Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv

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

Contributors

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

TEL AVIV UNIVERSITY

UCLA

Microsoft Research

TEL AVIV UNIVERSITY

TEL AVIV UNIVERSITY

James R. Wilcox, Doug Woos

UNIVERSITY of WASHINGTON

And Also

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





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

Virtual Machine

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

Deductive Verification of Distributed Protocols in First-Order Logic

[CAV'13] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, MS:

Effectively-Propositional Reasoning about Reachability in Linked Data Structures

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

[POPL'16] Oded Padon, Neil Immerman, Aleksandr Karbyshev, Sharon Shoham, MS
Decidability of Inferring Inductive Invariants

[OOPSLA'17] Oded Padon, Giuliano Losa, MS, Sharon Shoham
Paxos made EPR: Decidable Reasoning about Distributed Protocols

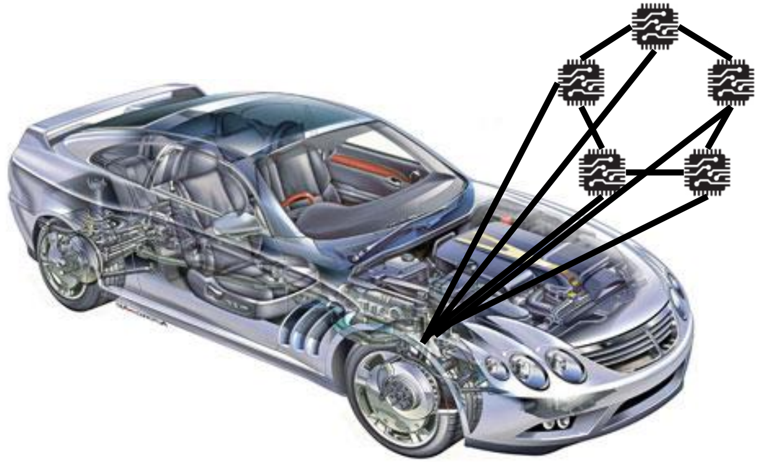
[PLDI'18] Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, MS, Sharon Shoham, James R. Wilcox, Doug Woos: Modularity for Decidability of Deductive Verification with Applications to Distributed Systems

Agenda

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

Why verify distributed protocols?

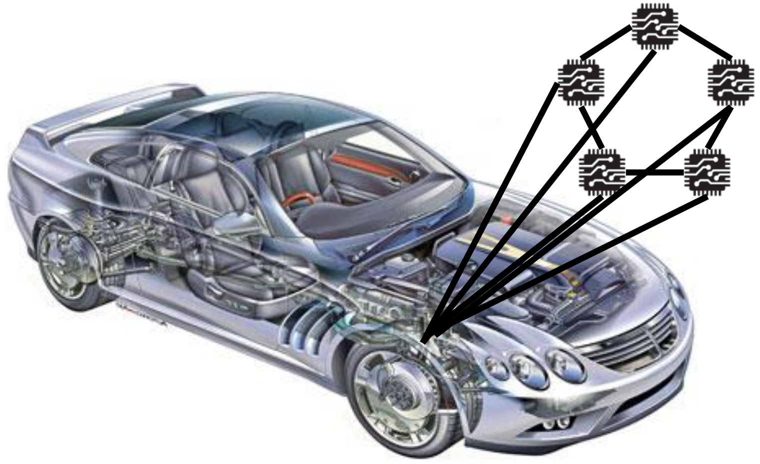
- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchain



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

Why verify distributed protocols?

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



- Distributed systems are notoriously hard to get right

SIGCOMM'01

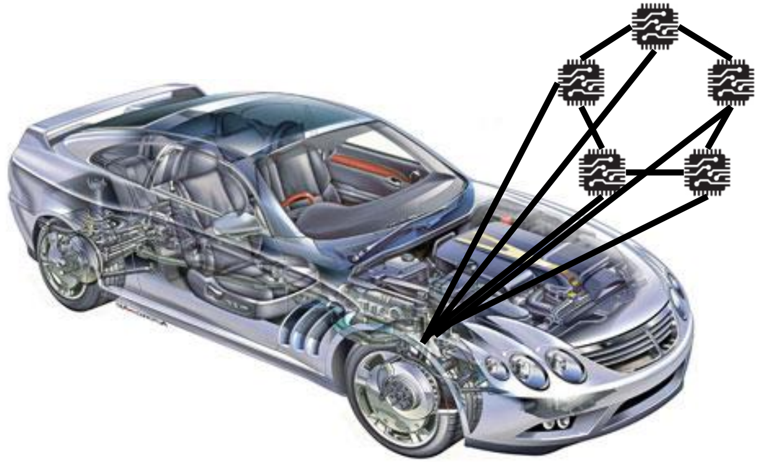
Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Member, IEEE*

Attractive features of Chord include its simplicity, provable correctness, and provable performance even in the face of concurrent node arrivals and departures. It continues to func-

Why verify distributed protocols?

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



- Distributed systems are notoriously hard to get right

SIGCOMM'01

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

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

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

CCR'12

Using Lightweight Modeling To Understand Chord

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

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

SOSP'07

Best Paper Award

Zyzzyva: Speculative Byzantine Fault Tolerance

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

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

CACM'08

Zyzzyva: Sp... Fault...

ACM Transactions on Computer Systems '09

Zyzzyva: Speculative Byzantine Fault Tolerance

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

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

Revisiting Fast Practical Byzantine Fault Tolerance

Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

with:
Lorenzo Alvisi (Cornell),
Rama Kotla (Amazon),
Jean-Philippe Martin (Verily)

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

Proving distributed systems is hard

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

Our goal: reduce human effort while maintaining flexibility

Our approach: decompose verification into decidable problems

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

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

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

Automatic verification of infinite-state systems

System S

Property φ

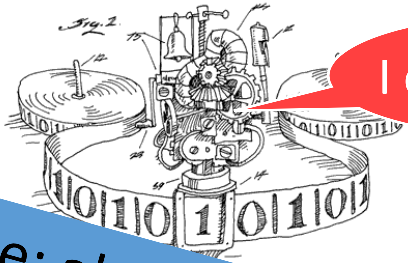
Verification

behavior

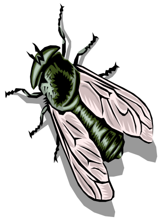
"Program verification is the holy grail of computer science; always was; always will be"

