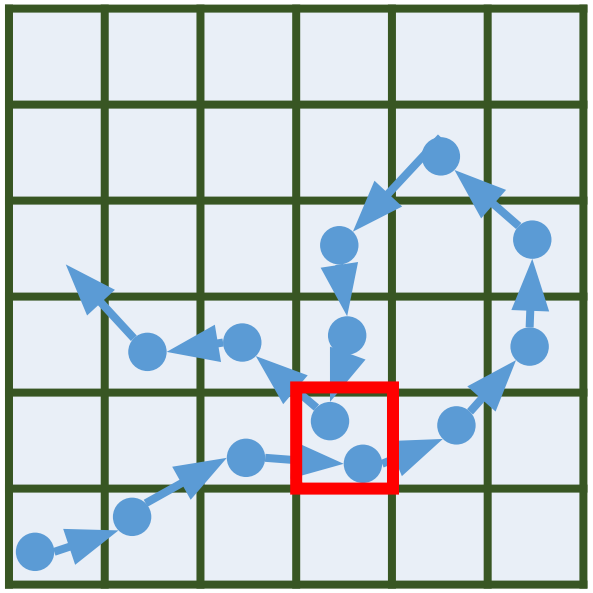
Lasso & Dynamic Abstraction

Finite State
Parameterized



Liveness ⇔ No Lasso

Infinite State
Finite Abstraction

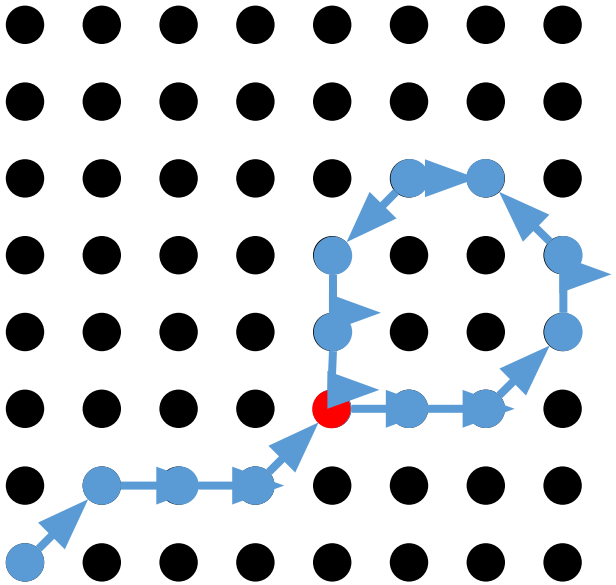


Liveness ⇐ No Lasso
Problem: Spurious Lasso

Lasso & Dynamic Abstraction

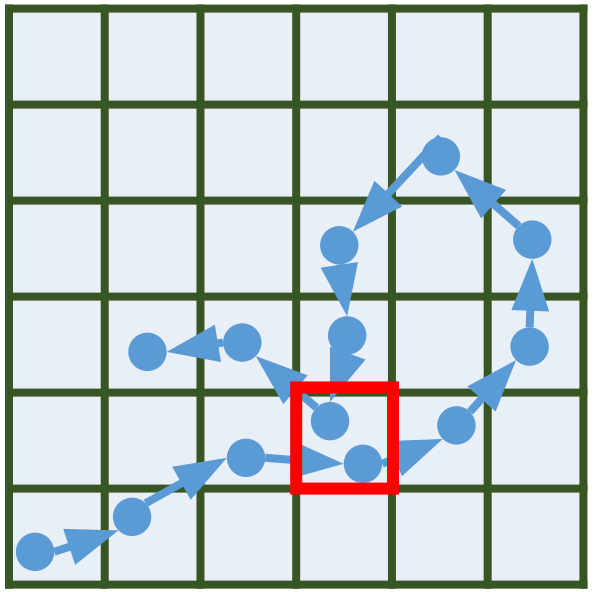
All expressible in EPR

Finite State
Parameterized



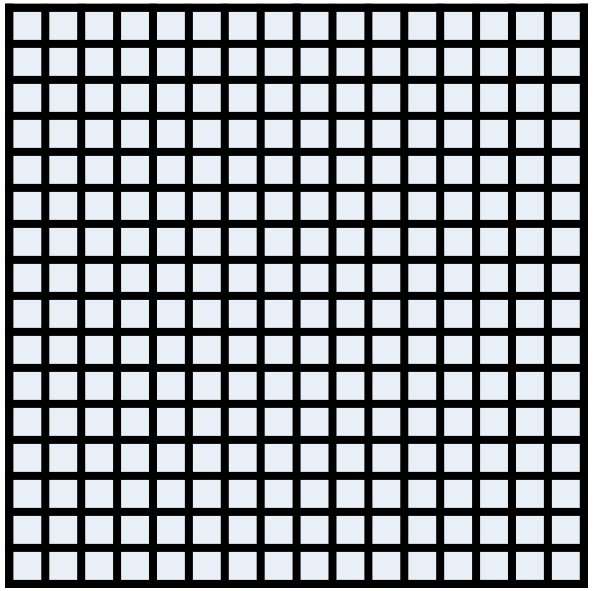
Liveness ⇔ No Lasso

Infinite State
Finite Abstraction



Liveness ⇐ No Lasso
Problem: Spurious Lasso

Dynamic Abstraction

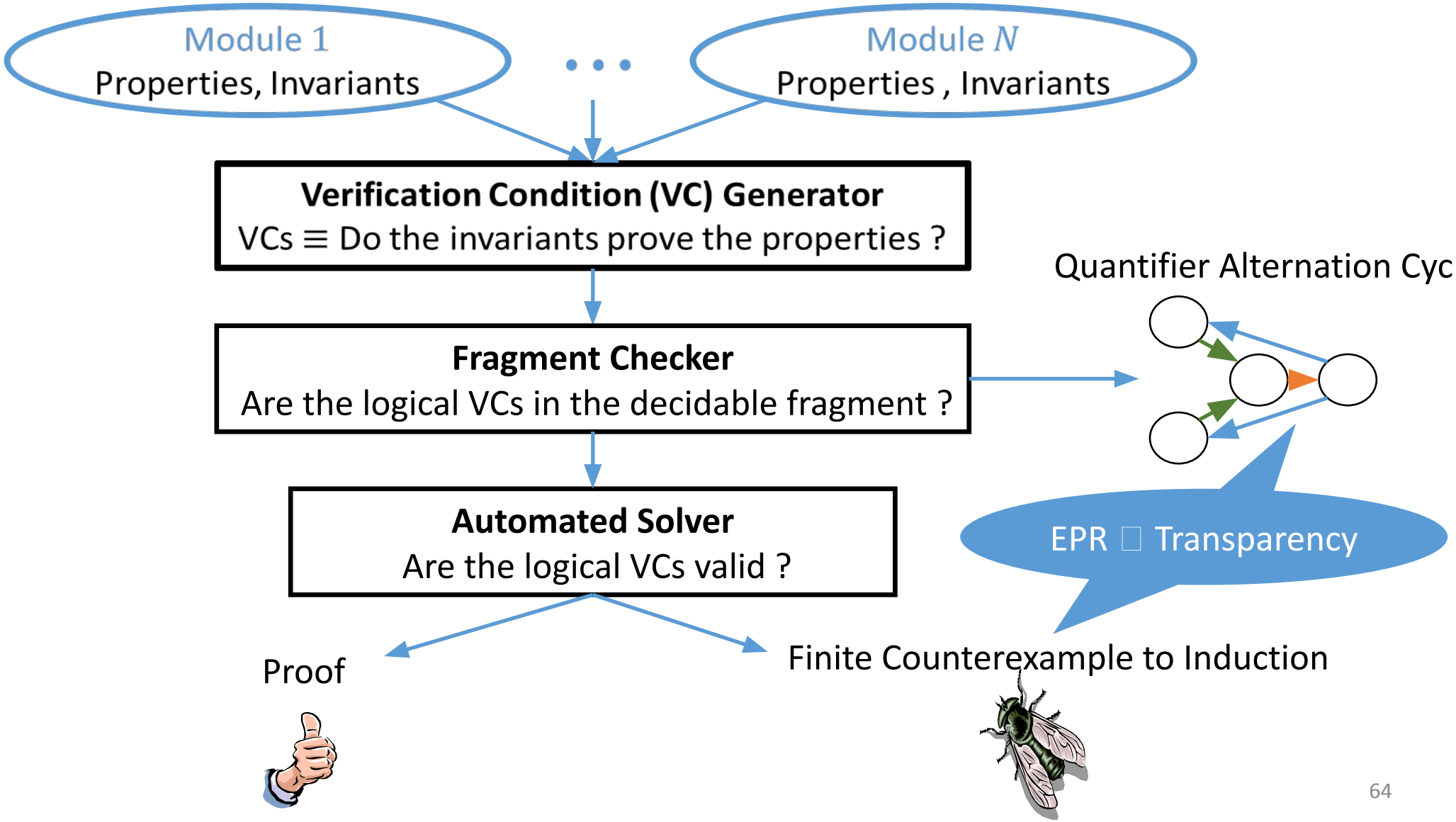


Liveness ⇐ No Lasso
Fewer Spurious Lassos

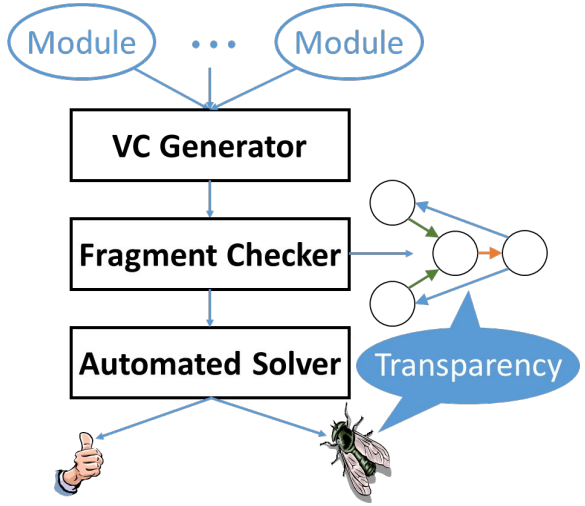
Challenge: encoding into EPR

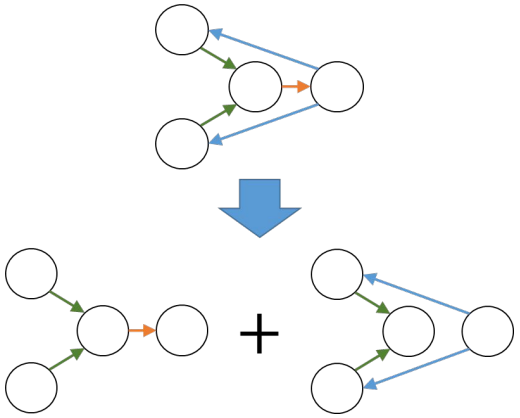
- But what can you possibly express in such a restricted logic?
  - Transitive closure over deterministic paths
  - Set cardinalities
  - Avoiding quantifier alternations
  - Encoding liveness and LTL

Logic-based deductive verification in Ivy

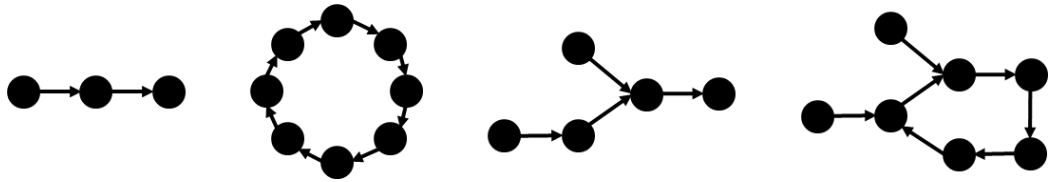


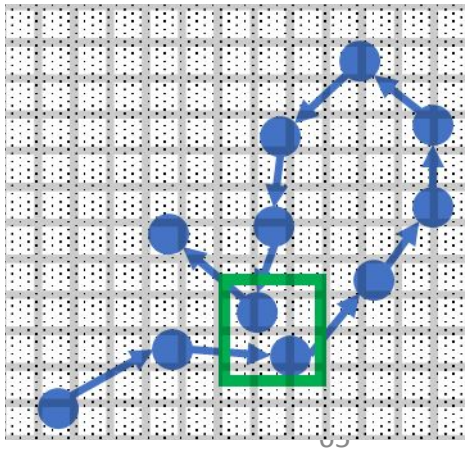
Part 1: Conclusion





- Verification in EPR
  - Deduction is decidable
  - Finite counterexamples
  - Transparent failures
- Powerful encodings
  - Transitive closure for deterministic paths
  - Set cardinalities
  - Modular decomposition to battle ∀∃ cycles
  - Liveness and temporal properties





Part 1: Open Questions

- What cannot be proven with EPR?
  - Proof theory
- Why are solvers stable on EPR?
  - Can it be generalized?
  - Part of why DPLL/CDCL works in practice
- Can it be used beyond a small group of fans?
  - Transitive closure ☺, LTL 😐, Cardinalities 😐, Breaking ∀∃ cycles ☹
- Can you get the benefit of EPR without the costs?
  - Bounded quantifier instantiations
  - Special case of a more general “simple proofs” principle?

