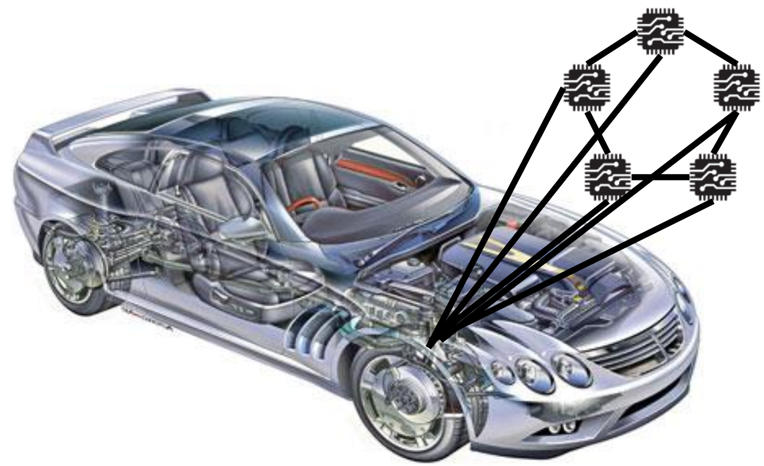
Interactive Verification of Distributed Protocols

Sharon Shoham

Tel Aviv University

erc Supervised Verification of Infinite-State Systems

Why Verify Distributed Protocols?

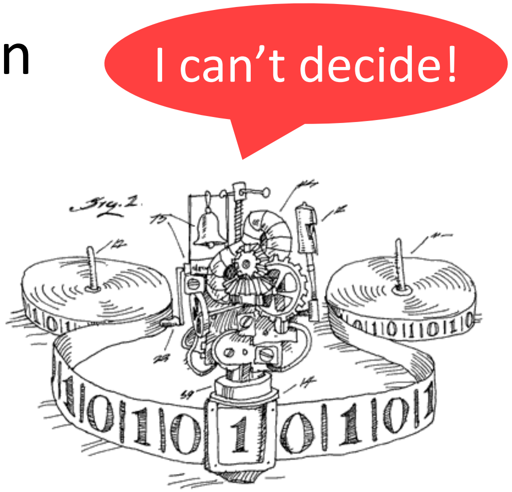


- Distributed systems are everywhere
  - e.g., safety-critical systems
- Distributed systems are notoriously hard to get right
- Testing is costly and not sufficient
  - Bugs occur on rare scenarios
  - Testing covers tiny fraction of behaviors
  - Leaves most bugs for production
  - Amazon employs TLA+ for testing protocols, but scaling is an issue

Verifying Distributed Protocols is Hard

- Infinite state-space
  - unbounded number of objects
  - unbounded number of threads
  - unbounded number of messages

- Asymptotic complexity of program verification
  - The halting problem
  - Rice theorem
  - The ability of simple programs to represent complex behaviors



State of the art in formal verification

- Automatic techniques
  - Model checking
    - Exploit finite state / finite abstraction
  - Abstract Interpretation
    - Sound abstraction
  - Limited for infinite state systems due to undecidability
- Deductive techniques
  - Use SMT for deduction with manual program annotations (e.g. Dafny)
    - Requires programmer effort to provide inductive invariants
    - SMT solver may diverge (matching loops, arithmetic)
  - Interactive theorem provers (e.g. Coq, Isabelle/HOL, LEAN)
    - Programmer gives inductive invariant and proves it
    - Huge programmer effort (~10-50 lines of proof per line of code)