Chet Murthy

Rice's Theorem



Counterexample

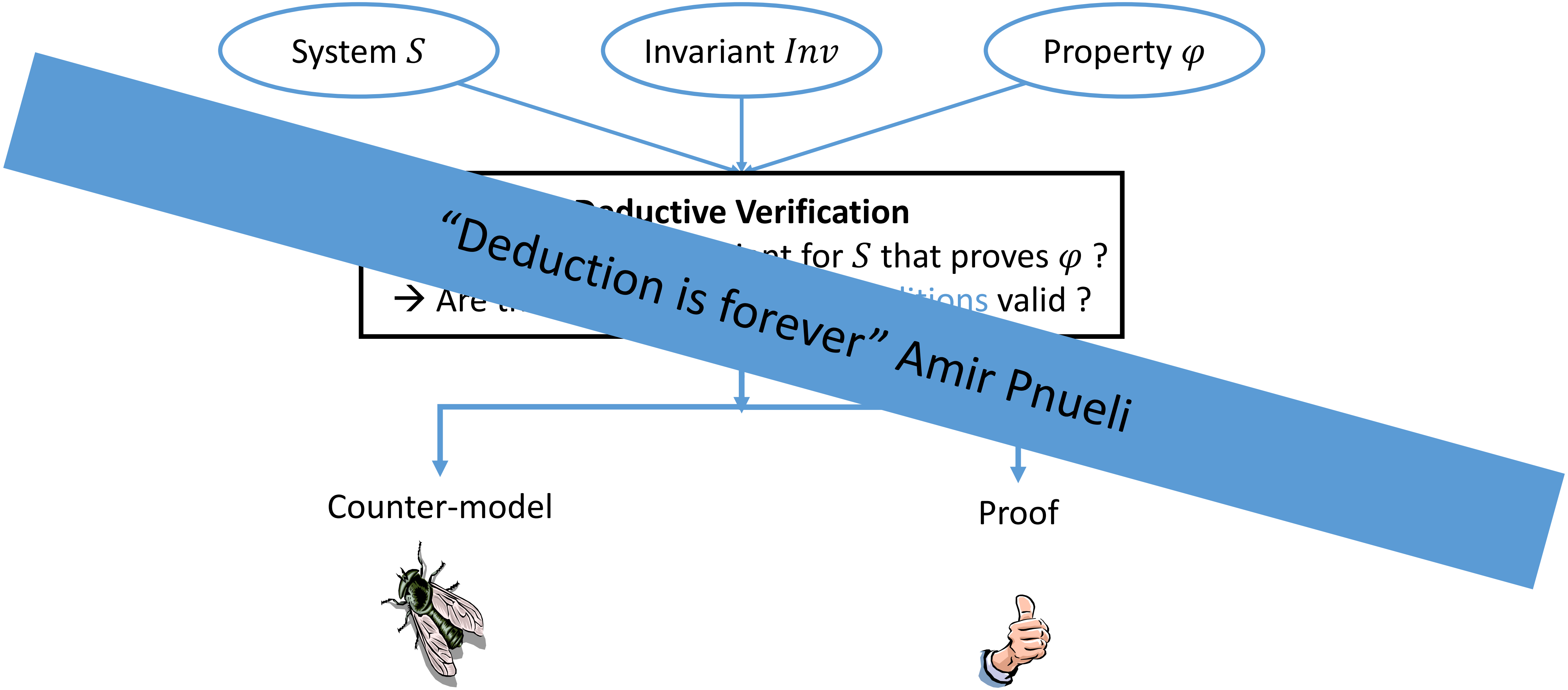


Unknown / Diverge

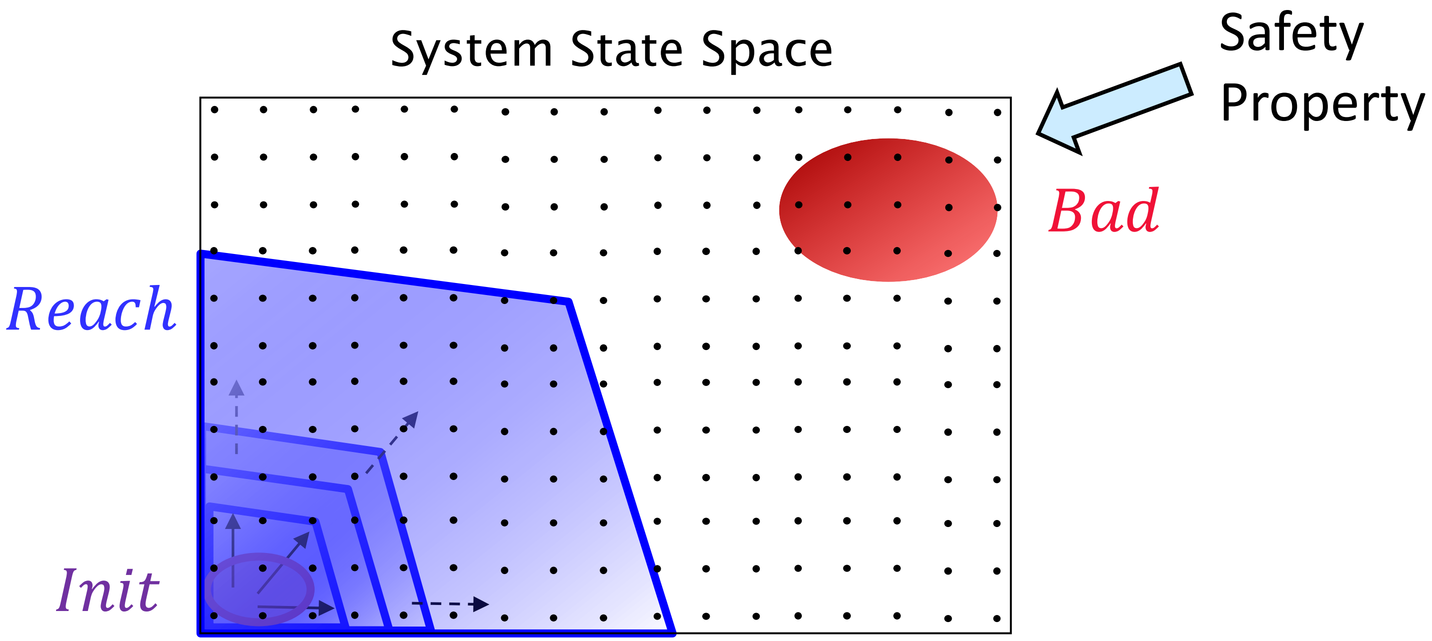
Proof

Semi-automatic deductive verification

Deductive verification

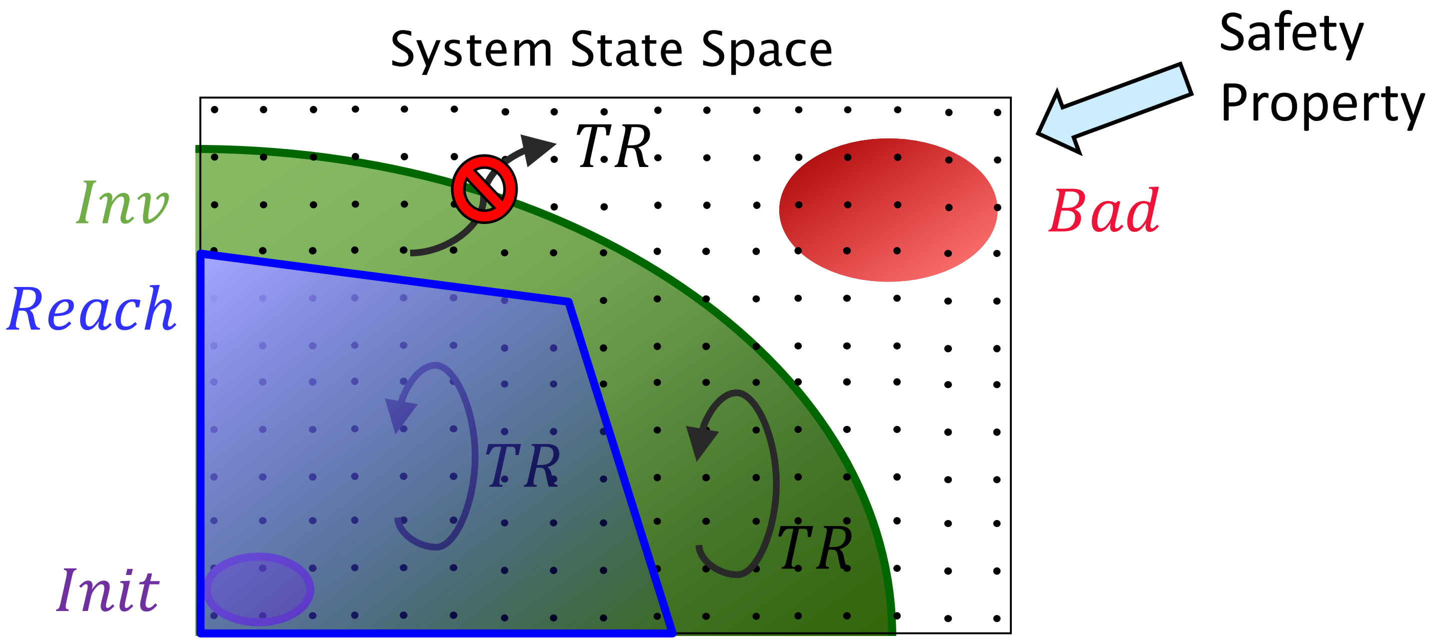


Inductive invariants



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

Inductive invariants



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

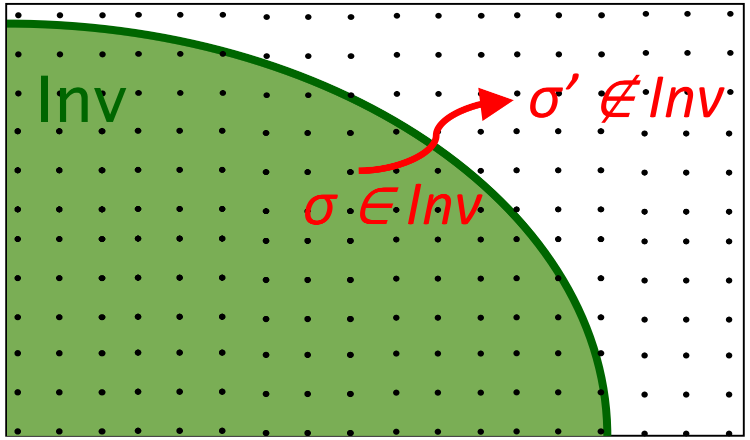
System S is safe iff there exists an **inductive invariant** Inv :

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

} translated to VC's

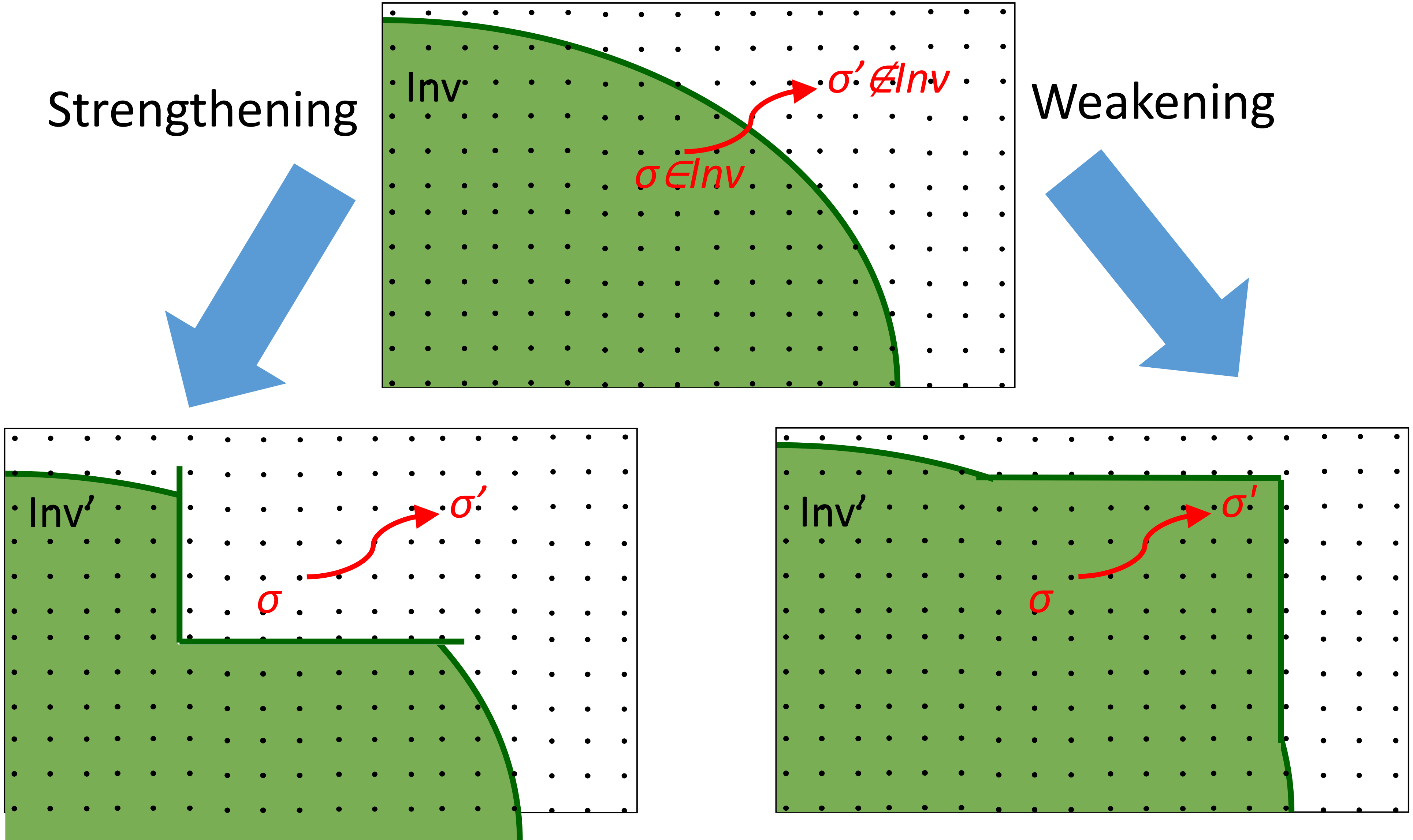
Counterexample To Induction (CTI)

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



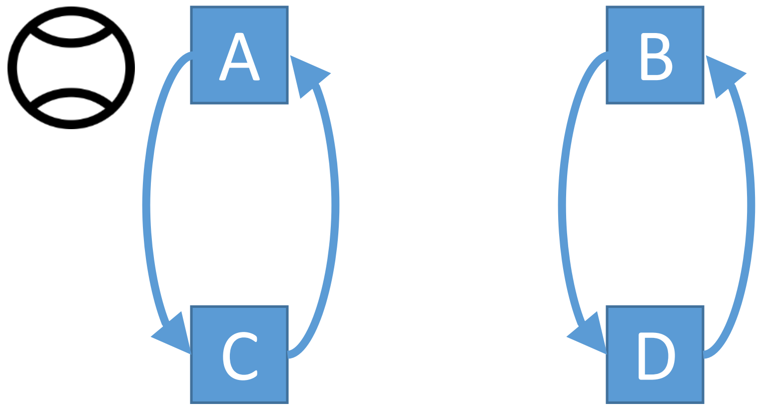
- A CTI may indicate:
  - A bug in the system
  - A bug in the safety property
  - A bug in the inductive invariant
    - Too weak
    - Too strong

Strengthening & weakening from CTI



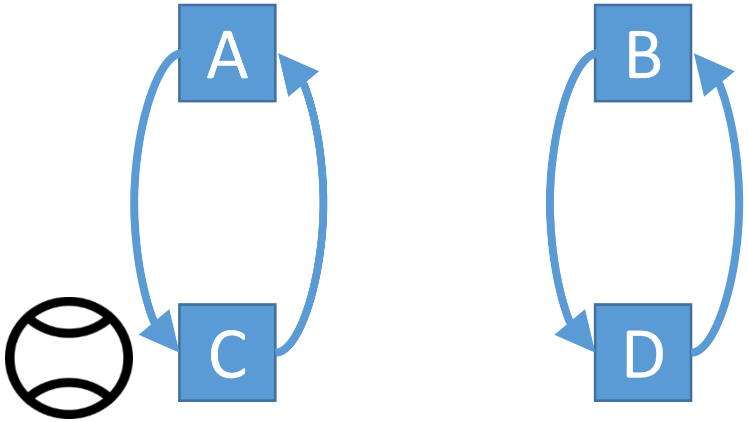
Induction on a ball game

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



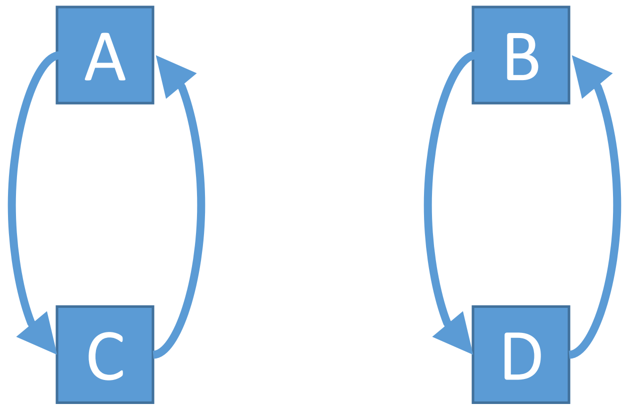
Induction on a ball game

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



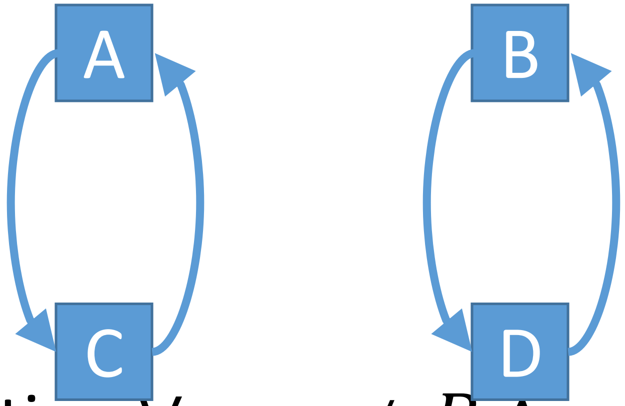
Formalizing with induction

- $x_0 = A$
- $x_{n+1} = \begin{cases} C \; if \; x_n = A \\ D \; if \; x_n = B \\ A \; if \; x_n = C \\ B \; if \; x_n = D \end{cases}$
- Prove by induction $\forall n. x_n \neq D$
  - $x_0 \neq D$?
  - $x_m \neq D \Rightarrow x_{m+1} \neq D$?