Verification of Distributed Protocols: Decidable Modeling and Invariant Inference

Part 1: Decidable Modeling

- The Ivy deductive verification system
- The many-sorted EPR fragment
- Main challenge: expressing interesting systems and properties in EPR
  - Expressing transitive closure
  - Expressing sets and cardinalities
  - Liveness and temporal verification

Part 2: Invariant Inference

- Problem setting

Problem setting

Several Ivy papers provide:

- A collection of examples with manually written invariants
- Invariant checking is decidable, typically ~1s, complex protocols ~10s
- Counterexamples are finite
- Invariants aren't too large or too complex

Protocol	Model [LOC]	Invariant [Conjectures]	EPR [sec] μ	EPR [sec] σ
Paxos	85	11	2.2	0.1
Multi-Paxos	98	12	2.6	0.1
Vertical Paxos	123	18	2.2	0.2
Fast Paxos	117	17	6.2	1.6
Flexible Paxos	88	11	2.2	0
Stoppable Paxos	132	16	5.4	0.9

```
# safety property
invariant decision(N1,R1,V1) & decision(N2,R2,V2) -> V1 = V2
# proposals are unique per round
invariant proposal(R,V1) & proposal(R,V2) -> V1 = V2
# only vote for proposed values
invariant vote(N,R,V) -> proposal(R,V)
# decisions come from quorums of votes:
invariant forall R, V. (exists N. decision(N,R,V)) -> exists Q. forall N. member(N, Q) -> vote(N,R,V)
# properties of one_b_max_vote
invariant one_b_max_vote(N,R2,none,V1) & ~le(R2,R1) -> ~vote(N,R1,V2)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none -> ~le(R,RM) & vote(N,RM,V)
invariant one_b_max_vote(N,R,RM,V) & RM ~= none & ~le(R,RO) & ~le(RO,RM) -> ~vote(N,RO,VO)
# property of choosable and proposal
invariant ~le(R2,R1) & proposal(R2,V2) & V1 ~= V2 -> exists N. member(N,Q) & left_rnd(N,R1) & ~vote(N,R1,V1)
# property of one_b, left_rnd
invariant one_b(N,R2) & ~le(R2,R1) -> left_rnd(N,R1)
```

UPDR [CAV'15, JACM'17] Aleksandr Karbyshev, Nikolaj Bjorner, Shachar Itzhaky, Noam Rinetzky and Sharon Shoham.
Property-Directed Inference of Universal Invariants or Proving Their Absence.

I4 [SOSP'19] Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem Sakallah.
I4: incremental inference of inductive invariants for verification of distributed protocols.

FOL-IC3 [PLDI'20] Jason Koenig, O. P., Neil Immerman, and Alex Aiken.
First-Order Quantified Separators.

SWISS [NSDI'21] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno.
Finding Invariants of Distributed Systems: It's a Small (Enough) World After All.

DistAI [OSDI'21] Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan.
DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols.

IC3PO [NSF'21] Aman Goel and Karem Sakallah.
On Symmetry and Quantification: A New Approach to Verify Distributed Protocols.

FOL-IC3 [TACAS'22] Jason Koenig, O. P., Sharon Shoham, and Alex Aiken.
Inferring Invariants with Quantifier Alternations: Taming the Search Space Explosion.

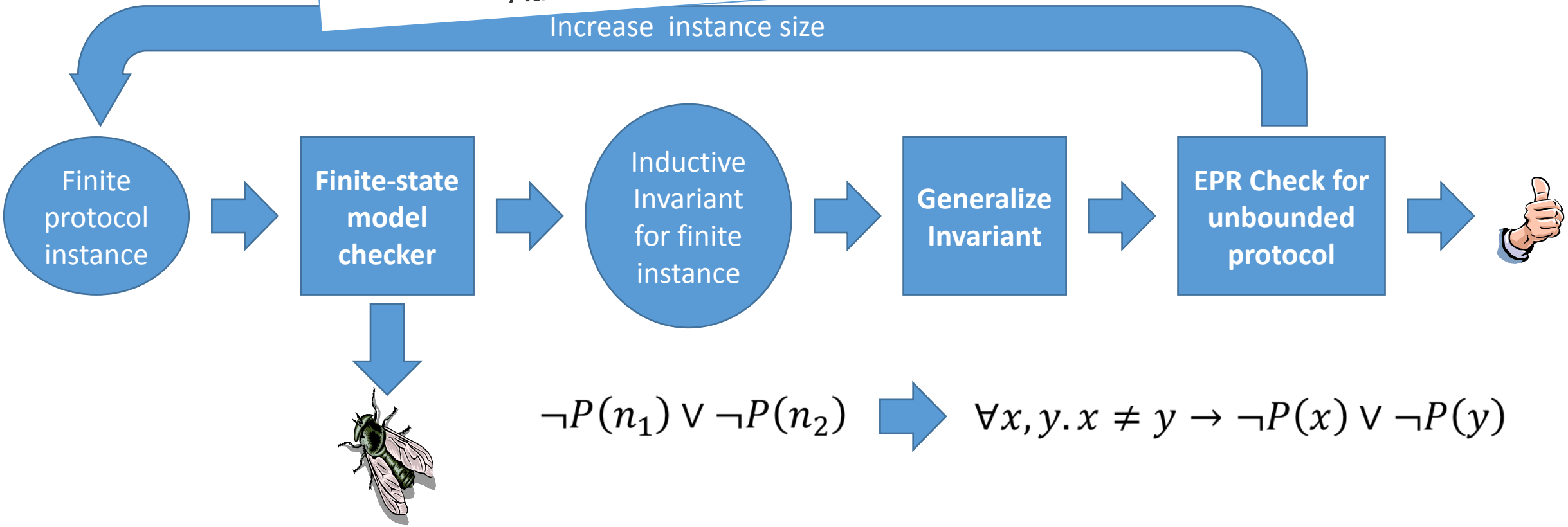
DuoAI [OSDI'22] Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh.
DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols.

Interesting ideas

- Generalize from finite instances
  - I4
  - IC3PO
- Explicitly enumerate candidate invariants
  - SWISS
  - DistAI
  - DuoAI
- Generalize IC3/PDR by adapting lemma generation
  - UPDR
  - FOL-IC3

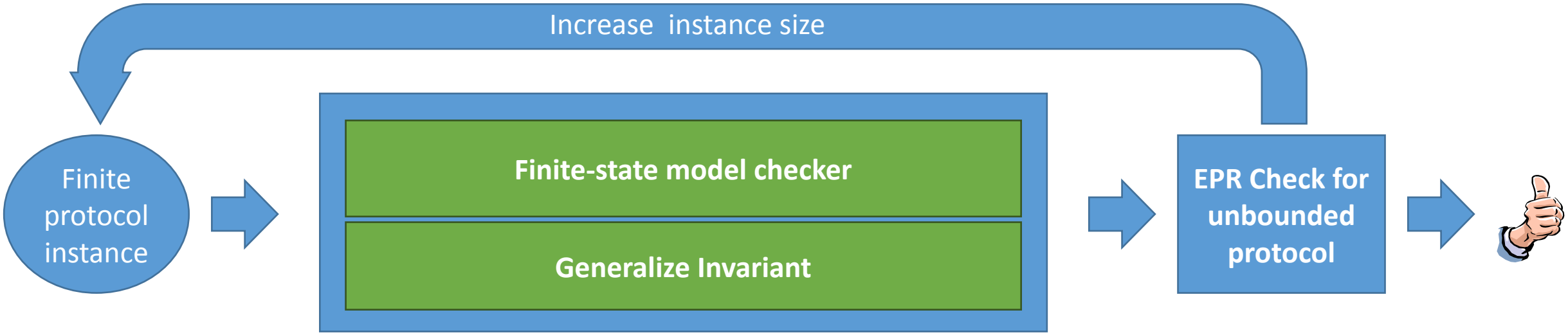
Generalize from finite instances (I4)

[TACAS'01] Amir Pnueli, Sitvanit Ruah, and Lenore Zuck.
Automatic deductive verification with invisible invariants.



[SOSP'19] Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem Sakallah.
I4: incremental inference of inductive invariants for verification of distributed protocols.

Generalize from finite instances (IC3PO)



$$\neg P(n_1) \vee \neg P(n_2) \Rightarrow \forall x, y.\, x \neq y \rightarrow \neg P(x) \vee \neg P(y) \Rightarrow \begin{array}{l} \left(\neg P(n_1) \vee \neg P(n_2)\right) \wedge \\ \left(\neg P(n_1) \vee \neg P(n_3)\right) \wedge \\ \left(\neg P(n_2) \vee \neg P(n_3)\right) \end{array}$$

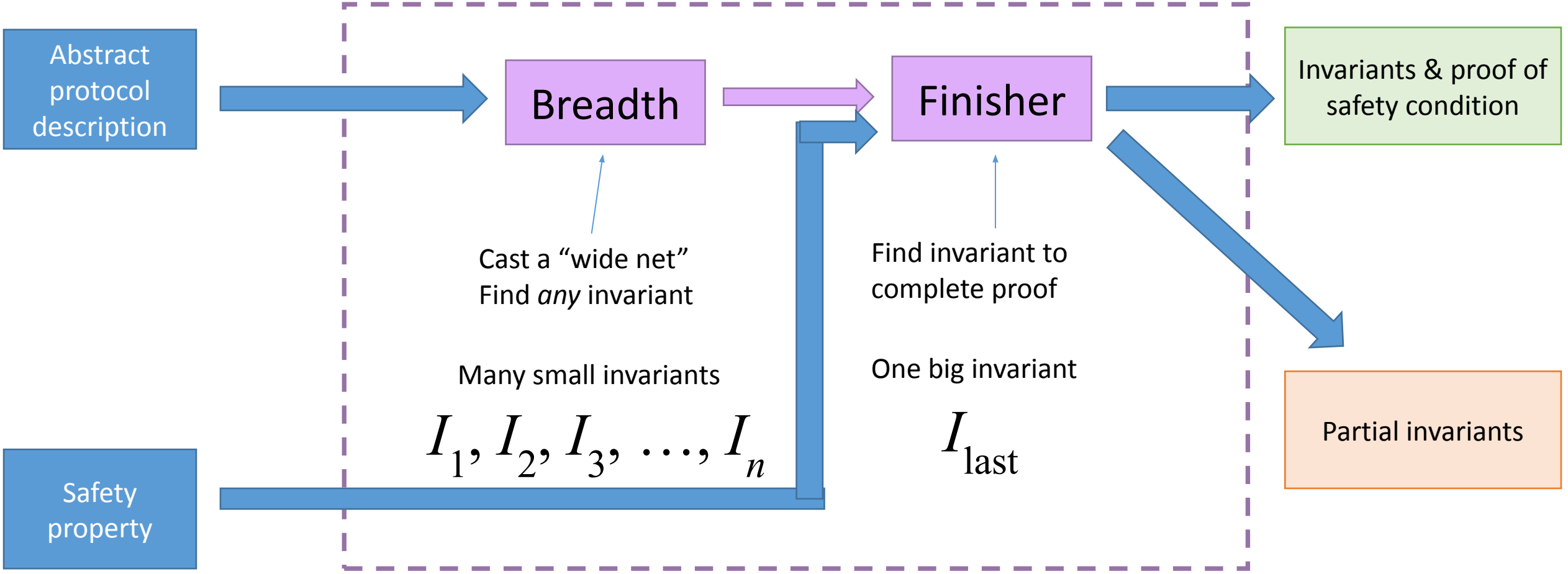
[NSF'21] Aman Goel and Karem Sakallah.
On Symmetry and Quantification: A New Approach to Verify Distributed Protocols.

Explicitly enumerate invariants (SWISS)

- Typically $Inv = p_1 \wedge \cdots \wedge p_n$
- Each p_i isn't too complex, maybe we can explicitly enumerate all of them? Would that be useful?
- If we find some p that is inductive, we can learn it as an invariant
  - May help make additional p's inductive
- Ultimately, we want to prove the safety property

[NSDI'21] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno.
Finding Invariants of Distributed Systems: It's a Small (Enough) World After All.

Explicitly enumerate invariants (SWISS)



[NSDI'21] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno.
Finding Invariants of Distributed Systems: It's a Small (Enough) World After All.