State of the art in formal verification

Expressiveness

Proof Assistants

Ultimately limited by human

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

Automation

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

State of the art in formal verification

Expressiveness

Proof Assistants

Ultimately limited by human

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

Ultimately limited by undecidability

Model Checking
Static Analysis

Automation

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

State of the art in formal verification

Expressiveness

Proof Assistants

Ultimately limited by human

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

Supervised Verification

proof/code: IVy ~1/10

Ultimately limited by undecidability

Model Checking
Static Analysis

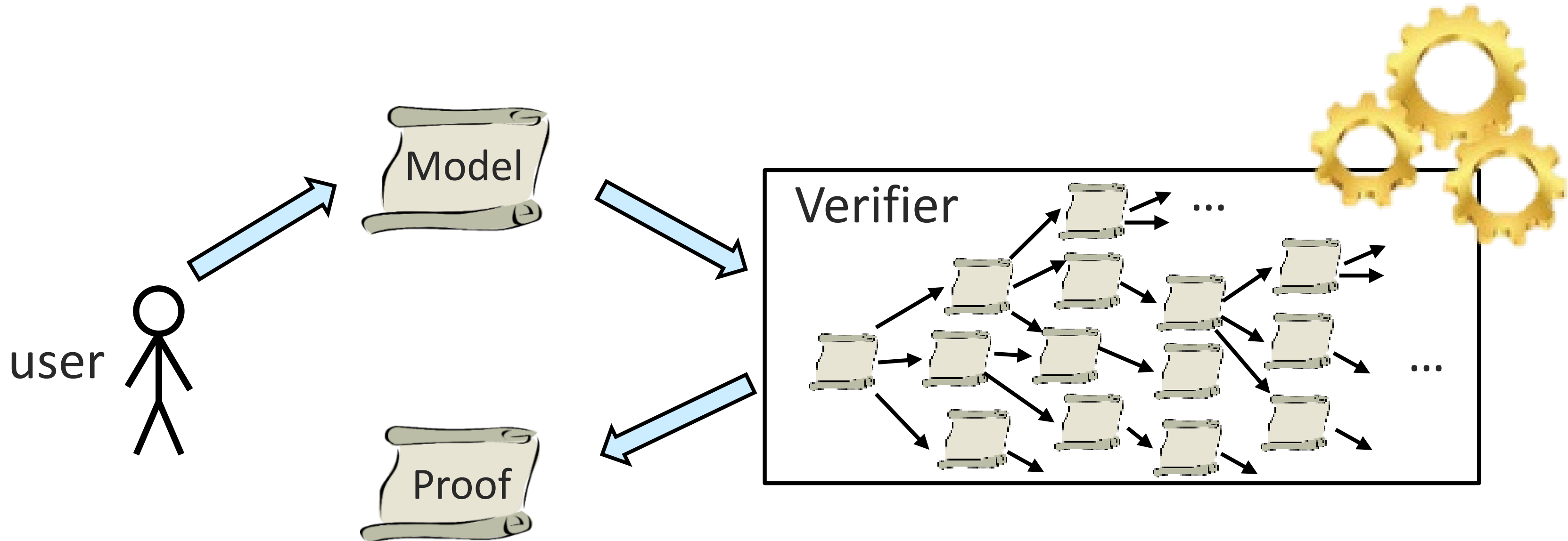
Automation

Supervised Verification of Infinite-State Systems

IVy: Verified Protocols

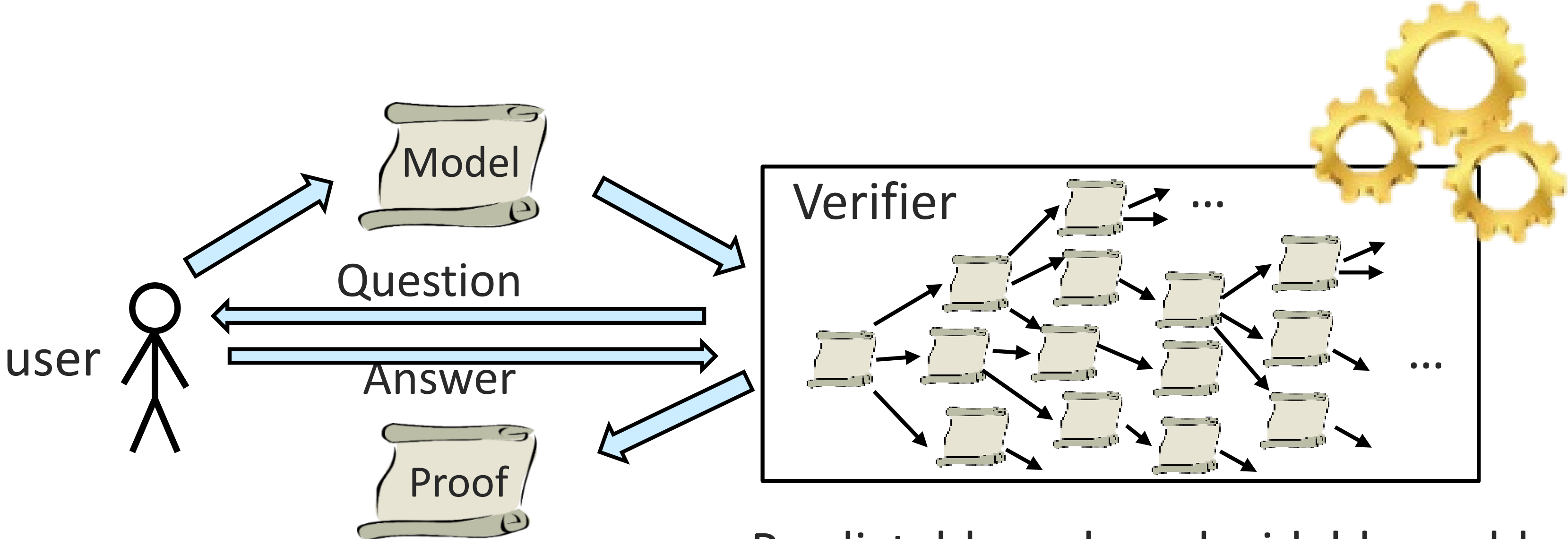
Protocol	Model (# LOC)	Property (# Literals)	Invariant (# Literals)
Leader in Ring	59	3	12
Learning Switch	50	11	18
DB Chain Replication	143	11	35
Chord	155	35	46
Lock Server (500 Coq lines [Verdi])	122	3	21
Distributed Lock (1 week [IronFleet])	41	3	26
Single Decree Paxos	85	3	32
Multi Paxos	102	3	38
Vertical Paxos	123	3	65
Fast Paxos	117	3	59
Flexible Paxos	88	3	32
Stoppable Paxos	130	6	60
Virtually Synchronous Paxos	Work in progress		