Formalizing with induction

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

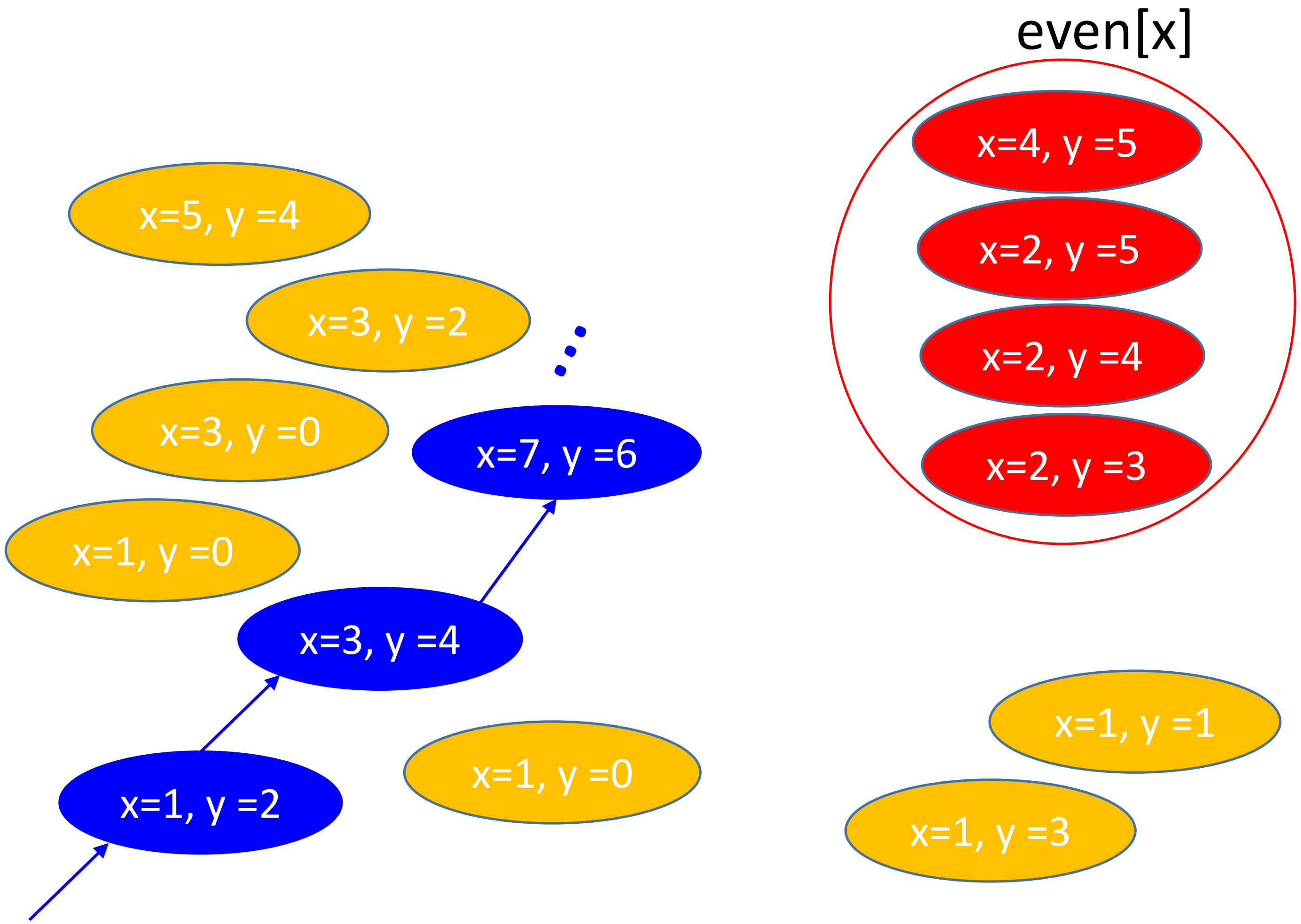


- Prove a stronger claim by induction $\forall n. x_n \neq B \wedge x_n \neq D$
  - $x_0 \neq B \wedge x_0 \neq D$
  - $x_m \neq B \wedge x_m \neq D \Rightarrow x_{m+1} \neq B \wedge x_{m+1} \neq D$

Simple example: loop invariants

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

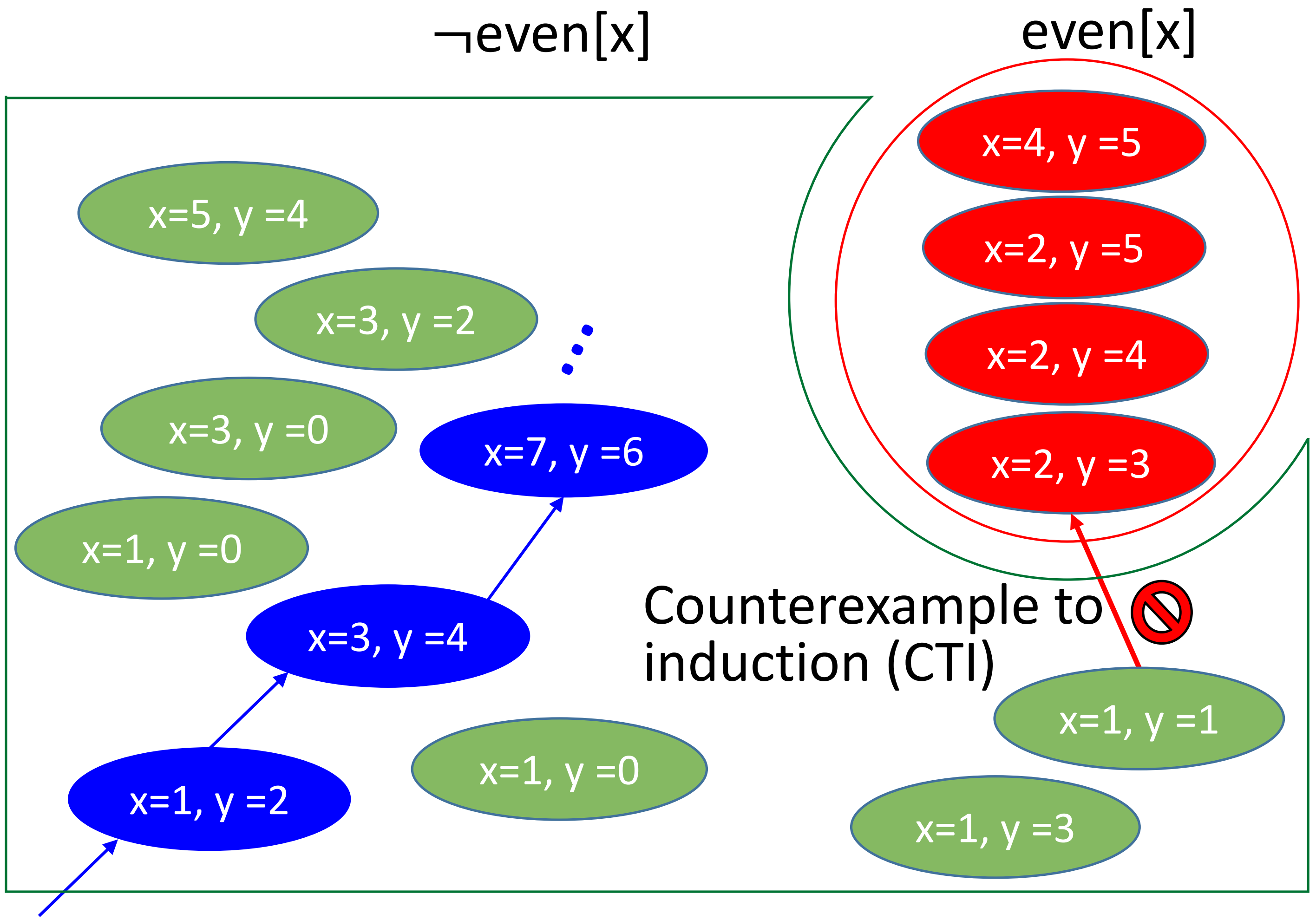
TR (bracketing the two assignments in the loop body)



Simple example: loop invariants

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

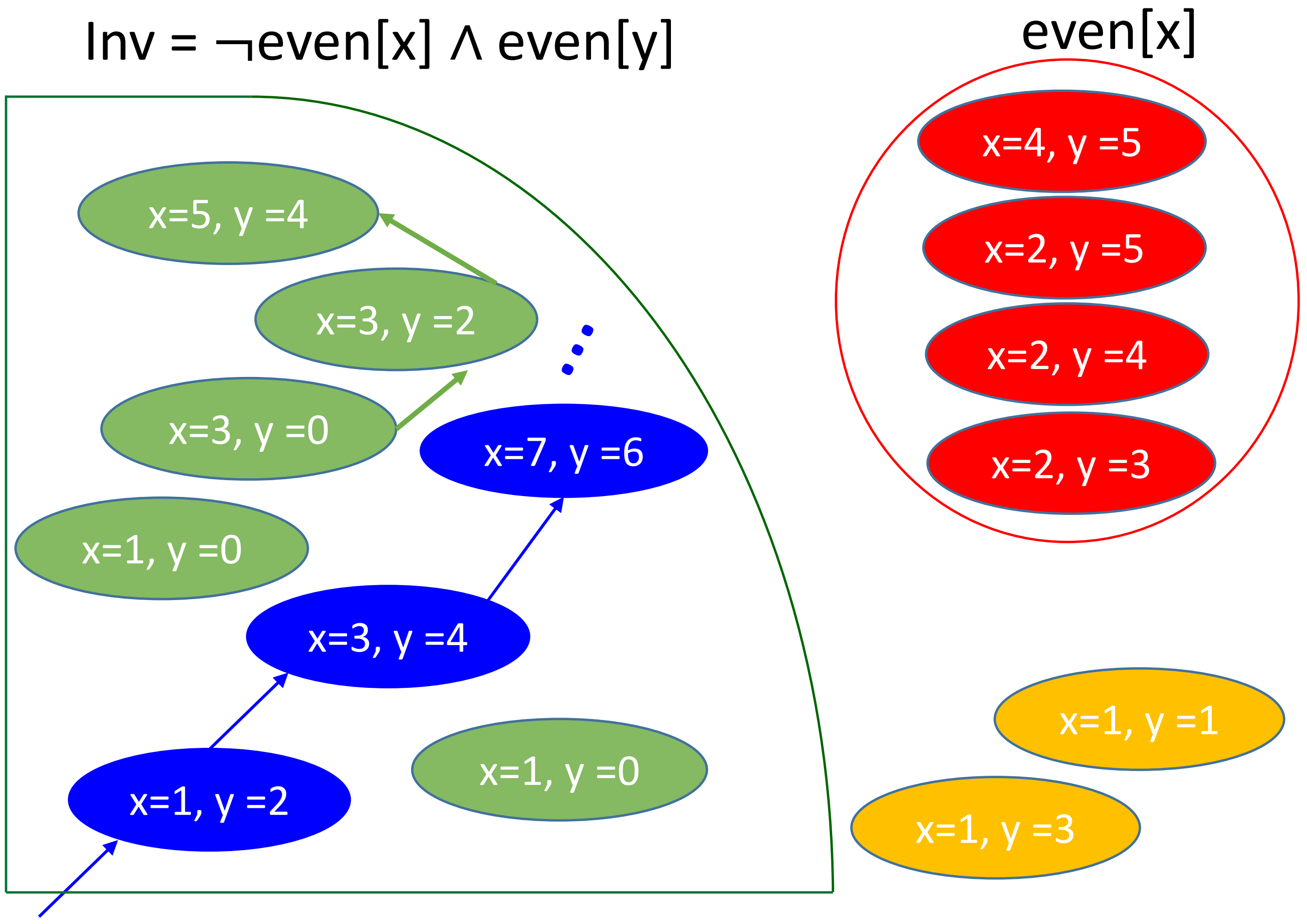
TR (marks the lines `x := x + y;` and `y := y + 2;`)