Explicitly enumerate invariants (SWISS)

Exploring the space of candidate invariant predicates for Paxos

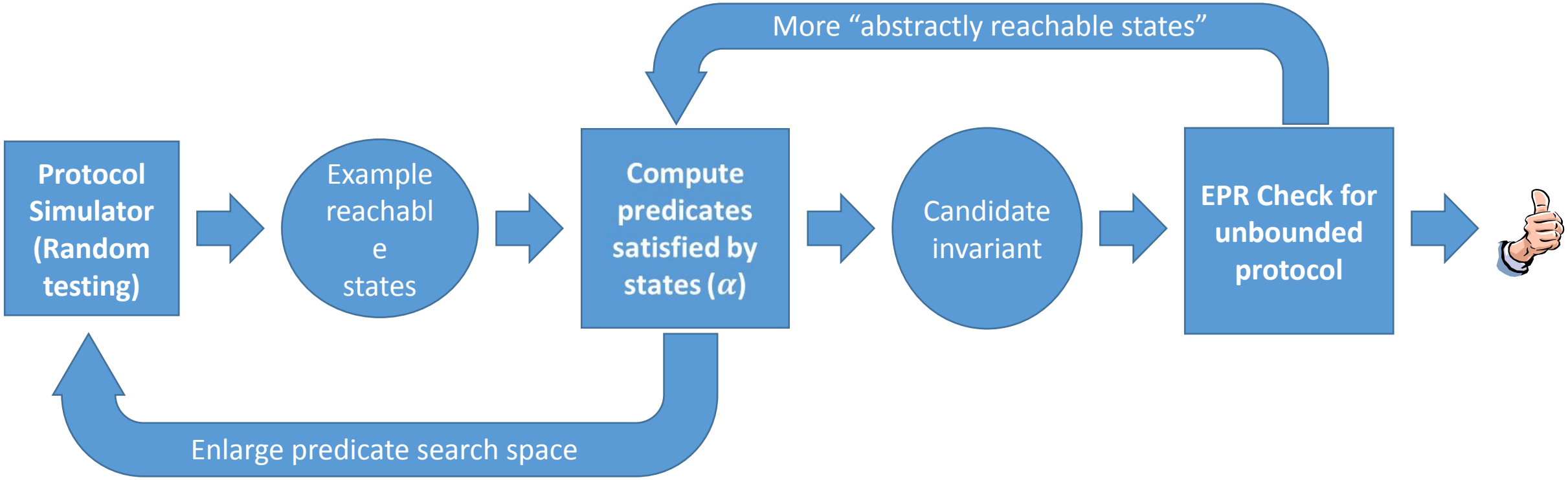
	Candidate invariant space	Number of candidate invariants	Symmetries	Counter-example filters	Removing redundant invariants	Invariant predicates
Finisher	6 terms	~ 99,000,000,000,000	~ 200,000,000,000	155	155	5
Breadth	3 terms	~ 820,000,000	~ 3,000,000	~ 900,000	2,250	801

100 ms on average
Brute force is not feasible

Counterexample-guided synthesis:
When one predicate fails to be inductive, use it to narrow your search space.

[NSDI'21] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno.
Finding Invariants of Distributed Systems: It's a Small (Enough) World After All.

Explicitly enumerate invariants (DistAI, DuoAI)



[OSDI’21] Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan.
DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols.

[OSDI’22] Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh.
DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols.

Adapt IC3/PDR (UPDR, FOL-IC3)

- Core task for lemma learning in IC3/PDR:
  - Given $F = p_1 \wedge \cdots \wedge p_n$ and state s, find p s.t. $s \not\models p$ and $\mathrm{post}([F] \cap [p]) \subseteq [p]$
- UPDR: for p a forall-only formula, the diagram of s can be used
- FOL-IC3:
  - Closely related problem is separation: given $s_1^+, \ldots, s_n^+$ and $s_1^-, \ldots, s_m^-$ find p such that $s_1^+, \ldots, s_n^+ \models p$ and $s_1^-, \ldots, s_m^- \models \neg p$
  - For forall-exists formulas, separation is NP-complete
  - Use a SAT solver!
  - Many more techniques to scale to complex protocols

[CAV'15, JACM'17] Aleksandr Karbyshev, Nikolaj Bjorner, Shachar Itzhaky, Noam Rinetzky and Sharon Shoham.
Property-Directed Inference of Universal Invariants or Proving Their Absence.

[PLDI'20] Jason Koenig, O. P., Neil Immerman, and Alex Aiken.
First-Order Quantified Separators.

[TACAS'22] Jason Koenig, O. P., Sharon Shoham, and Alex Aiken.
Inferring Invariants with Quantifier Alternations: Taming the Search Space Explosion.

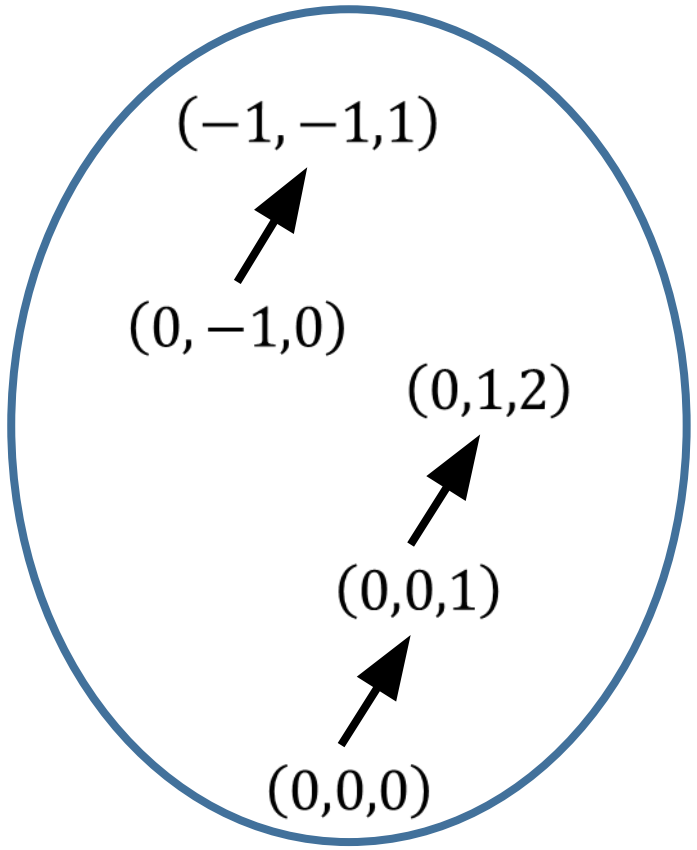
Induction Duality and Primal-Dual Houdini

- Induction Duality
  - Formal symmetric connection between execution traces and incremental induction proofs
- Primal-Dual Houdini

  - Houdini + Dual Houdini = Primal-dual invariant inference
  - Interesting theoretical properties
  - Promising empirical results for distributed protocols

[POPL'22] O. P., James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. Induction Duality: Primal-Dual Search for Invariants.

Execution vs Incremental Induction

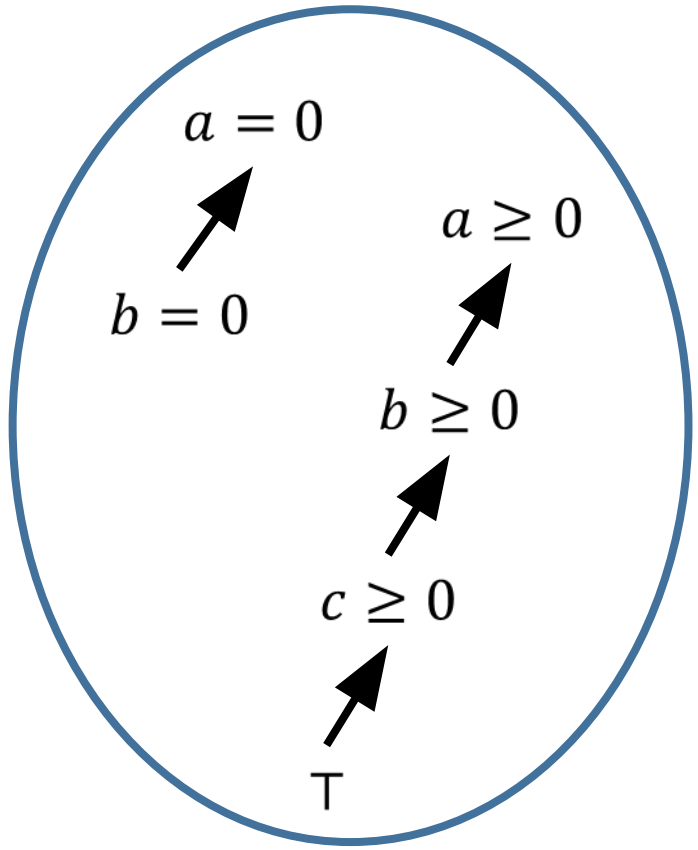


States and Transitions

```
a, b, c := 0, 0, 0
while * {
   assert a ≥ 0
   a, b, c := a+b, b+c, c+1
}
```

Loop invariant:

$a \geq 0 \wedge b \geq 0 \wedge c \geq 0$

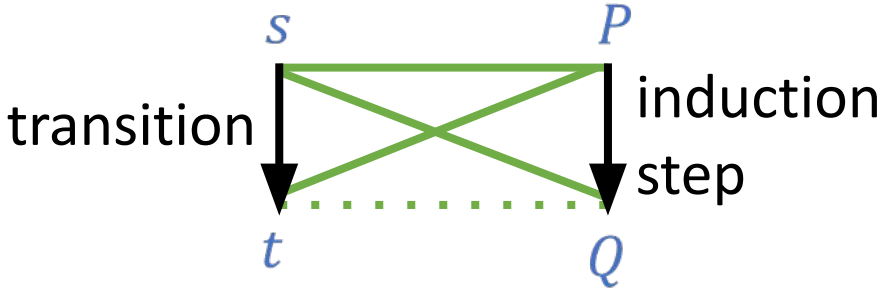


Predicates and Incremental Induction

Formalizing symmetric incremental induction

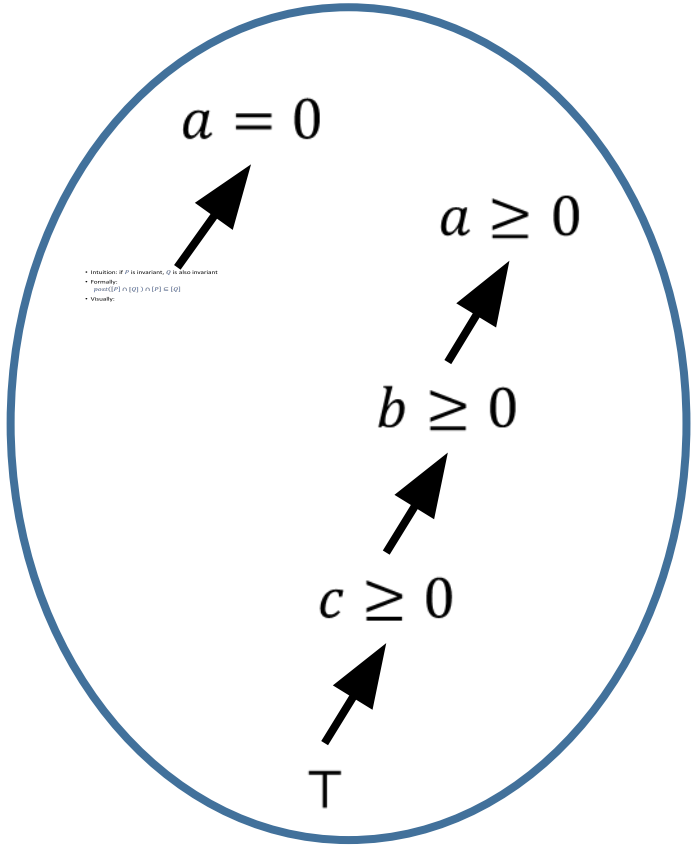
- Intuition: if P is invariant, Q is also invariant
- Formally:

  $post([P] \cap [Q]) \cap [P] \subseteq [Q] \;\Leftrightarrow\; P \wedge Q \vDash wp(P \rightarrow Q)$
- Visually:



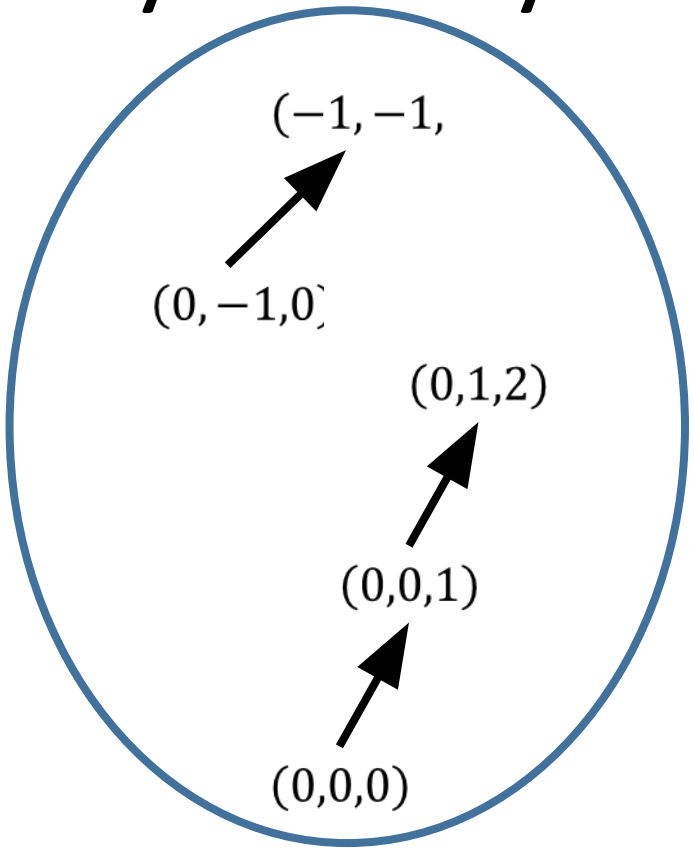
- Contrast with IC3/PDR relative inductiveness:

  $post([P] \cap [Q]) \subseteq [Q]$

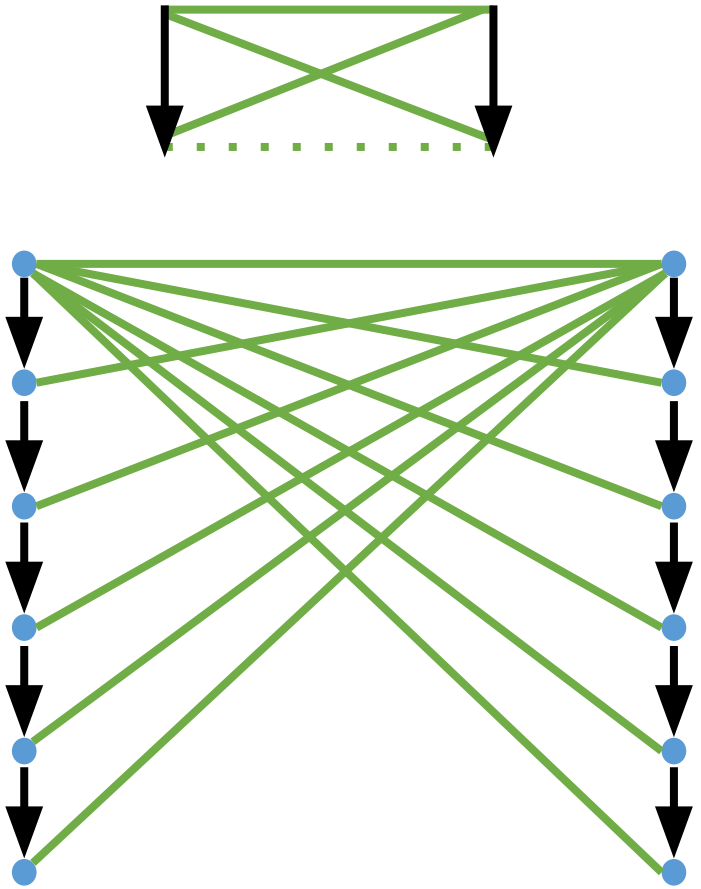


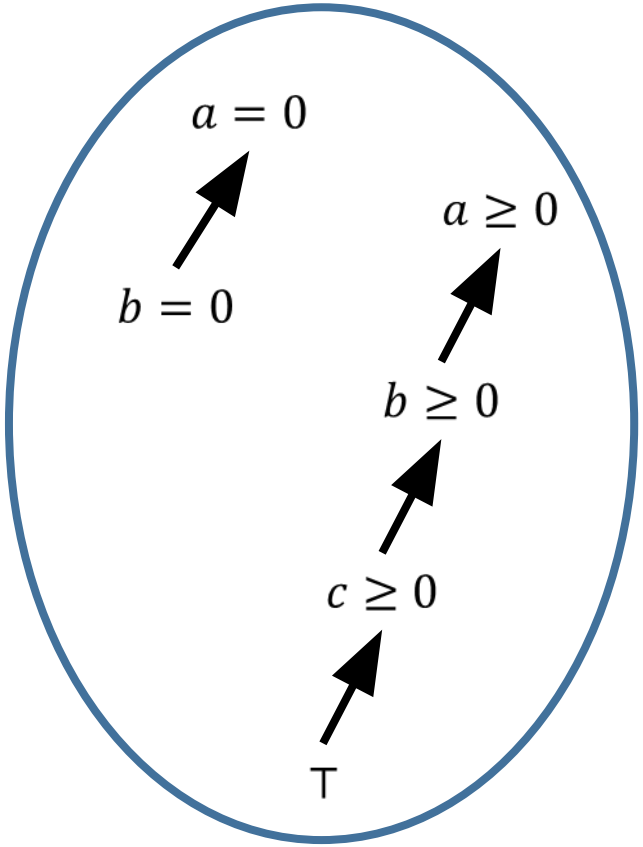
Predicates and Incremental Induction

Executions vs Incremental Induction: Symmetry



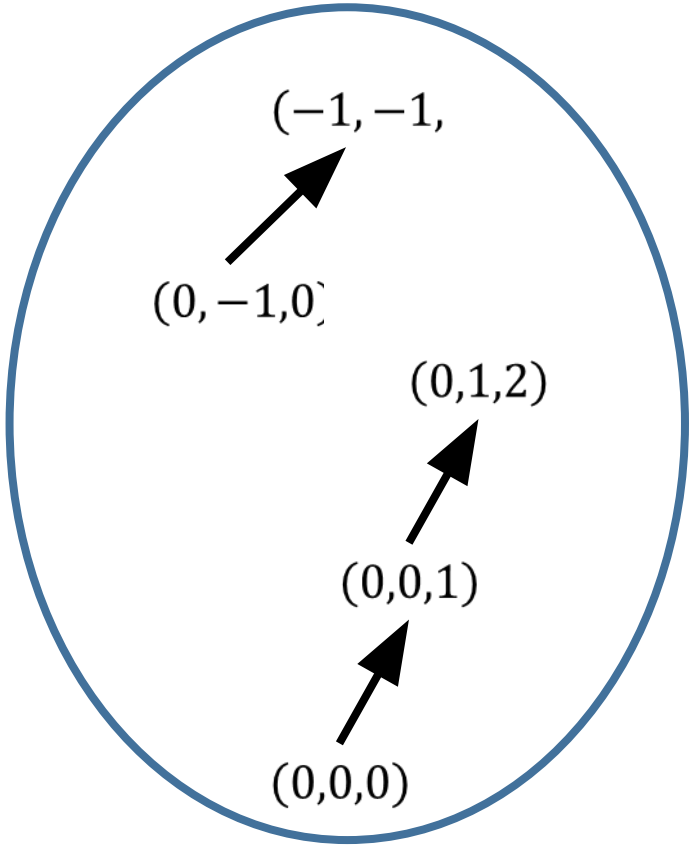
States and Transitions



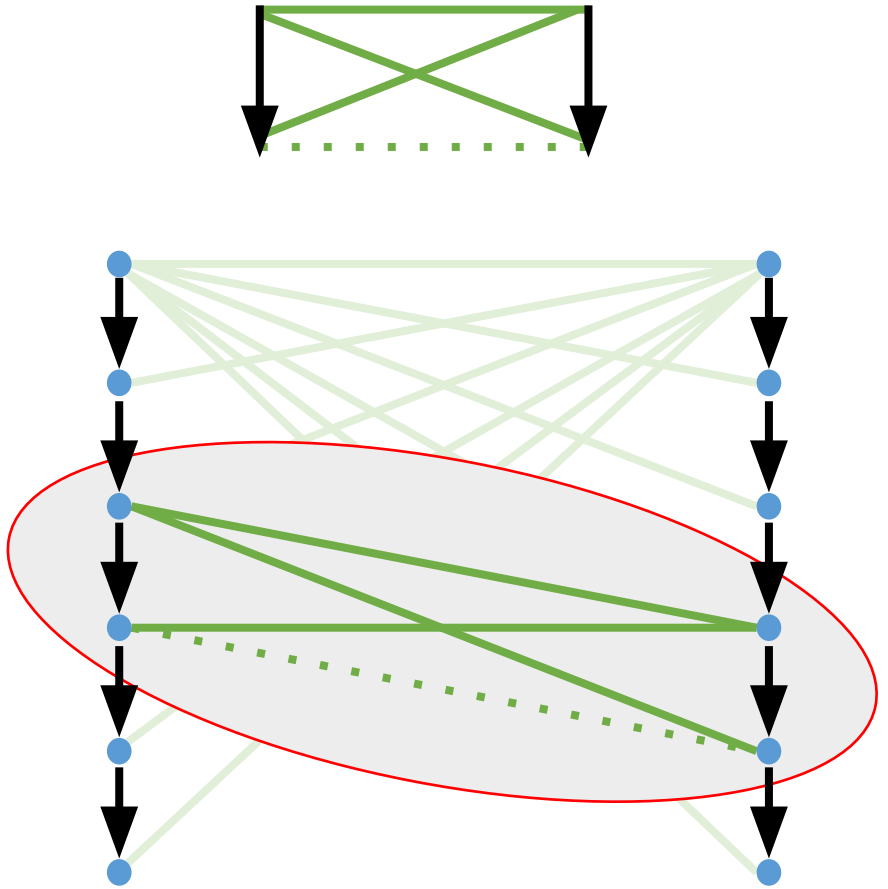


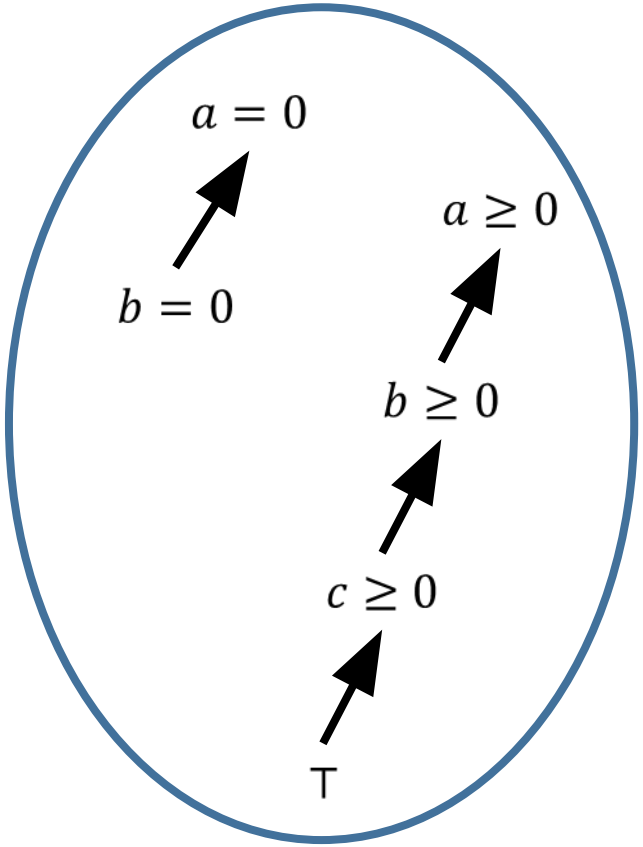
Predicates and
Incremental Induction Steps