Simple example: loop invariants

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

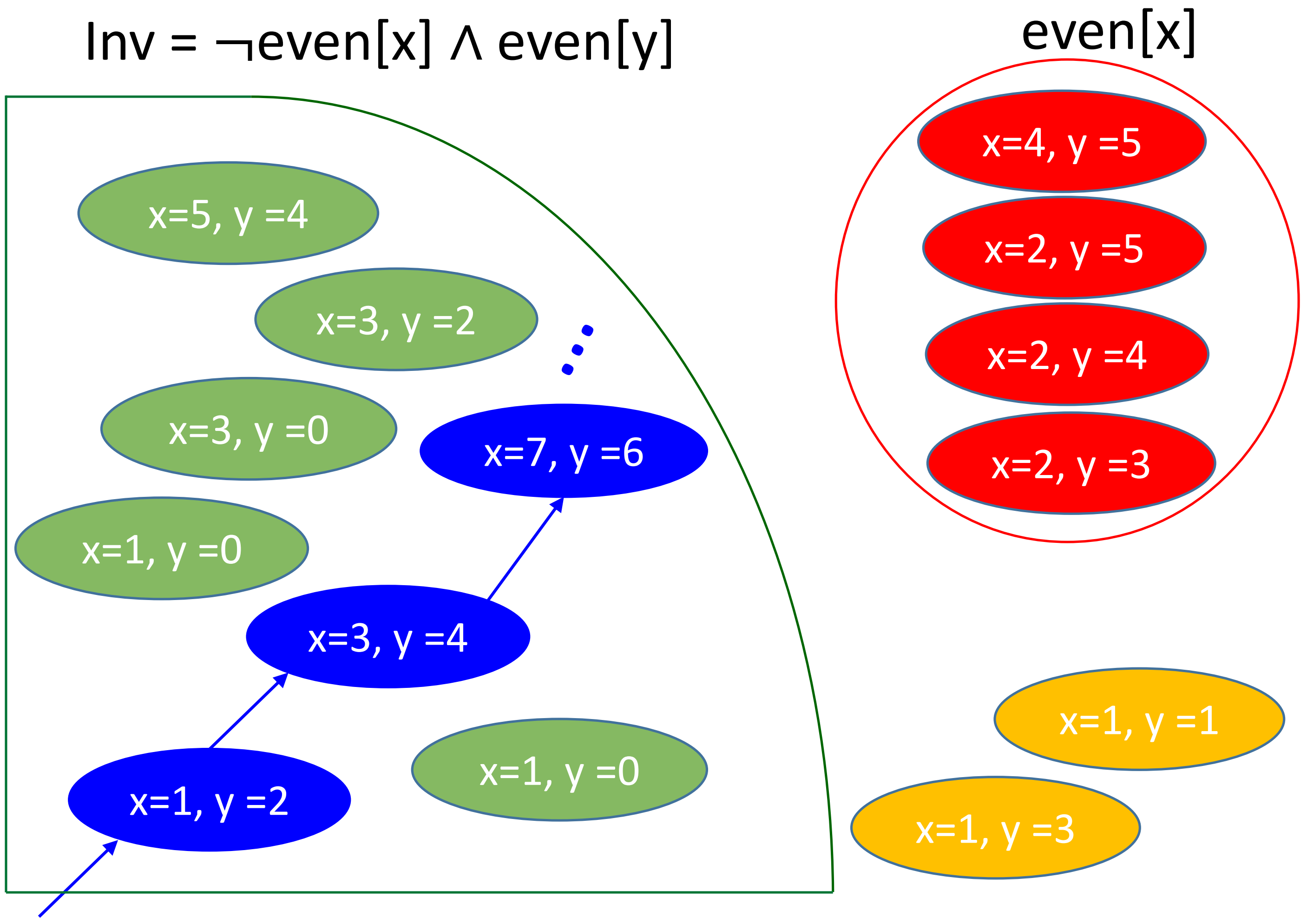
TR (marks `x := x + y; y := y + 2;`)



Simple example: loop invariants

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

TR (annotating the lines `x:=(x*x–y*y)/(x-y);` and `y := y + 2;`)

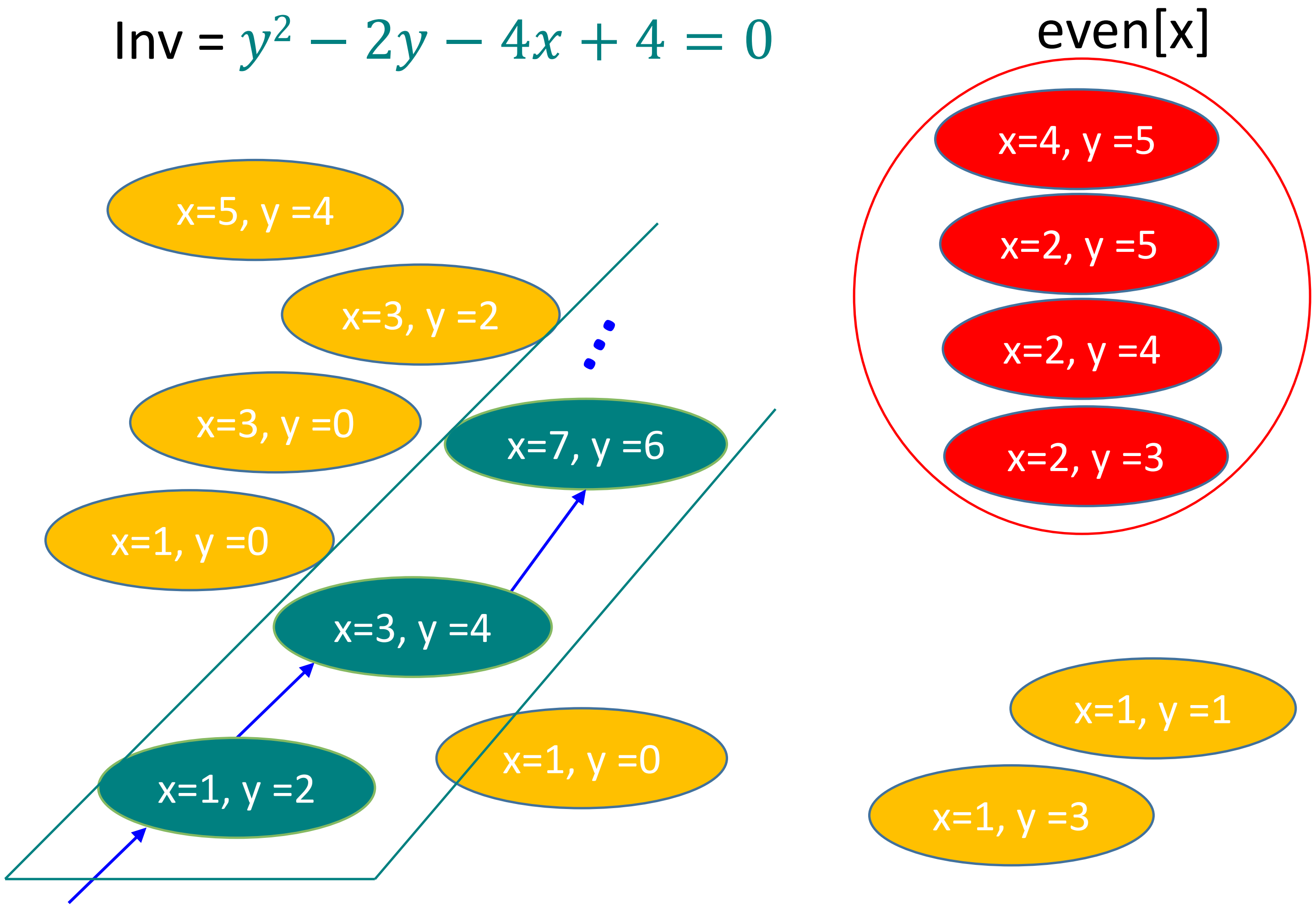


Simple example: loop invariants

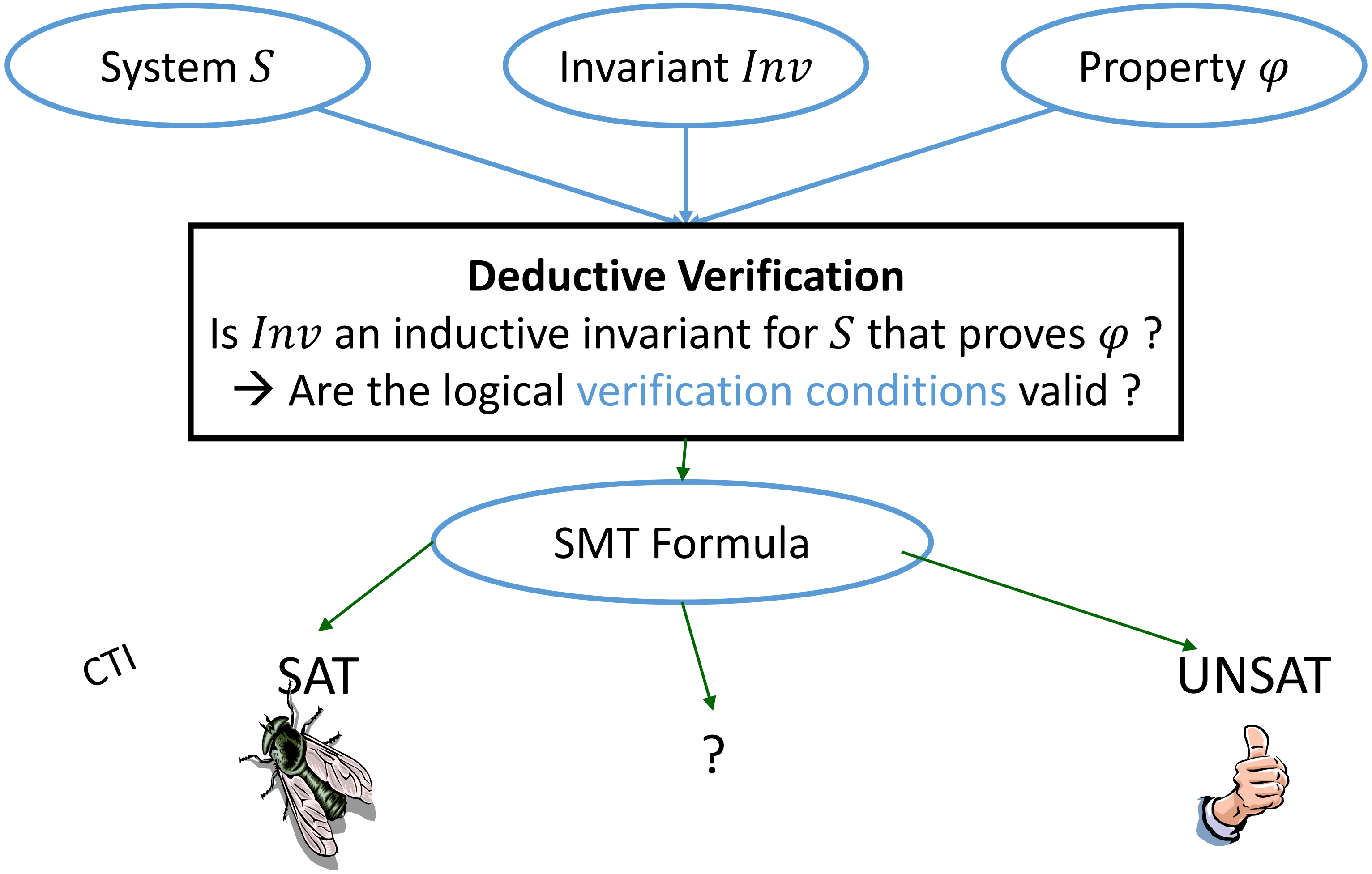
```
x := 1;
y := 2;
while * do {
 assert ¬even[x];
 x := x + y;
 y := y + 2;
}
```

TR (bracketing `x := x + y;` and `y := y + 2;`)

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



Dafny [Leino'17]



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

Deductive verification

System S

Invariant Inv

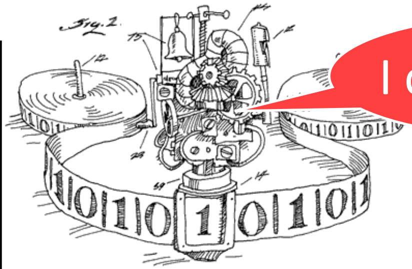
Property φ

Deductive Verification

Is Inv an inductive invariant for S that proves φ ?

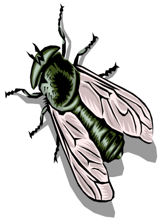
→ Are the logical verification conditions valid ?

Church's Theorem



I can't decide!

Counter-model

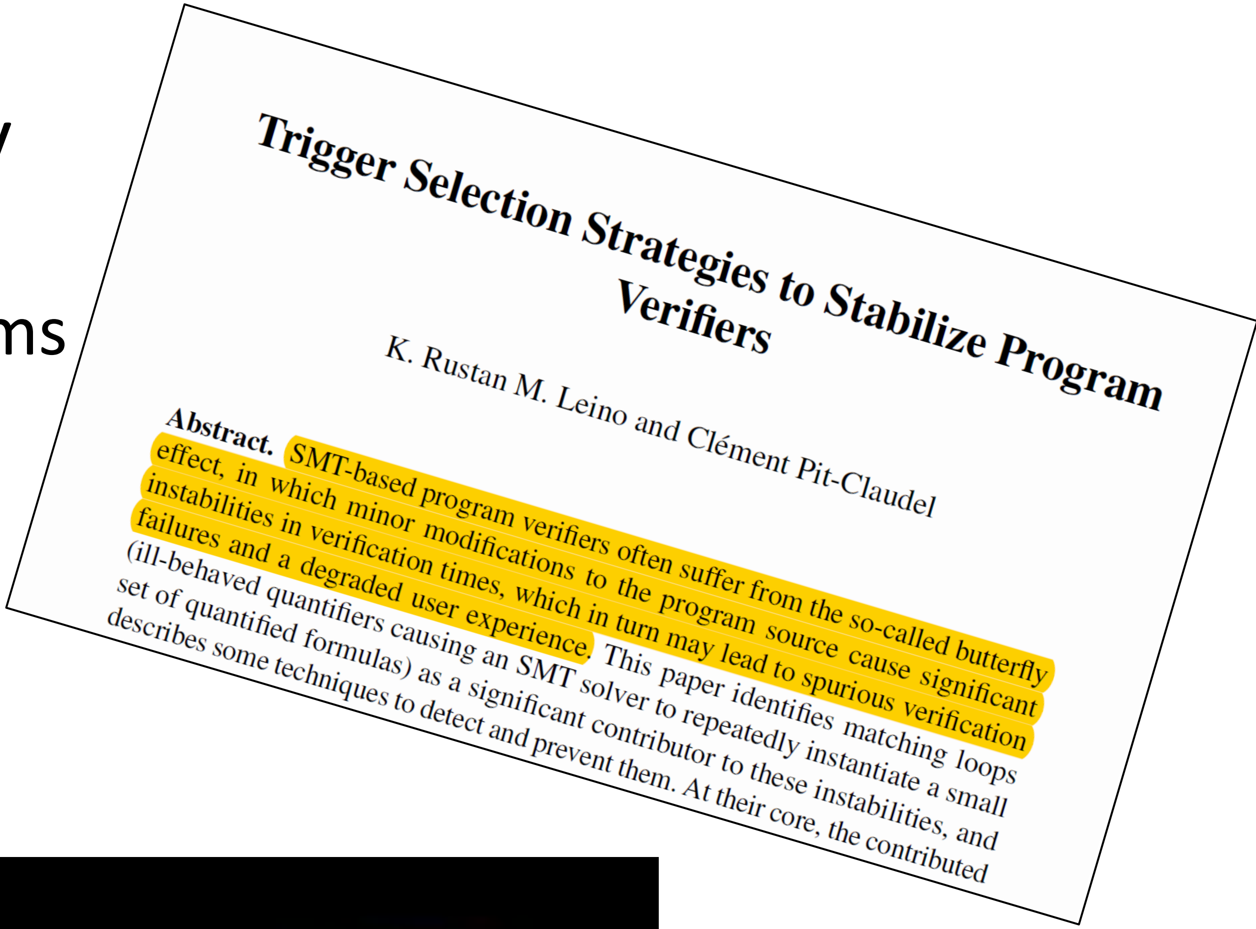


Unknown / Diverge

Proof

Effects of undecidability

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



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

Challenges in deductive verification

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

State of the art in formal verification

Proof Assistants

Ultimately limited by human

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

Ivy

Decidable deduction
Finite counterexamples
proof/code: ~0.2

Ultimately limited by undecidability

Decidable Models
Model Checking
Static Analysis

Expressiveness

Automation

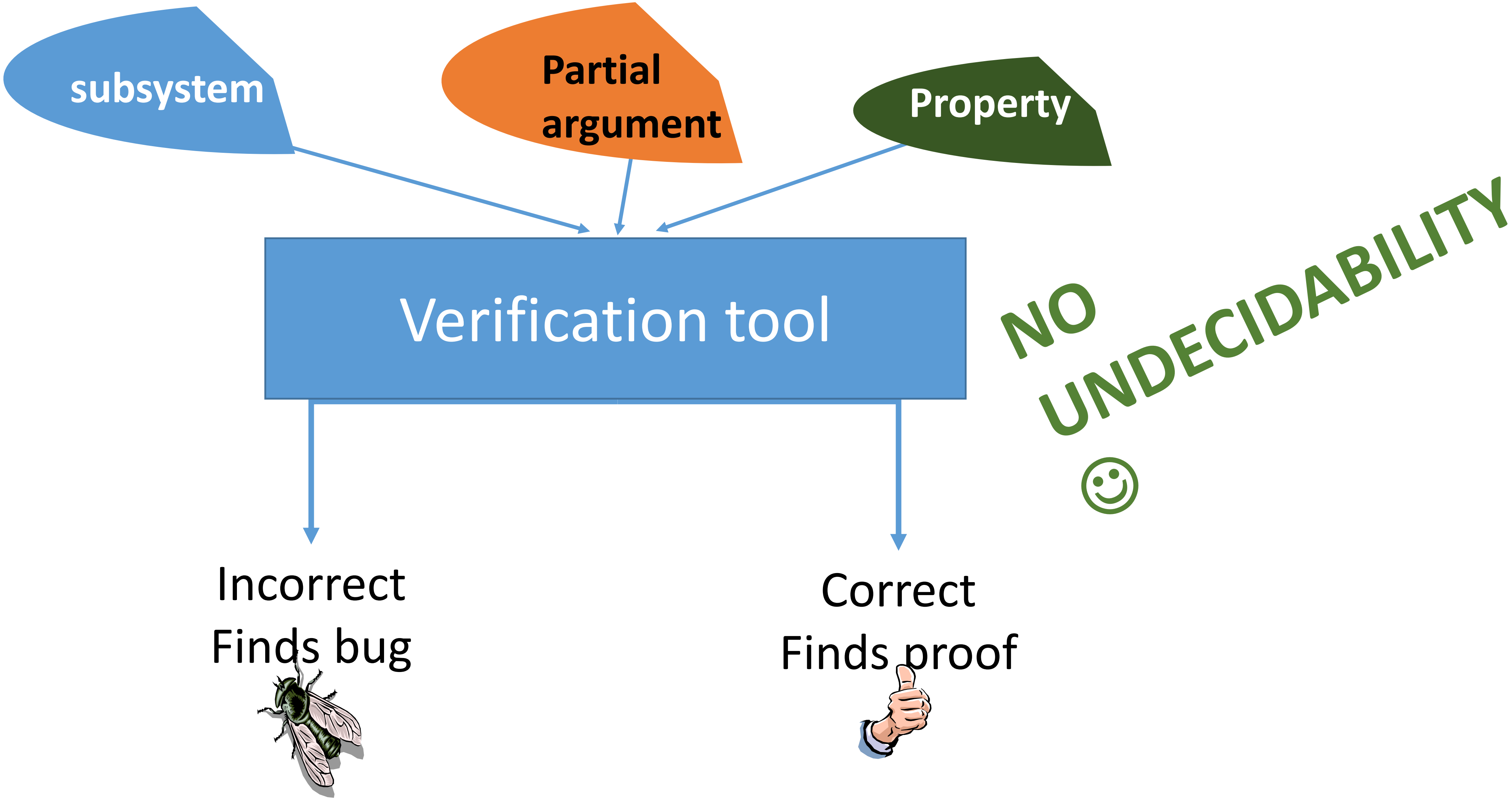
Modularity

Original system

Original inductive argument

Original property

Verification of each module



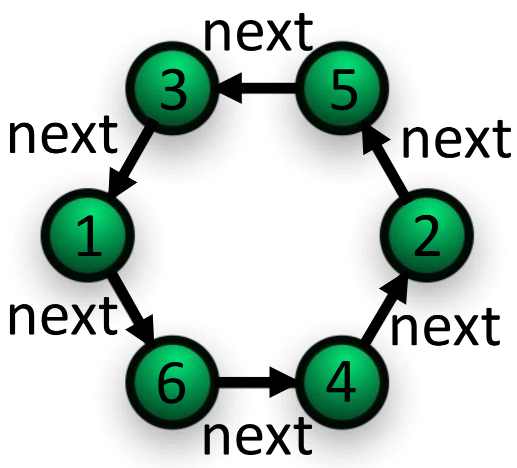
Ivy's principles

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

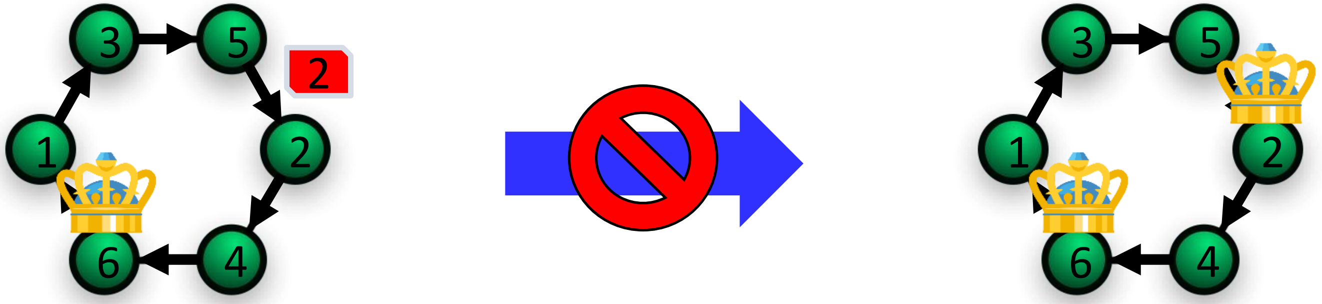
Languages and verification

Language	Executable	Expressiveness	Inductiveness
C, Java, Python…	☑	Turing-Complete	Undecidable
SMV	☒	Finite-state	Temporal Properties
TLA+	☒	Turing-Complete	Manual
Coq, Isabelle/HOL	☑	Turing-Complete	Manual with tactics
Dafny	☑	Turing-Complete	Undecidable with lemmas
Ivy	☑	Turing-Complete	Decidable(EPR)

Example: Leader election in a ring

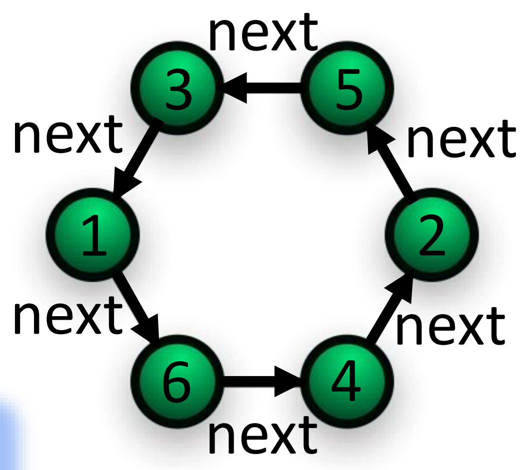


- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node’s own id
  - A node that receives its own id becomes a leader
- Theorem: The protocol selects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