Executions vs Incremental Induction: Symmetry



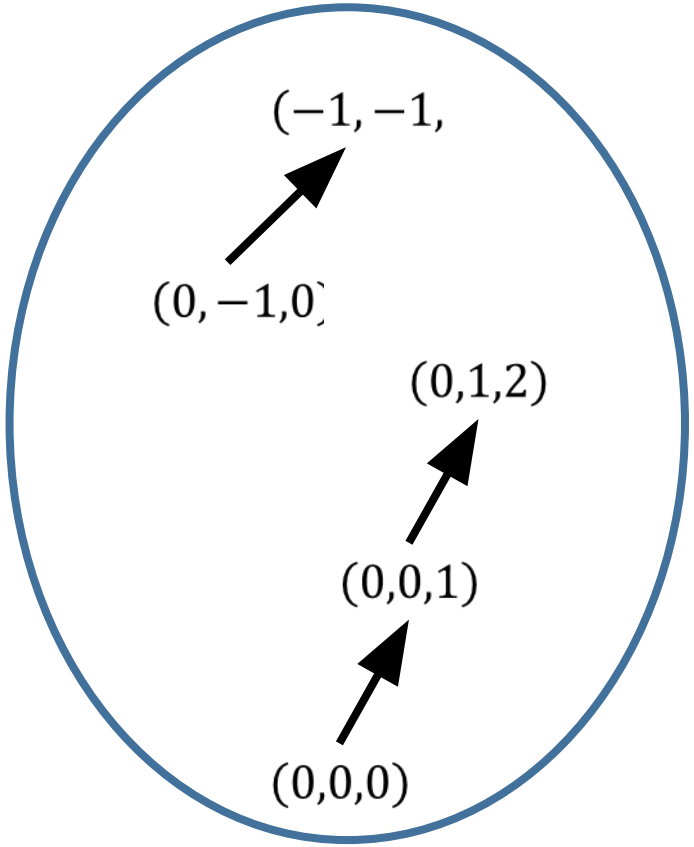
States and Transitions



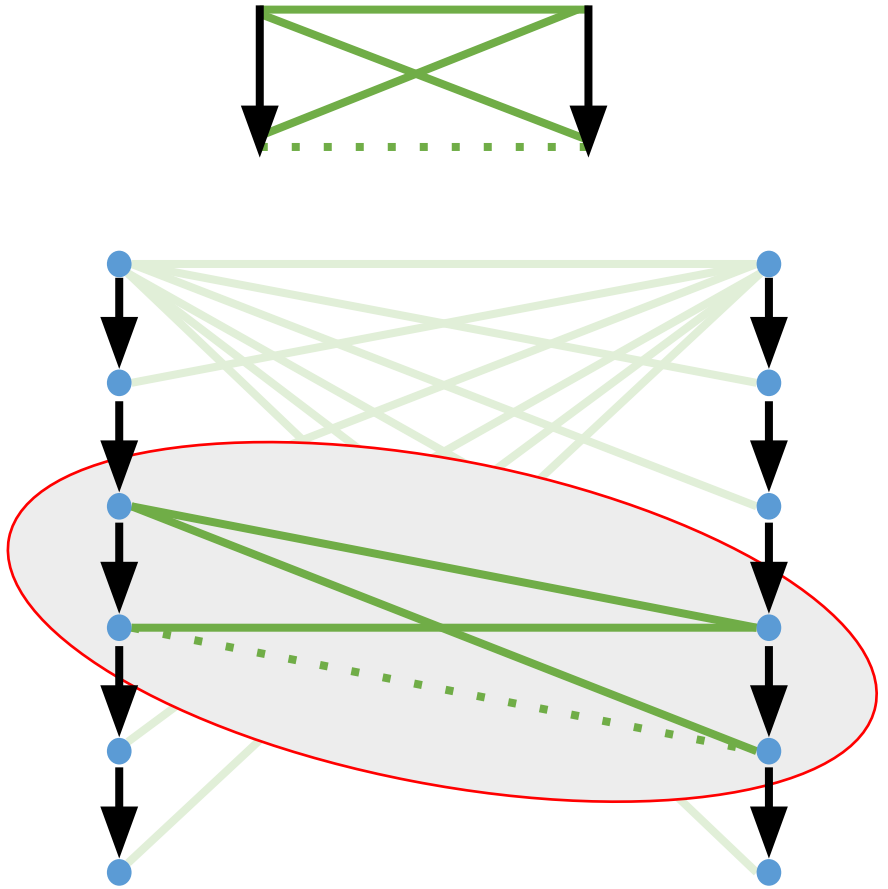


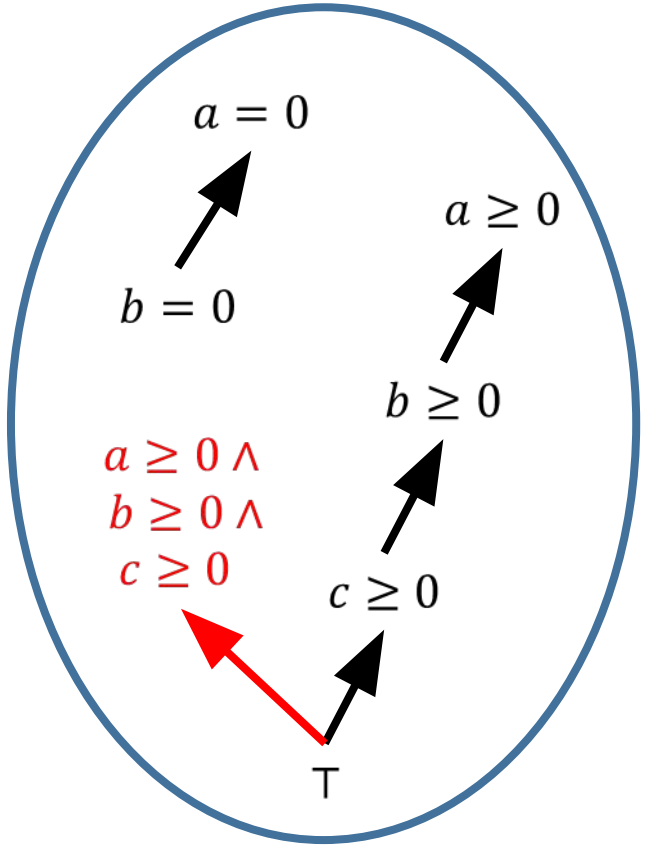
Predicates and
Incremental Induction Steps

Executions vs Incremental Induction: Symmetry



States and Transitions





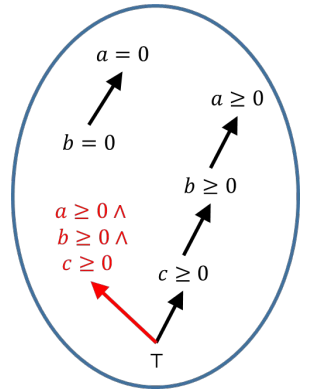
Predicates and
Incremental Induction Steps

Induction Duality Structure

V: finite sets of states

Induction Duality

V': finite sets of predicates

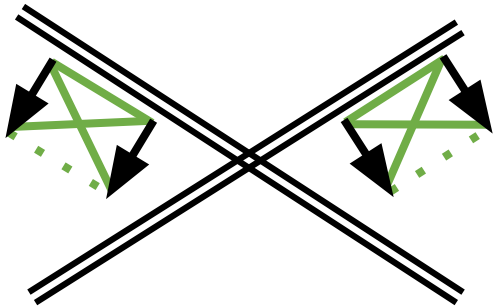


Induction over bounded proofs

$E_\omega = \{(S, S \cup T) \in V \times V \mid \cdots\}$

Dual-Inductive

$\ni$



Induction over executions

$E'_\omega = \{(P, P \cup Q) \in V' \times V' \mid \cdots\}$

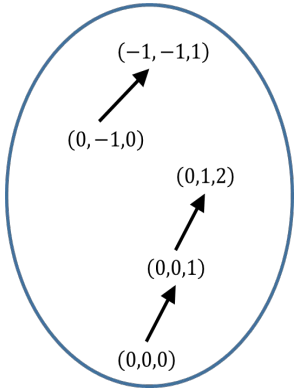
Inductive

$\ni$

Transitions

$E = \{(S, S \cup \{t\}) \in V \times V \mid t \in post(S)\}$

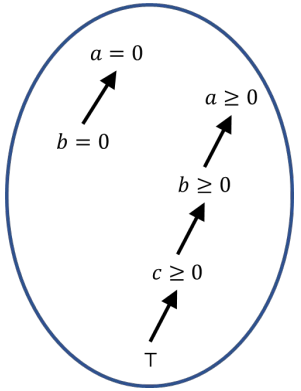
Reachable



Bounded incremental induction

$E' = \{(P, P \cup Q) \in E'_\omega \mid |Q| \leq k\}$

k-provable



Induction Duality

Induction over bounded proofs

$E_\omega = \{(S, S \cup T) \in V \times V \mid \cdots\}$

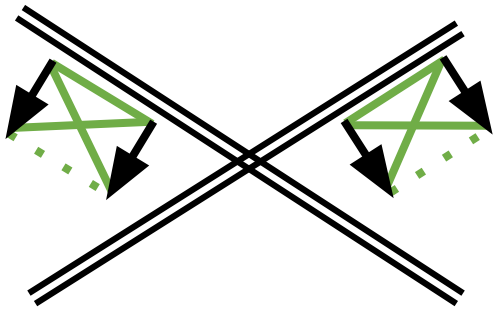
Dual-Inductive

$\supseteq$

Transitions

$E = \{(S, S \cup \{t\}) \in V \times V \mid t \in post(S)\}$

Reachable



Induction over executions

$E'_\omega = \{(P, P \cup Q) \in V' \times V' \mid \cdots\}$

Inductive

$\supseteq$

Bounded incremental induction

$E' = \{(P, P \cup Q) \in E'_\omega \mid |Q| \leq k\}$

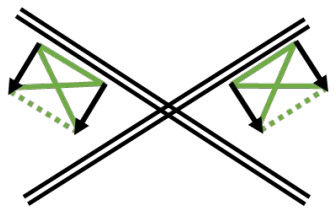
k-provable

Induction over bounded proofs

$E_\omega = \{(S, S \cup T) \in V \times V \mid \cdots\}$

Dual-Inductive

UI



Induction over executions

$E'_\omega = \{(P, P \cup Q) \in V' \times V' \mid \cdots\}$

Inductive

UI

Transitions

$E = \{(S, S \cup \{t\}) \in V \times V \mid t \in post(S)\}$

Reachable

Bounded incremental induction

$E' = \{(P, P \cup Q) \in E'_\omega \mid |Q| \leq k\}$

k-provable

Concept	Induction dual	γ ⇄ α dual
	k-provable	invariant
	reachable	k-abstractly-reachable
	k-abstractly-reachable	reachable
	Invariant	k-provable
	dual-inductive	
	inductive	

Induction over bounded proofs

$E_\omega = \{(S, S \cup T) \in V \times V \mid \cdots\}$

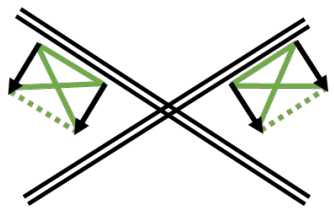
Dual-Inductive

UI

Transitions

$E = \{(S, S \cup \{t\}) \in V \times V \mid t \in post(S)\}$

Reachable



Induction over executions

$E'_\omega = \{(P, P \cup Q) \in V' \times V' \mid \cdots\}$

Inductive

UI

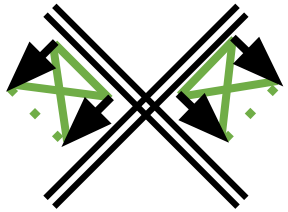
Bounded incremental induction

$E' = \{(P, P \cup Q) \in E'_\omega \mid |Q| \leq k\}$

k-provable

Concept	**Induction dual**	$\gamma \rightleftarrows \alpha$ **dual**
reachable: E^*	k-provable	invariant
k-provable: E'^*	reachable	k-abstractly-reachable
invariant: $\alpha(E^*)$	k-abstractly-reachable	reachable
k-abstractly-reachable: $\gamma(E'^*)$	invariant	k-provable
inductive: E'_ω	dual-inductive	⊇reachable
dual-inductive: E_ω	inductive	⊇k-provable

Houdini and Dual Houdini



Induction Duality

- Houdini

Input: P – set of predicates

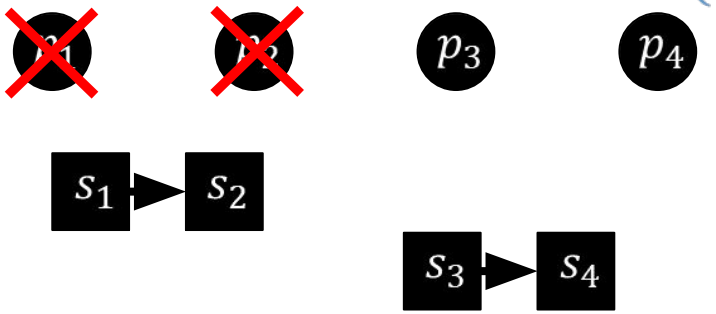
$P_H := P$

while $\exists (S, S \cup \{t\}) \in E. S \vDash P_H,\ \ t \nvDash P_H$:

$P_H := \{p \in P_H \mid t \vDash p\}$

CTI

$(\emptyset, P_H) \in E'_\omega$



Dual-Houdini

Input: S – set of states

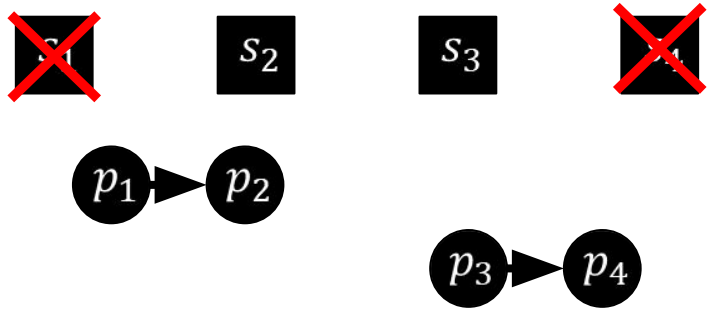
$S_H := S$

while $\exists (P, P \cup Q) \in E'.\ S_H \vDash P,\ S_H \nvDash Q$:

$S_H := \{s \in S_H \mid s \vDash Q\}$

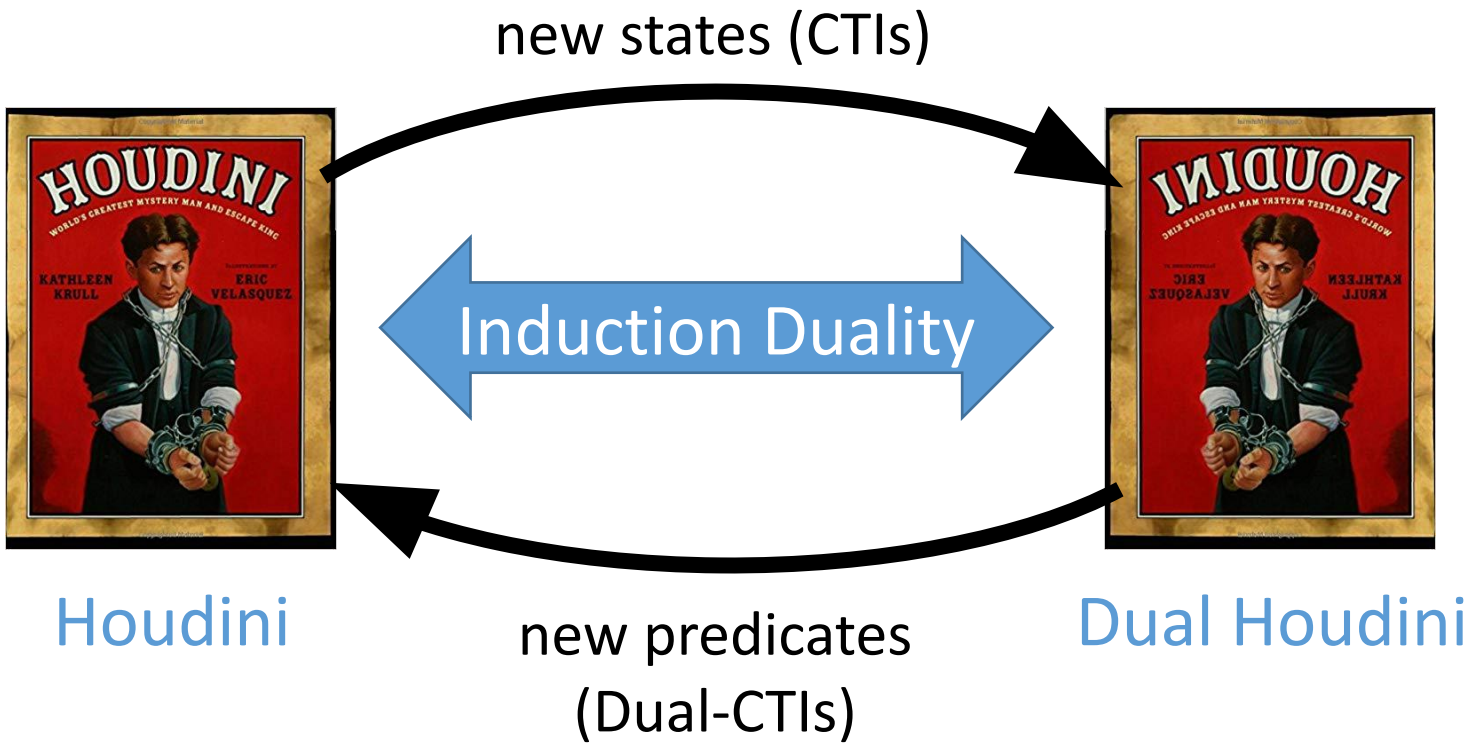
Dual-CTI

$(\emptyset, S_H) \in E_\omega$



Theorem: all the states in S_H cannot be ruled out by k-bounded incremental induction (no matter how long)