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

Example: Leader election in a ring



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

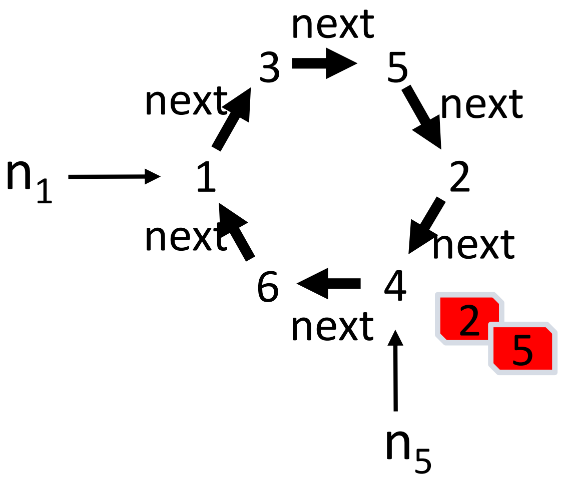
[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

Leader election protocol – first-order logic

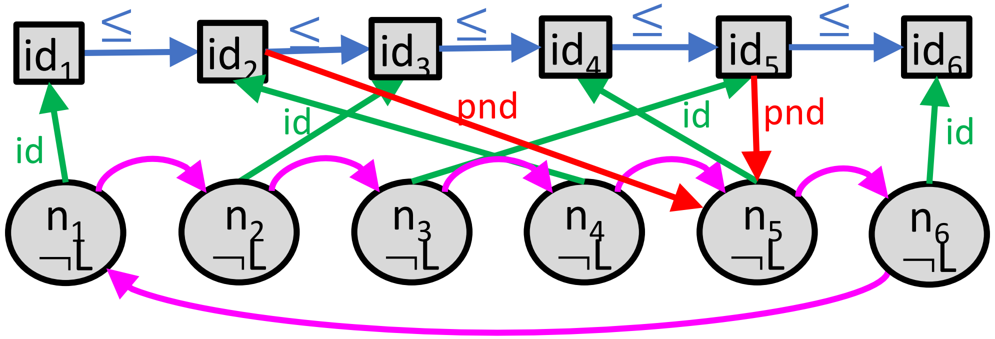
- $\leqslant$ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



first-order structure



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

Leader election protocol – first-order logic

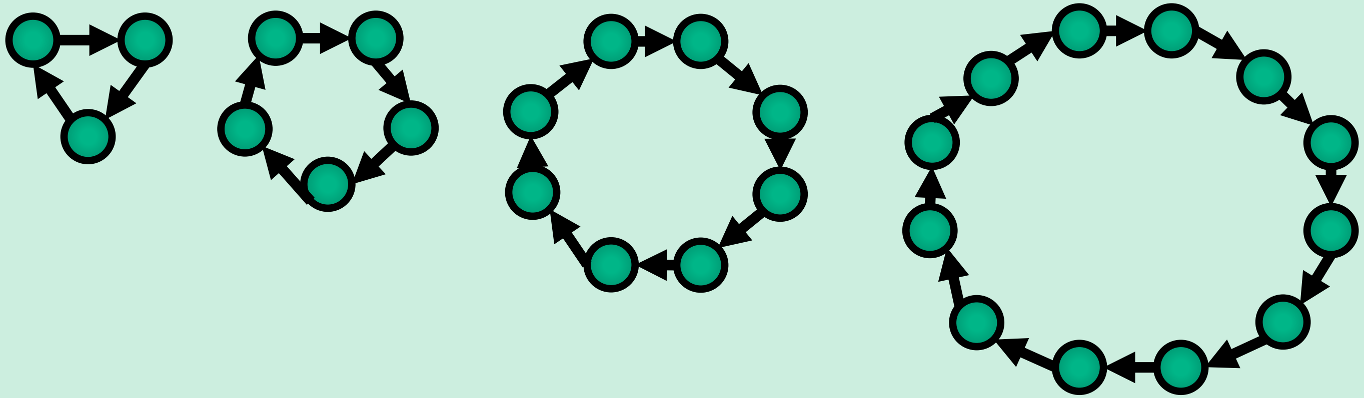
- ⩽ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} 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
}
```

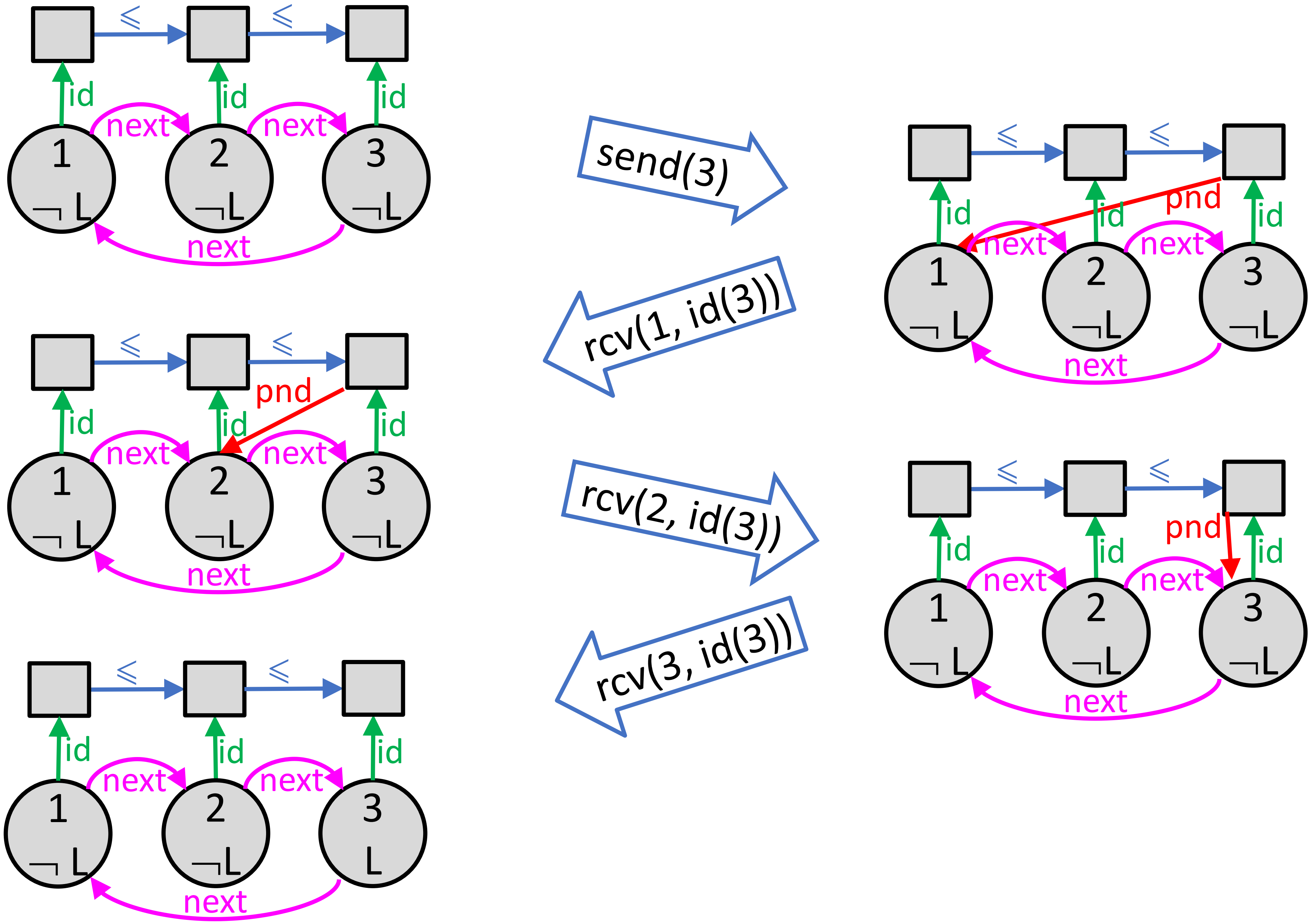
```
action receive(n: Node, m: ID) = {
  requires pending(m, n);
  if id(n) = m then
    // found leader
    leader(n) := true
  else if id(n) ⩽  m then
    // pass message
    "s := next(n)";
    pending(m, s) := true
}
```

TR (send):

```
∃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
```


send(3)
rcv(1, id(3))
rcv(2, id(3))
rcv(3, id(3))
id
next
pnd
⩽
1
2
3
¬L
L

Leader election protocol – inductive invariant

Safety property: I_0

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

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

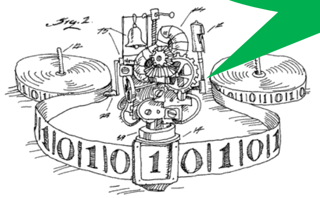
$I_1 = \forall n_1$ …

$I_2 = \forall n_1$ … n be self-pending

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

Cannot bypass higher nodes

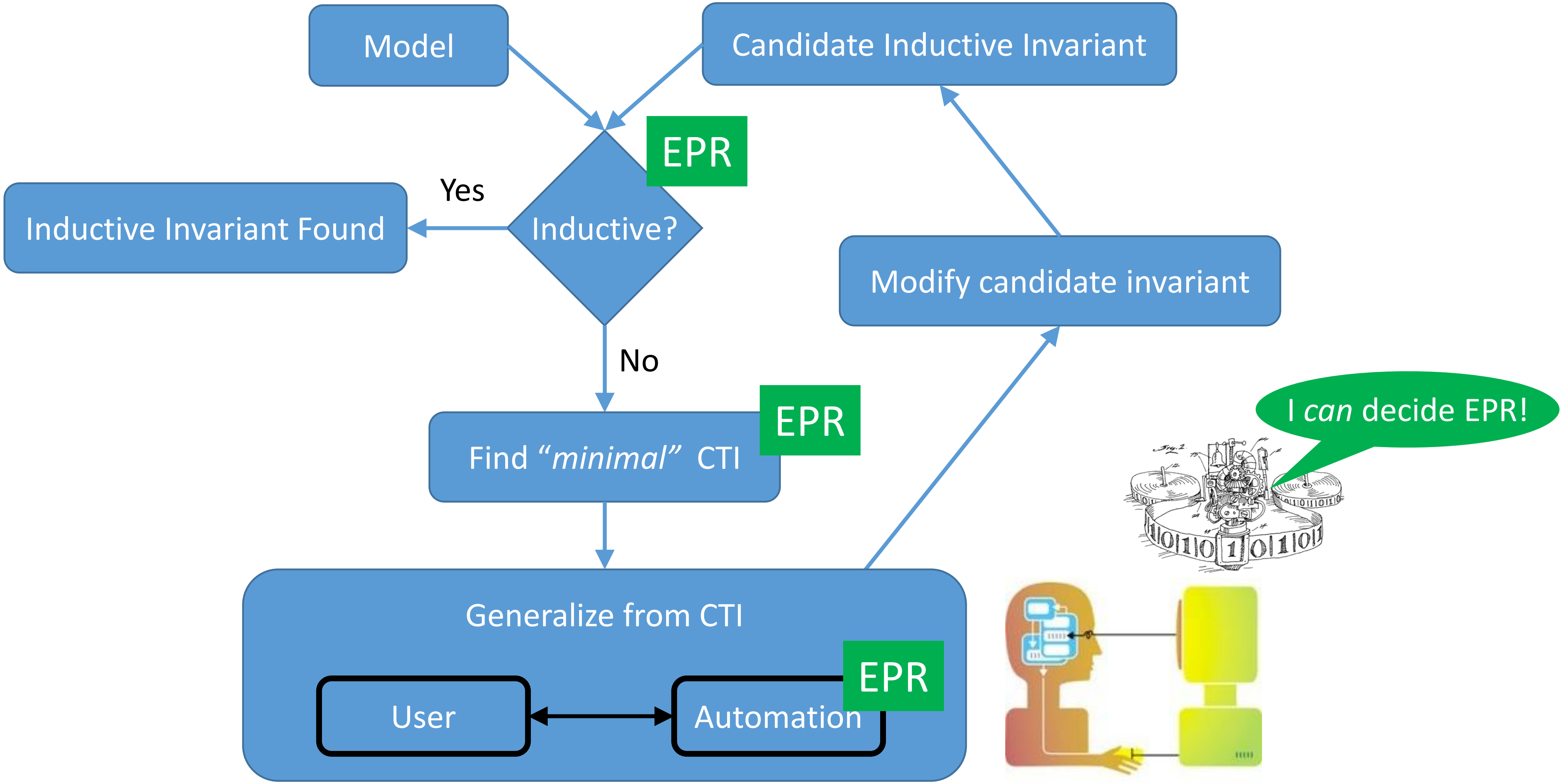
How can we find an inductive invariant without knowing it?



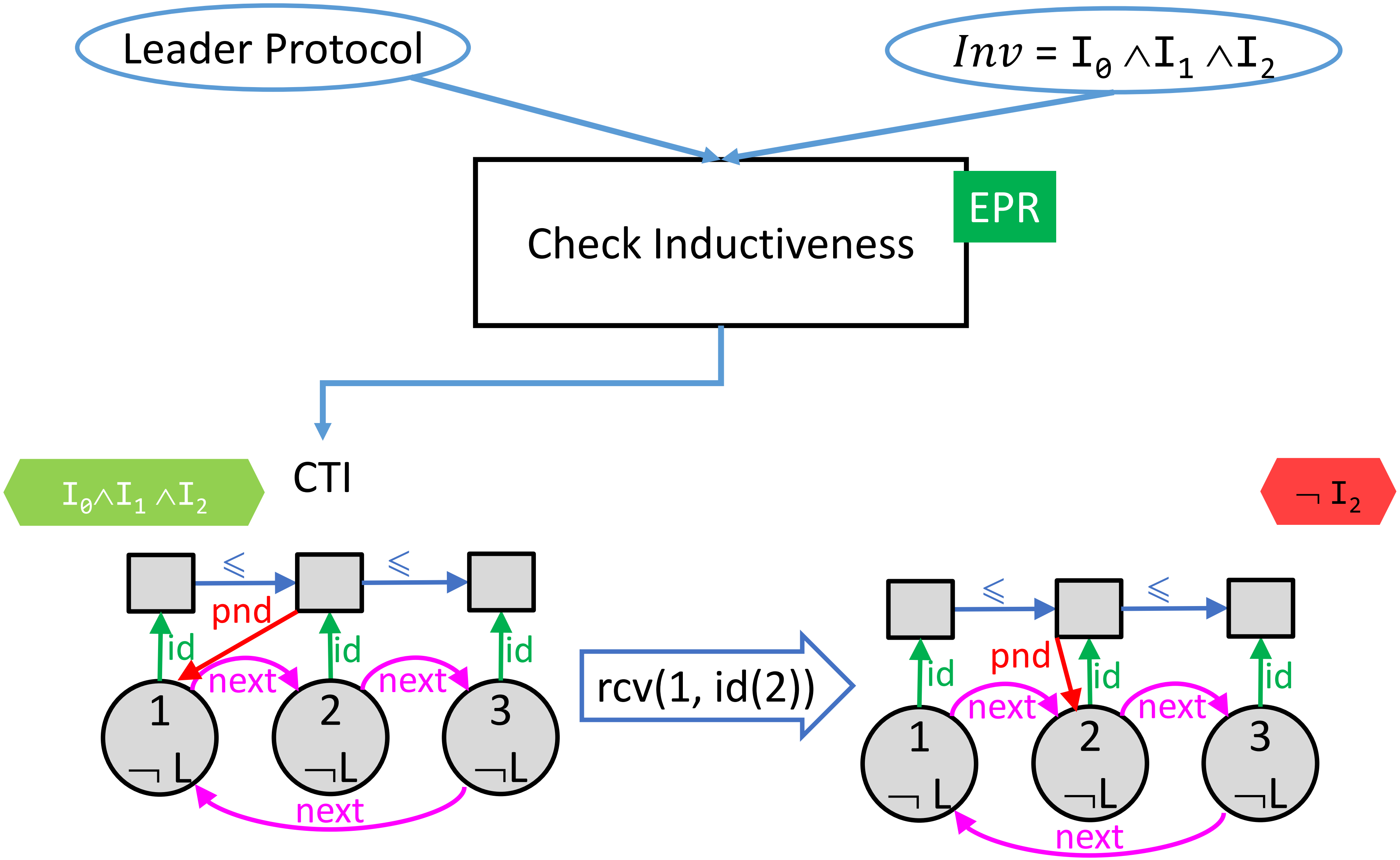
VC Generator → $Init(V) \land \neg Inv(V)$; $Inv(V) \land TR(V, V') \land \neg Inv(V')$; $Inv(V) \land Bad(V)$ → **EPR Solver** → Proof

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

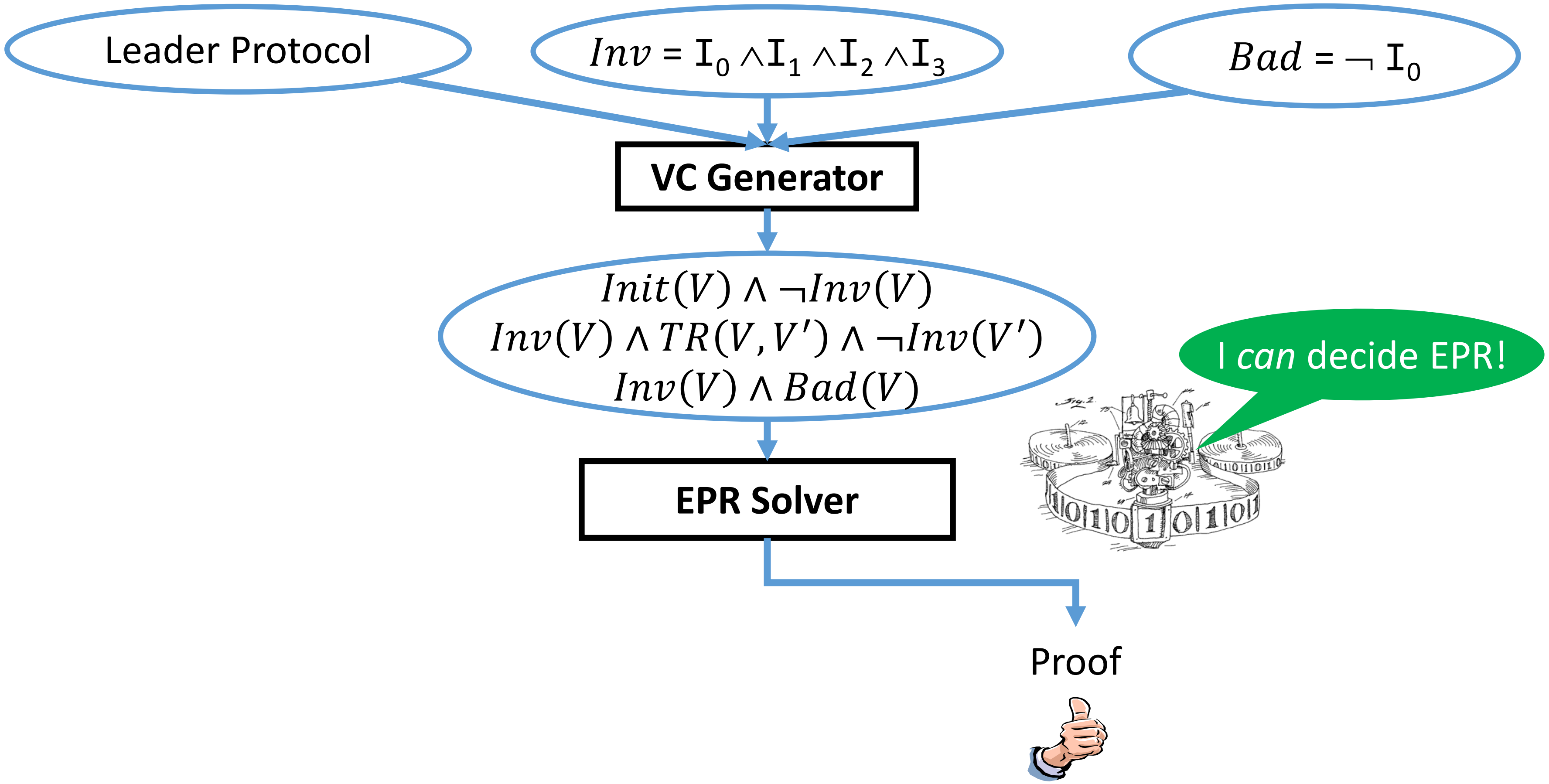
Interactive invariant inference [PLDI'16]



Ivy: check inductiveness

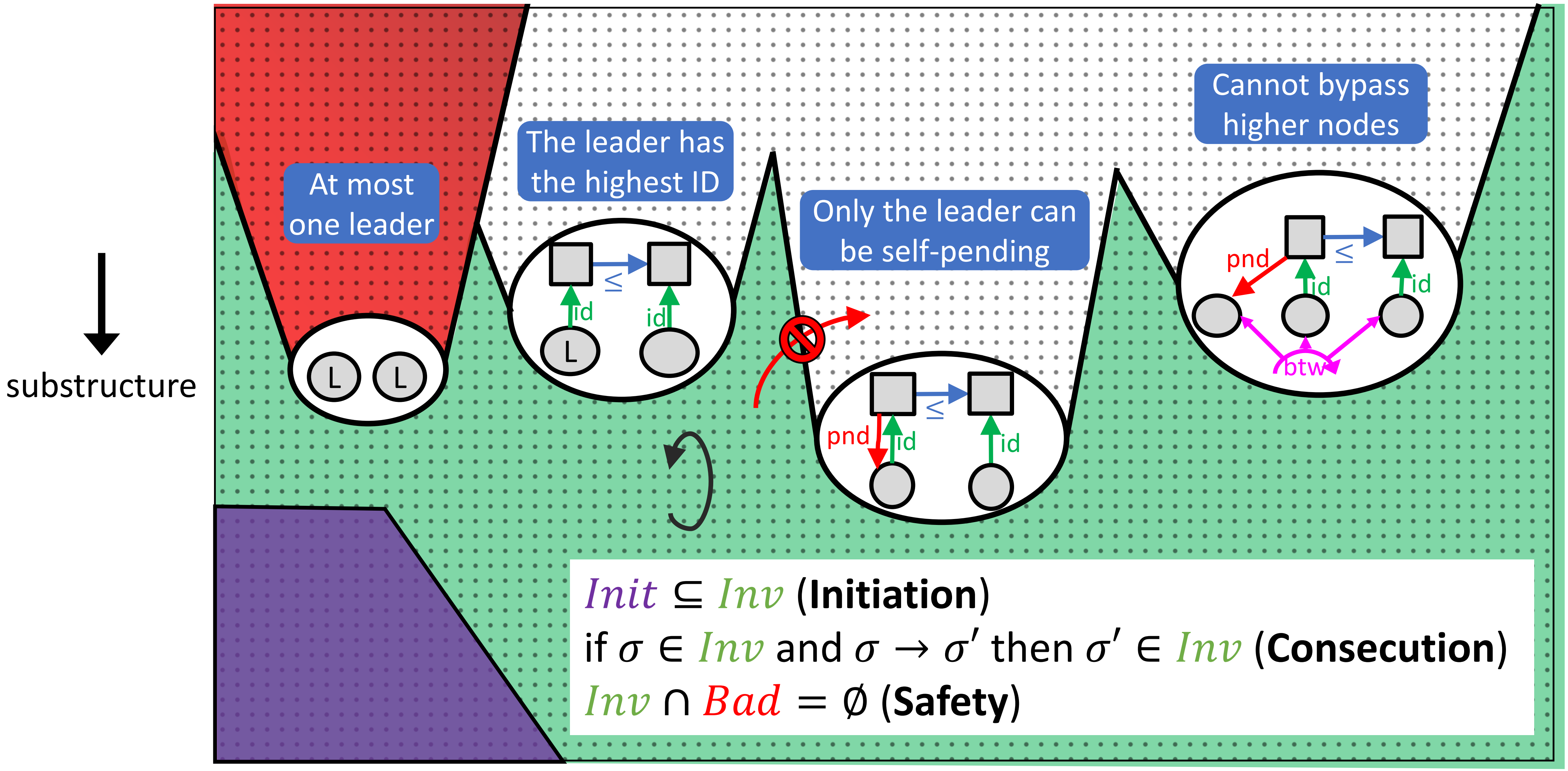


Ivy: check inductiveness



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

∀* invariant – excluded substructures



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

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

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

Principle: first-order abstractions/modularity

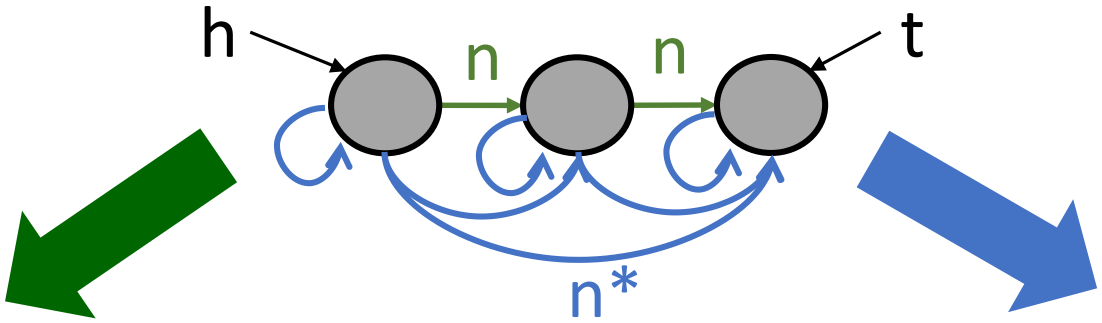
Concept	Intention	First-order abstraction
Node ID's	Integers	**function** id: Node → ID **relation** $\leq$(ID, ID) **axiom** $\forall x{:}ID.\ x \leq x$ *reflexive* **axiom** $\forall x,y,z{:}ID.\ x \leq y \land y \leq z \rightarrow x \leq z$ *transitive* **axiom** $\forall x,y{:}ID.\ x \leq y \land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

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

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

Key idea: representing deterministic paths

[Itzhaky SIGPLAN Dissertation Award 2016]



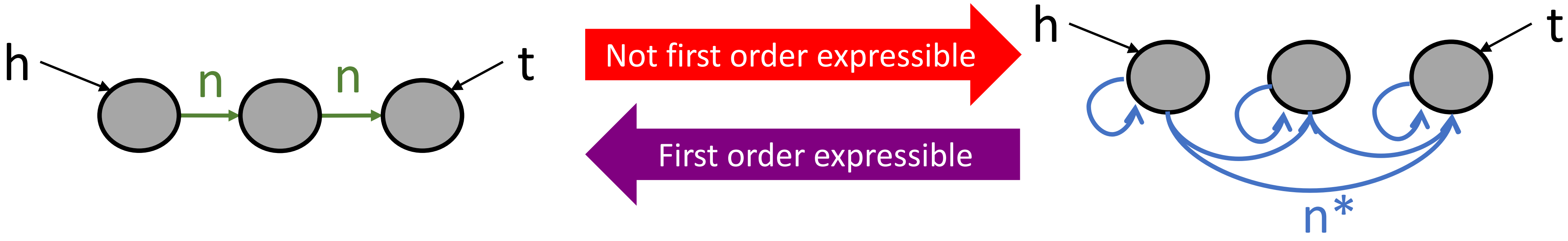
$n^* \approx btw$

Alternative 1: maintain n

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

Alternative 2: maintain n^*

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



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

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

Paxos

- Single decree Paxos – consensus
  lets nodes make a common decision despite node crashes and packet loss
- Paxos family of protocols – state machine replication
  variants for different tradeoffs, e.g., Fast Paxos is optimized for low contention, Vertical Paxos is reconfigurable, etc.
- Pervasive approach to fault-tolerant distributed computing
  - Google Chubby
  - VMware NSX
  - Amazon AWS
  - Many more…

Challenge: sets and cardinalities in FOL

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

  Can be modeled in first-order logic!
- Solution: axiomatize quorums in first-order logic

  sort Quorum

  relation member (Node, Quorum)

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

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