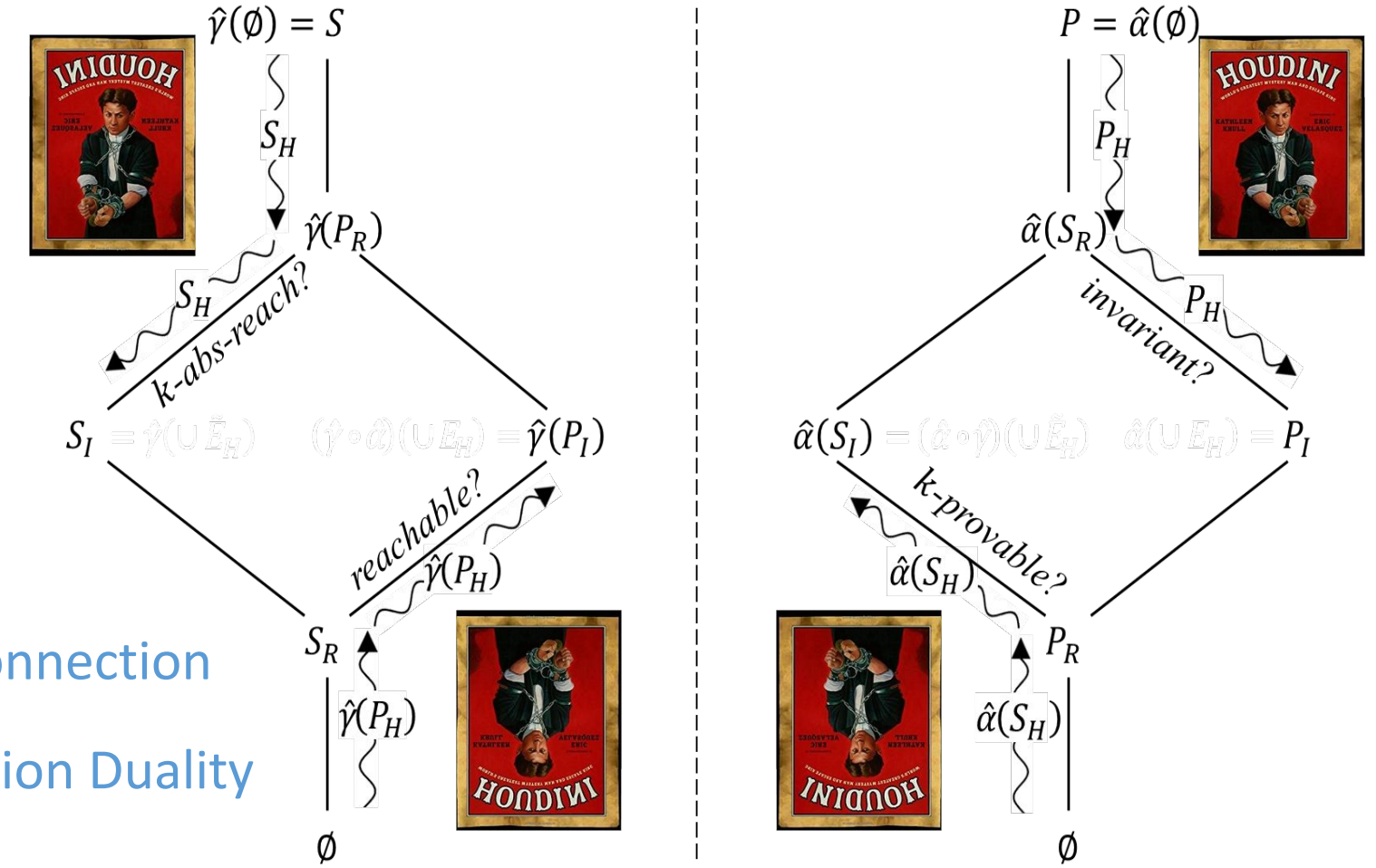
Primal-Dual Houdini



Galois Connection and Induction Duality

Lattice of sets of states

Lattice of sets of predicates



Vertical flip: Galois Connection

Horizontal flip: Induction Duality

Example: Toy Consensus Protocol

- $\boldsymbol{v}\ (Node, Value)$ – votes
- $\boldsymbol{b}(Node)$ – bit used by a node to remember if it voted already
- $\boldsymbol{d}(Value)$ – decisions
- $\boldsymbol{m}(Node, Quorum)$ – membership

axiom $\forall q_1, q_2 : Quorum.\ \exists n : Node.\ \boldsymbol{m}(n, q_1) \wedge \boldsymbol{m}(n, q_2)$

```
action vote(n: Node,  x:value)
  requires ¬b(n)
  v(n,x) := true
  b(n) := true
```

```
action decide(x:value)
  requires ∃q∀n. m(n,q) → v(n,x)
  d(x) := true
```

Safety specification:

$$\forall x, y.\ \boldsymbol{d}(x) \wedge \boldsymbol{d}(y) \rightarrow x = y$$

Inductive invariant:

$$\forall x.\ \boldsymbol{d}(x) \rightarrow \exists q\ \forall n.\ \boldsymbol{m}(n, q) \rightarrow \boldsymbol{v}(n, x)$$

$$\forall n, x, y.\ \boldsymbol{v}(n, x) \wedge \boldsymbol{v}(n, y) \rightarrow x = y$$

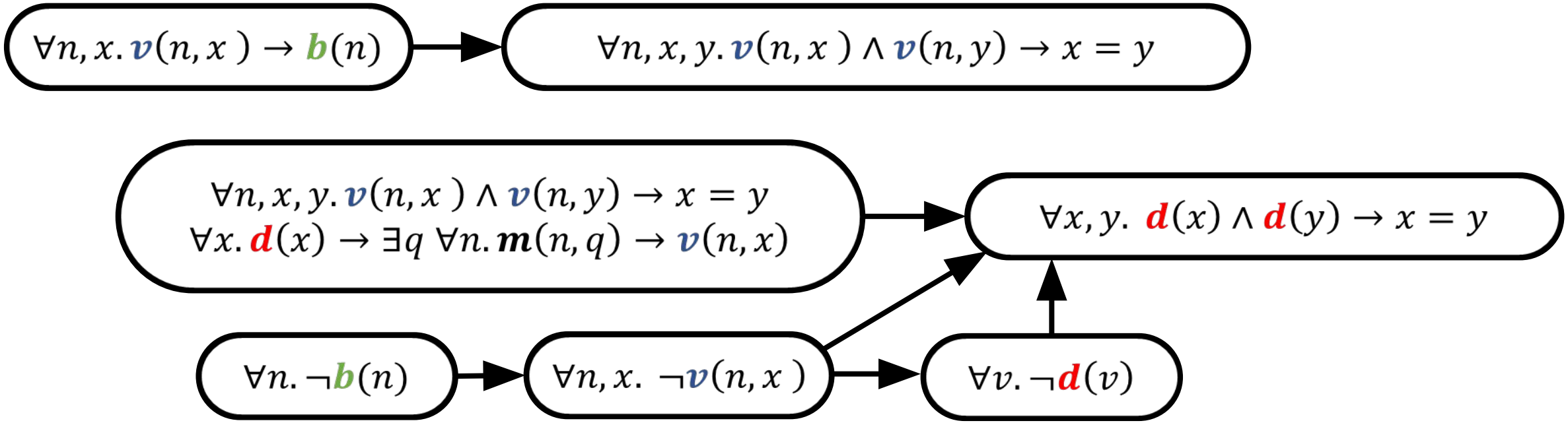
$$\forall n, x.\ \boldsymbol{v}(n, x) \rightarrow \boldsymbol{b}(n)$$

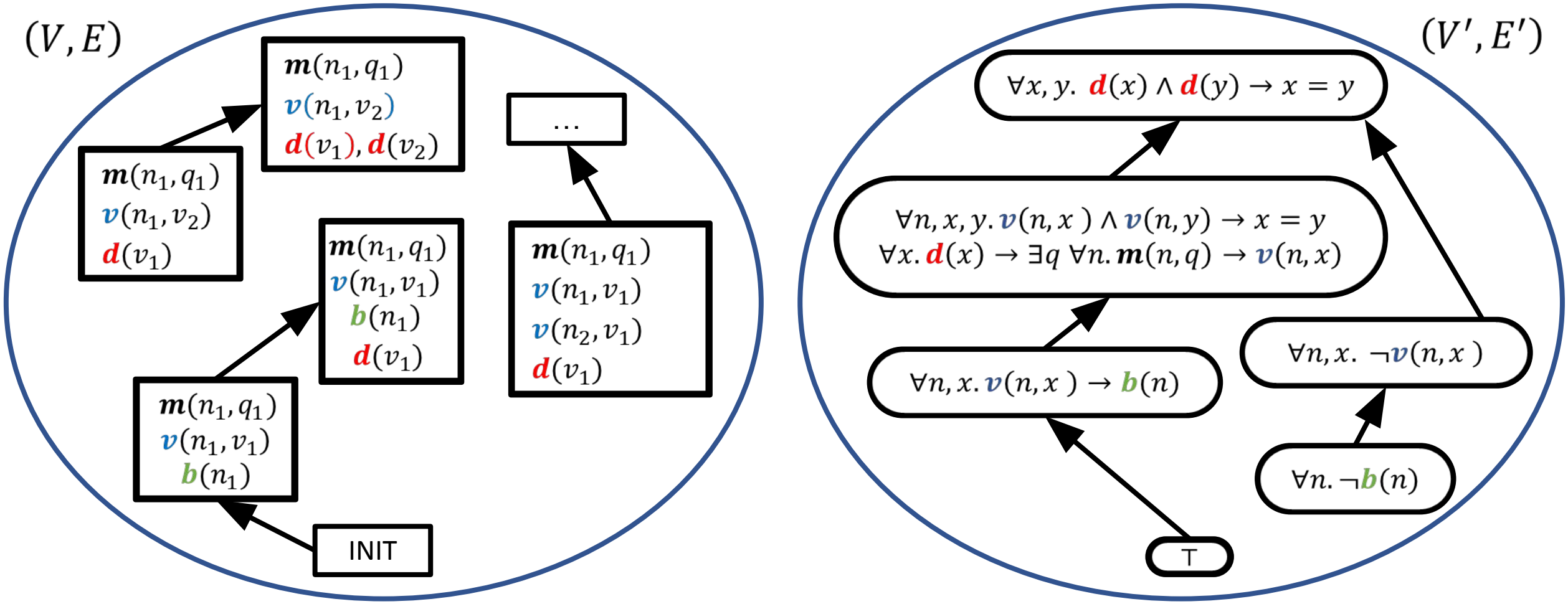
Induction Edges

- Induction edge from P to Q if:

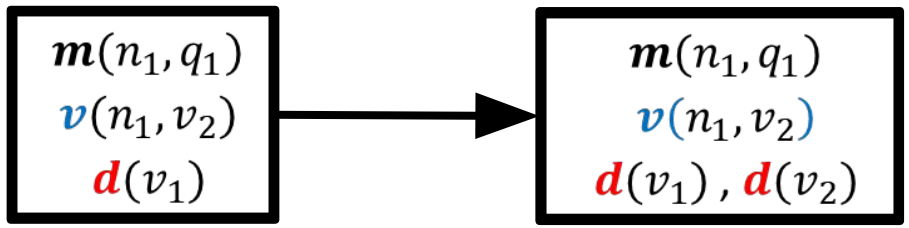
$$post([P] \cap [Q]) \cap [P] \subseteq [Q] \quad \Leftrightarrow \quad P \wedge Q \vDash wp(P \rightarrow Q)$$