```
action propose(r:Round) {
 require “>N/2 join_msg’s”
 …
}
```

↓

```
action propose(r:Round) {
 require ∃q.∀n.member(n,q) →
   ∃r’,v’.join_msg(n,r,r’,v’)
 …
}
```

Principle: first-order abstractions

Concept	Intention	First-order abstraction
Quorums	Majority sets	**relation** member (Node, Quorum) **axiom** $\forall q_1,q_2$:Quorum$\exists$n:Node. member(n, q_1) $\land$ member(n, q_2)
Rounds	Natural numbers	**relation** $\leq$(Round, Round) **axiom** $\forall$x:Round. $x \leq x$ *reflexive* **axiom** $\forall$x,y,z:Round. $x \leq y \land y \leq z \rightarrow x \leq z$ *transitive* **axiom** $\forall$x,y:Round. $x \leq y \land y \leq x \rightarrow x = y$ *anti-symmetric* **axiom** $\forall$x,y:Round. $x \leq y \lor y \leq x$ *total*
Messages	Network with: 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)

Paxos in first-order logic

```
sort node, quorum, round, value

relation ≤ : round, round
axiom total_order(≤)
constant ⊥ : round

relation member : node, quorum
axiom ∀q1, q2 : quorum. ∃n : node. member(n, q1) ∧ member(n, q2)

relation start_round_msg : round
relation join_ack_msg : node, round, round, value
relation propose_msg : round, value
relation vote_msg : node, round, value
relation decision : node, round, value

init ∀r. ¬start_round_msg(r)
init ∀n, r1, r2, v. ¬join_ack_msg(n, r1, r2, v)
init ∀r, v. ¬propose_msg(r, v)
init ∀n, r, v. ¬vote_msg(n, r, v)
init ∀n, r, v. ¬decision(n, r, v)

action START_ROUND(r : round) {
  assume r ≠ ⊥
  start_round_msg(r) := true
}
action JOIN_ROUND(n : node, r : round) {
  assume r ≠ ⊥
  assume start_round_msg(r)
  assume ¬∃r′, r″, v. r′ > r ∧ join_ack_msg(n, r′, r″, v)
  # find maximal round in which n voted, and the corresponding vote.
  # maxr = ⊥ and v is arbitrary when n never voted.
  local maxr, v := max {(r′, v′) | vote_msg(n, r′, v′) ∧ r′ < r}
  join_ack_msg(n, r, maxr, v) := true
}
action PROPOSE(r : round, q : quorum) {
  assume r ≠ ⊥
  assume ∀v. ¬propose_msg(r, v)
  # 1b from quorum q
  assume ∀n. member(n, q) → ∃r′, v. join_ack_msg(n, r, r′, v)
  # find the maximal round in which a node in the quorum reported
  # voting, and the corresponding vote.
  # v is arbitrary if the nodes reported not voting.
  local maxr, v := max {(r′, v′) | ∃n. member(n, q)
                              ∧ join_ack_msg(n, r, r′, v′) ∧ r′ ≠ ⊥}
  propose_msg(r, v) := true   # propose value v
}
action VOTE(n : node, r : round, v : value) {
  assume r ≠ ⊥
  assume propose_msg(r, v)
  assume ¬∃r′, r″, v. r′ > r ∧ join_ack_msg(n, r′, r″, v)
  vote_msg(n, r, v) := true
}
action LEARN(n : node, r : round, v : value, q : quorum) {
  assume r ≠ ⊥
  # 2b from quorum q
  assume ∀n. member(n, q) → vote_msg(n, r, v)
  decision(n, r, v) := true
}
```

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

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

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

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

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

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

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

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

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

VC’s in first-order logic

Quantifier alternation cycles

- Axiom

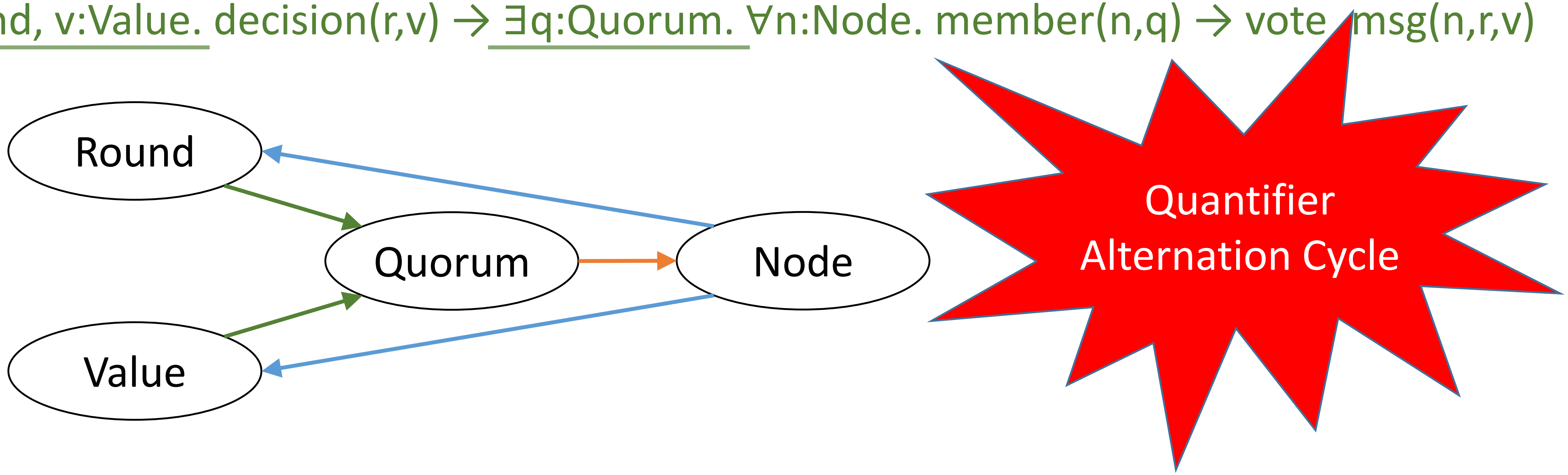
  $\forall q_1,q_2: Quorum.\ \exists n: Node.\ member(n, q_1) \land member(n, q_2)$

- Propose action precondition

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

- Inductive invariant

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



Paxos made EPR [OOPSLA'17]

Methodology for decidable verification of infinite-state systems

Modeling | Transforming

Protocol → 1 → Formal specification *in first-order logic* → 2 → Formal specification with *decidable VC*

First-order abstractions
Domain knowledge

I *can* decide EPR!

Z3

Inductive invariant of Paxos

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

Paxos made EPR: experimental evaluation

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

*first mechanized verification

Transformation to EPR reusable across all variants!

Paxos made EPR: experimental evaluation

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

Proof / code ratio:

IronFleet: ~4

Verdi: ~10

Ivy: ~0.2

*first mechanized verification

Transformation to EPR reusable across all variants!

Paxos made EPR: experimental evaluation

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

$\boldsymbol{\mu}$ – mean

$\boldsymbol{\sigma}$ – std. deviation

*first mechanized verification

Transformation to EPR reusable across all variants!

Paxos made EPR: experimental evaluation

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

Rounds	FOL [sec] μ	FOL [sec] σ	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!

Paxos made EPR: experimental evaluation

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

Rounds	FOL [sec] μ	FOL [sec] σ	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!

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.

⟨1⟩7. $NoneChoosableAfter(i, b, v)'$

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

⟨2⟩1. CASE: $sval2a(i, b, Q) = \top$

PROOF: Assumption ⟨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 $sval2a(j, b, Q) = \top$. The assumption $c < b$ and step ⟨1⟩4 then imply $NotChoosable(j, c, w)$.

⟨2⟩2. CASE: $sval2a(i, b, Q) \neq \top$

⟨3⟩1. $sval2a(i, b, Q) = val2a(i, b, Q) = v$

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

⟨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 $sval2a(i, b, Q) \in StopCmd$. Case assumption ⟨2⟩2 and the definition of $sval2a$ then imply $mbal2a(k, b, Q) < mbal2a(i, b, Q)$ for all $k > i$.

⟨4⟩2. $NotChoosable(j, c, w)$

PROOF: ⟨4⟩1 and case assumption ⟨3⟩4 imply $mbal2a(j, b, Q) < c < b$. By assumption ⟨1⟩1.4, Lemma 3 implies $NotChoosable(j, c, w)$. □

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

- Expressing transitive closure
  - Linked lists
  - Ring protocols
- Expressing sets and cardinalities
  - Paxos and its variants
  - Byzantine Fault Tolerance
  - Reconfiguration
- Liveness and temporal properties [POPL'18]

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

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

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

* First mechanized liveness proof

Summary

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