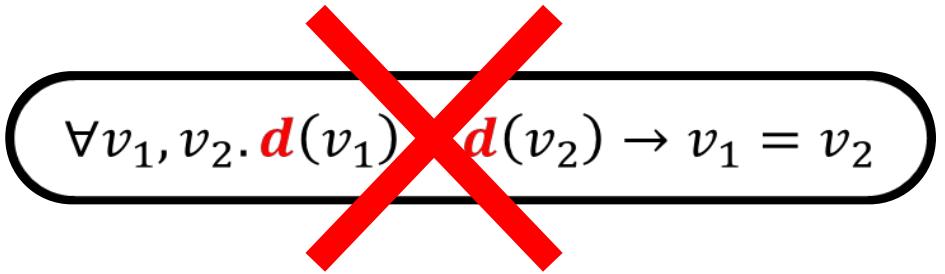
- Intuition: if P is invariant, Q is also invariant
  - If a trace violates Q, it must also violate P (no later than the violation of Q)



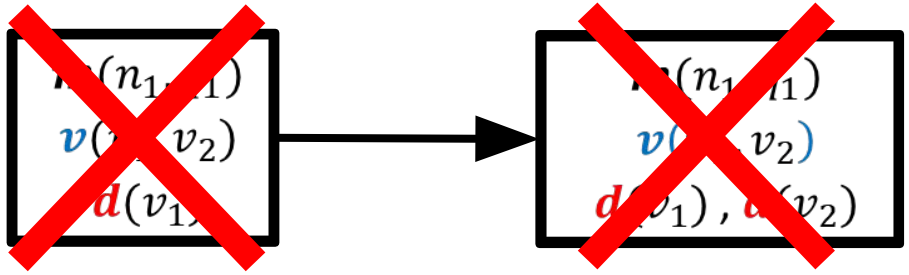

(V, E)
$\boldsymbol{m}(n_1, q_1)$
$\boldsymbol{v}(n_1, v_2)$
$\boldsymbol{d}(v_1), \boldsymbol{d}(v_2)$
$\boldsymbol{m}(n_1, q_1)$
$\boldsymbol{v}(n_1, v_2)$
$\boldsymbol{d}(v_1)$
...
$\boldsymbol{m}(n_1, q_1)$
$\boldsymbol{v}(n_1, v_1)$
$\boldsymbol{b}(n_1)$
$\boldsymbol{d}(v_1)$
$\boldsymbol{m}(n_1, q_1)$
$\boldsymbol{v}(n_1, v_1)$
$\boldsymbol{v}(n_2, v_1)$
$\boldsymbol{d}(v_1)$
$\boldsymbol{m}(n_1, q_1)$
$\boldsymbol{v}(n_1, v_1)$
$\boldsymbol{b}(n_1)$
INIT
(V', E')
$\forall x, y.\ \boldsymbol{d}(x) \wedge \boldsymbol{d}(y) \rightarrow x = y$
$\forall n, x, y.\ \boldsymbol{v}(n, x) \wedge \boldsymbol{v}(n, y) \rightarrow x = y$
$\forall x.\ \boldsymbol{d}(x) \rightarrow \exists q\ \forall n.\ \boldsymbol{m}(n, q) \rightarrow \boldsymbol{v}(n, x)$
$\forall n, x.\ \boldsymbol{v}(n, x) \rightarrow \boldsymbol{b}(n)$
$\forall n, x.\ \neg \boldsymbol{v}(n, x)$
$\forall n.\ \neg \boldsymbol{b}(n)$
T

Primal Houdini





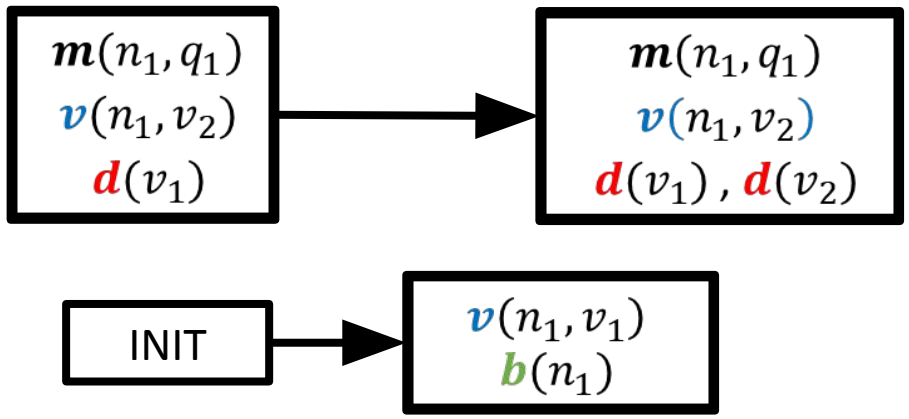
Dual Houdini

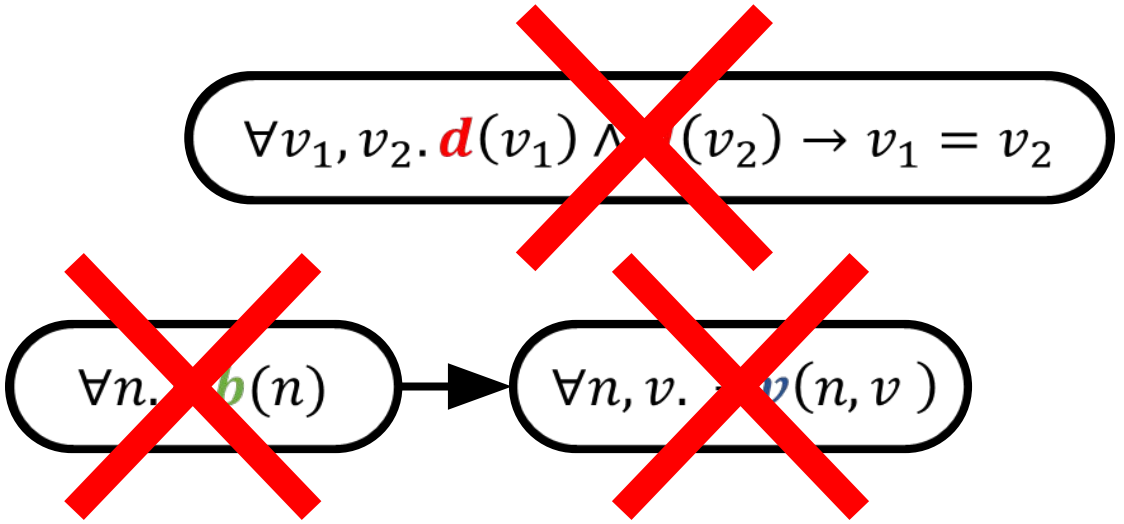


$\forall v_1, v_2.\, \boldsymbol{d}(v_1) \wedge \boldsymbol{d}(v_2) \rightarrow v_1 = v_2$

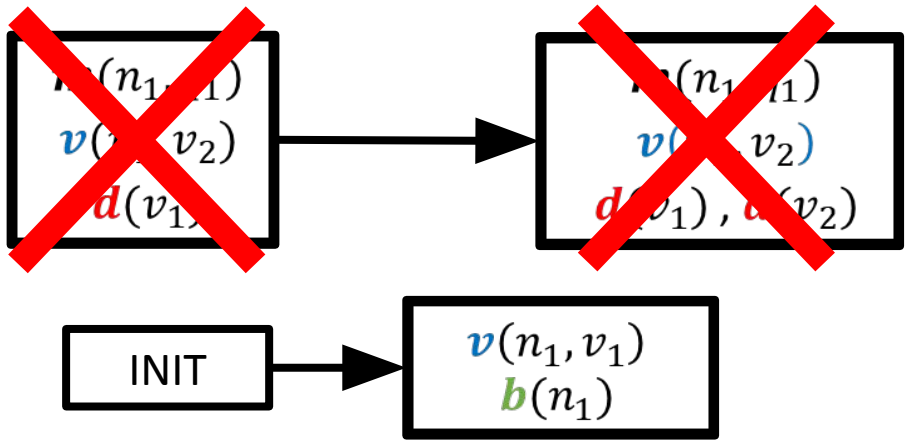
$\forall n.\, \neg \boldsymbol{b}(n)$ → $\forall n, v.\ \neg \boldsymbol{v}(n, v)$

Primal Houdini

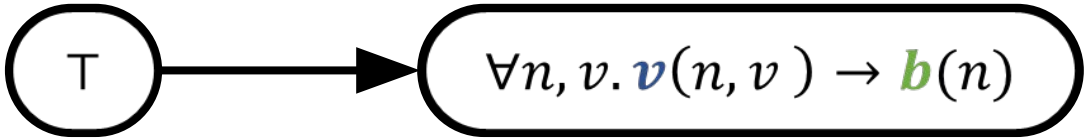




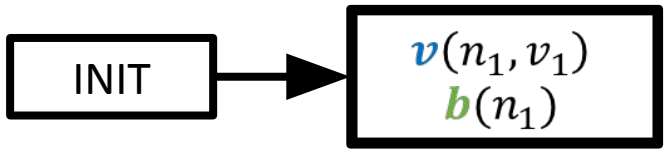
Dual Houdini

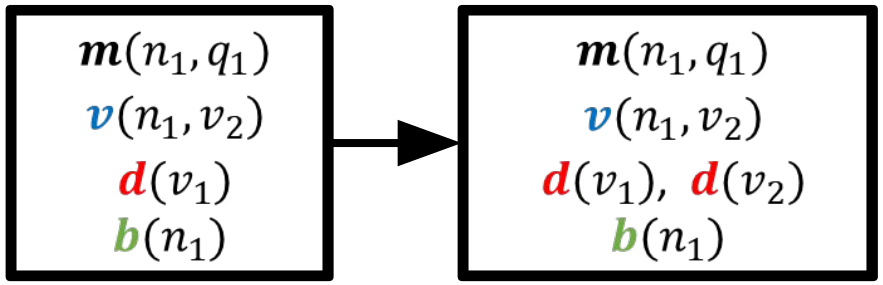


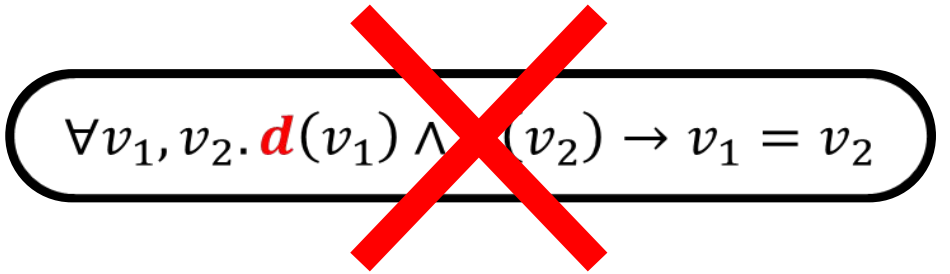
$$\forall v_1, v_2 . \boldsymbol{d}(v_1) \wedge \boldsymbol{d}(v_2) \rightarrow v_1 = v_2$$

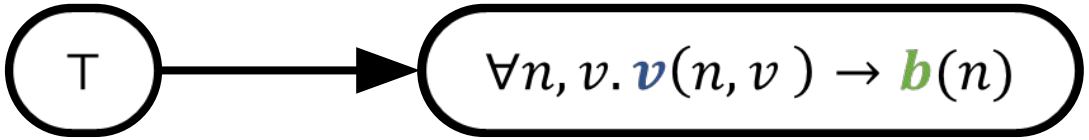


Primal Houdini



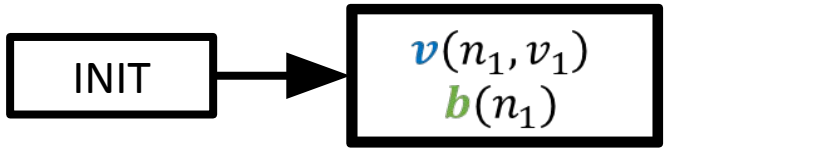


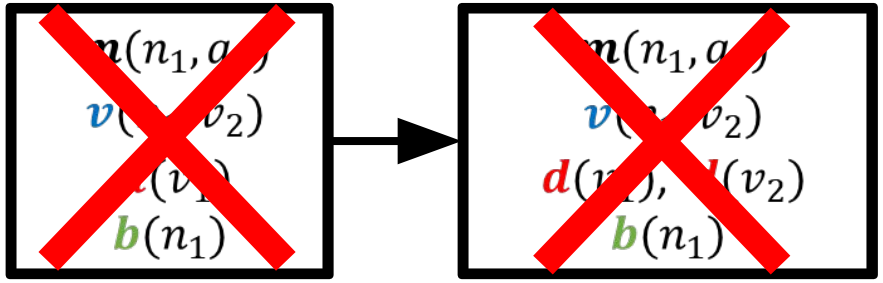


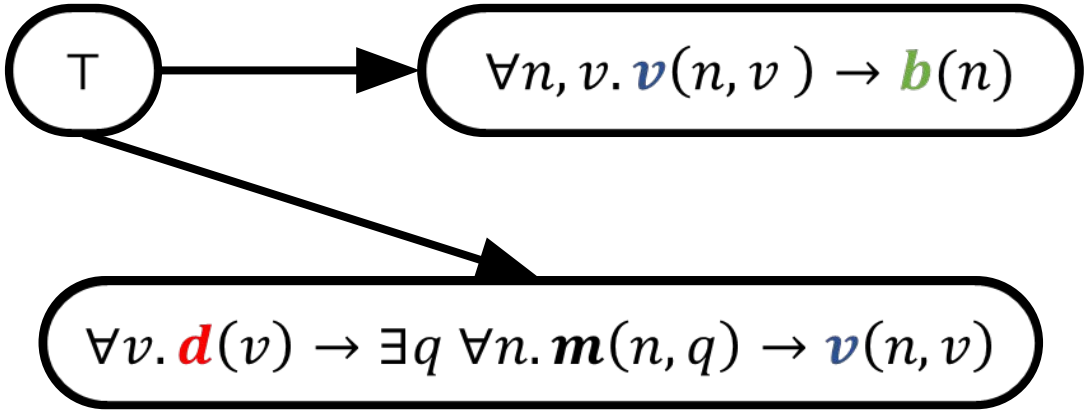


Dual Houdini

$\forall v_1, v_2.\, \boldsymbol{d}(v_1) \wedge \boldsymbol{d}(v_2) \rightarrow v_1 = v_2$



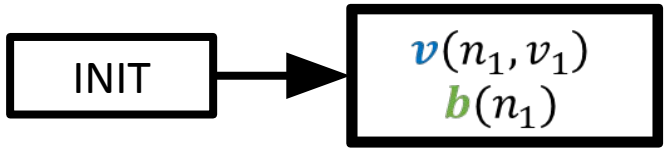


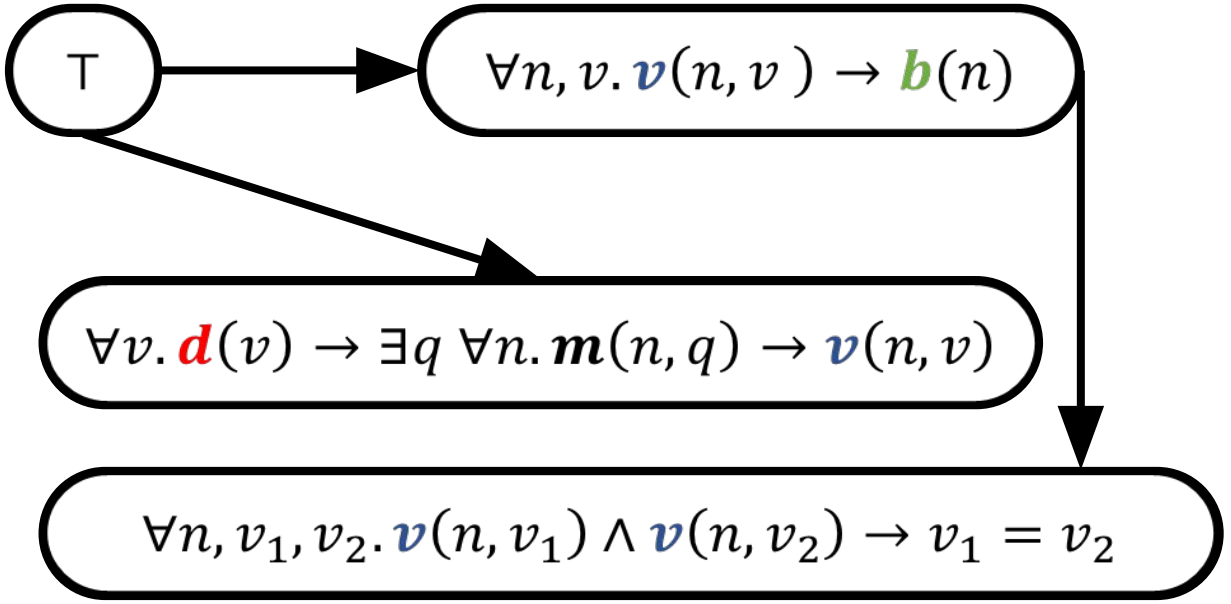


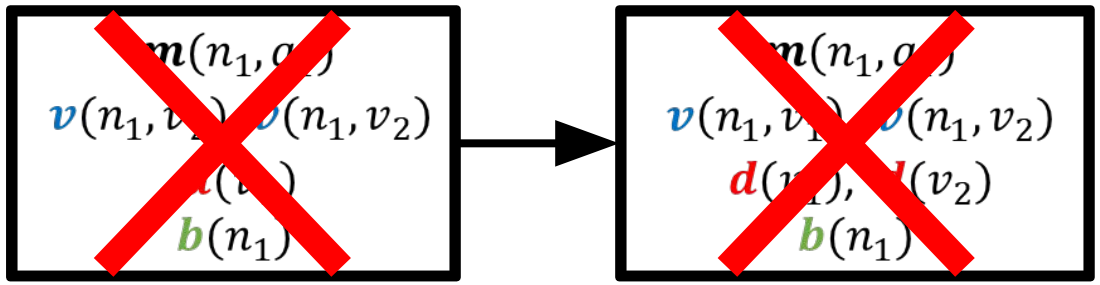
Primal Houdini

Dual Houdini

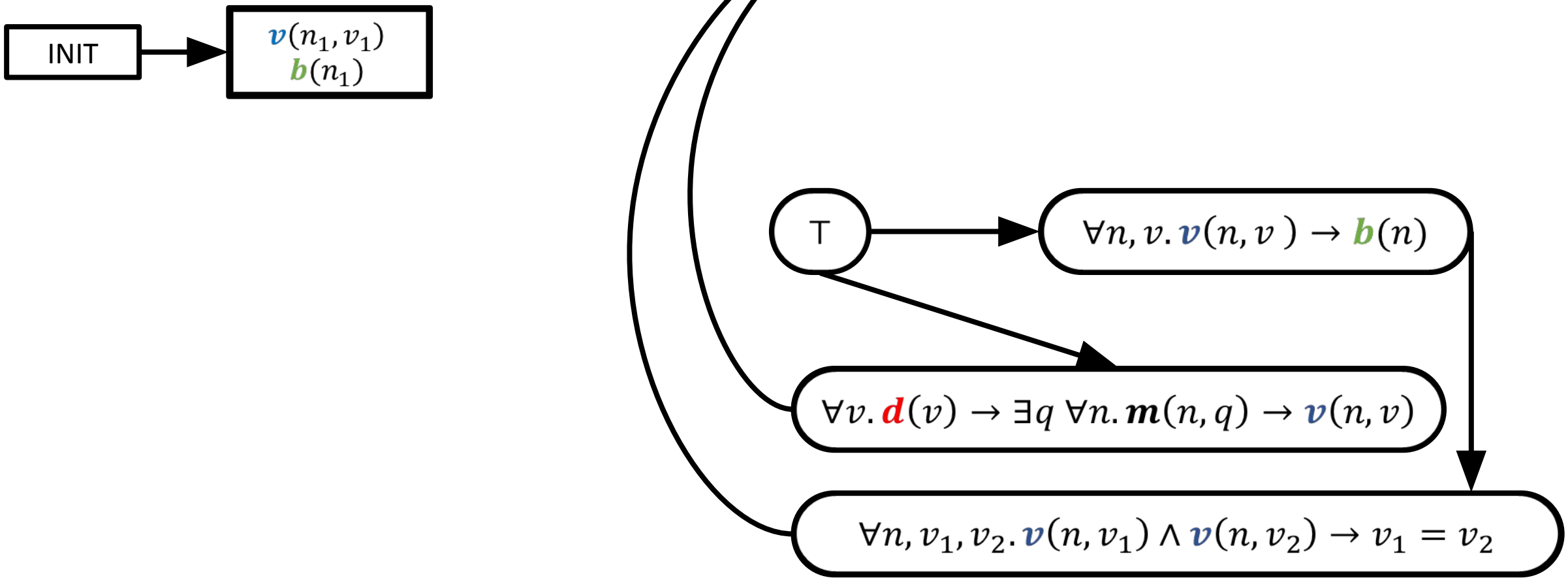
$\forall v_1, v_2.\, \boldsymbol{d}(v_1) \wedge \boldsymbol{d}(v_2) \rightarrow v_1 = v_2$







Primal Houdini



Primal-Dual Houdini: Theorems

- Exploration: every iteration discovers new states and predicates
- Possible Progress:
  - If k-provable, every iteration can discover a new useful predicate
  - If unsafe, every iteration can discover a new useful state
- Termination by Stratification: if states and predicates are discovered in a stratified manner, then the algorithm terminates for k-provable or unsafe cases

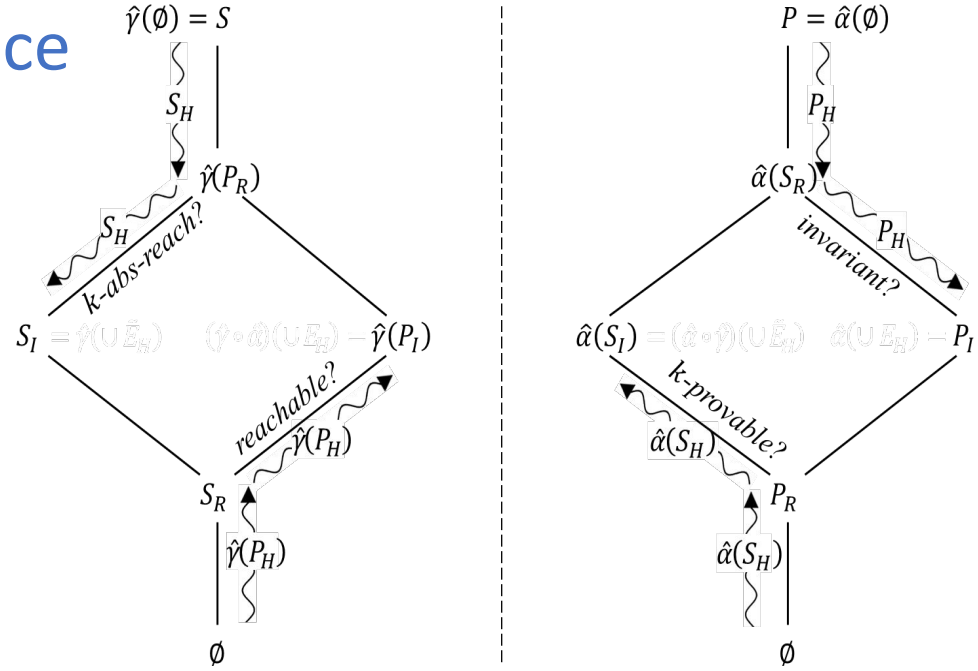
Evaluation for Distributed Protocols ($k = 1$)

Example	pdH	UPDR	FOL-IC3	SWISS	IC3PO	DistAI
ring	✓	✓	✓	✓	✓	✓
cons	✓	✓	✓	✓	✓	✓
paxos	✓	✓		✓		
spaxos	✓				✓	
paxos-h	✓					
spaxos-h						
locksrv	✓	✓	✓	✓	✓	✓
skv		✓	✓	✓	✓	✓
skvr		✓				
cache					✓	

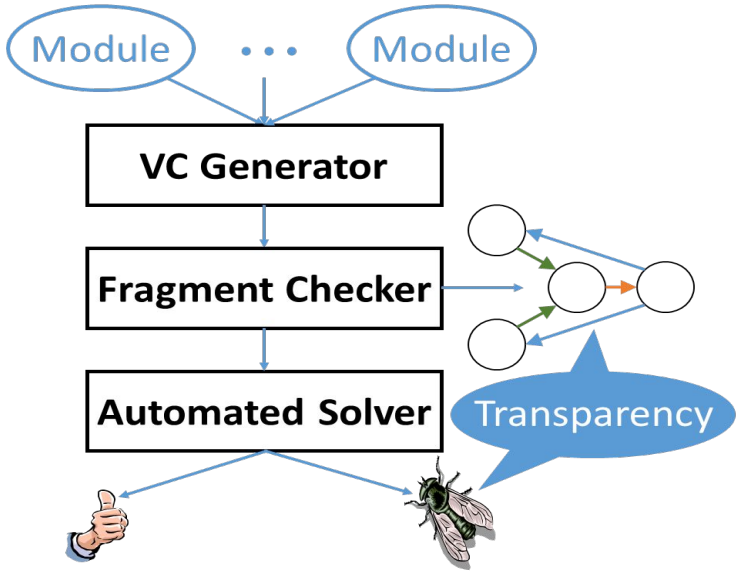
[POPL'22] O. P., James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. Induction Duality: Primal-Dual Search for Invariants.

Conclusion (Primal-dual Houdini)

- Goal: symmetric primal-dual invariant inference
- Key idea: bounded incremental induction
- Result 1: Induction duality
- Result 2: Primal-Dual Houdini
- New synthesis task: check dual-inductiveness and find induction edges
- Future directions
  - Application to more domains
  - Primal-dual version of more advanced algorithms



Conclusion



- Using undecidable reasoning is powerful, but at a price
- Decidable logic (EPR) offers a different tradeoff
  - More effort in encoding systems and properties
  - More reliable automation
- Reliable invariant checking opens the path to invariant inference
  - Many recent ideas (and new adaptations of old ideas)
  - We are exhausting the benchmark set – progress would come from new benchmarks!
- Induction duality
  - Promising new idea
  - May be applicable in other domains
- Reach out to discuss more or collaborate!