Automatic Verification



- Unpredictable, often diverges
- Restricted expressivity

Supervised Verification



- Predictable: solves decidable problems
- High expressivity

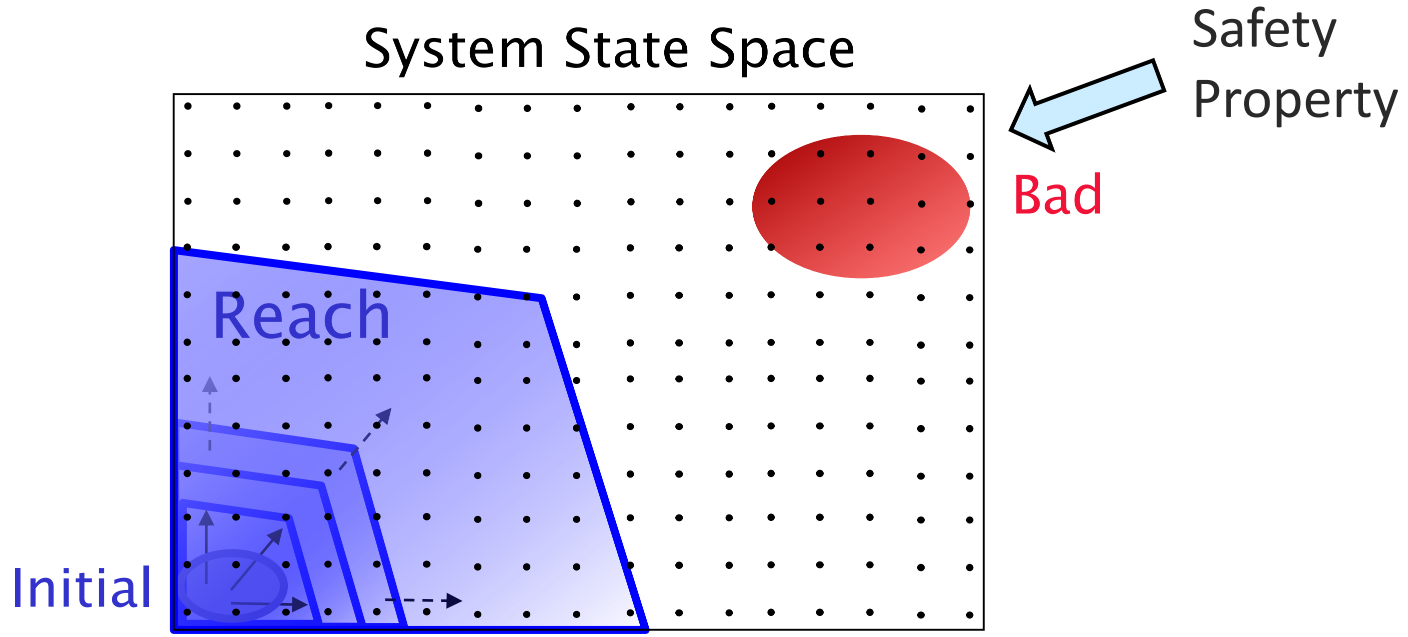
- How to divide the problem between the human and the machine?
- How to conduct the interaction?

IVy

IVy: https://github.com/Microsoft/ivy

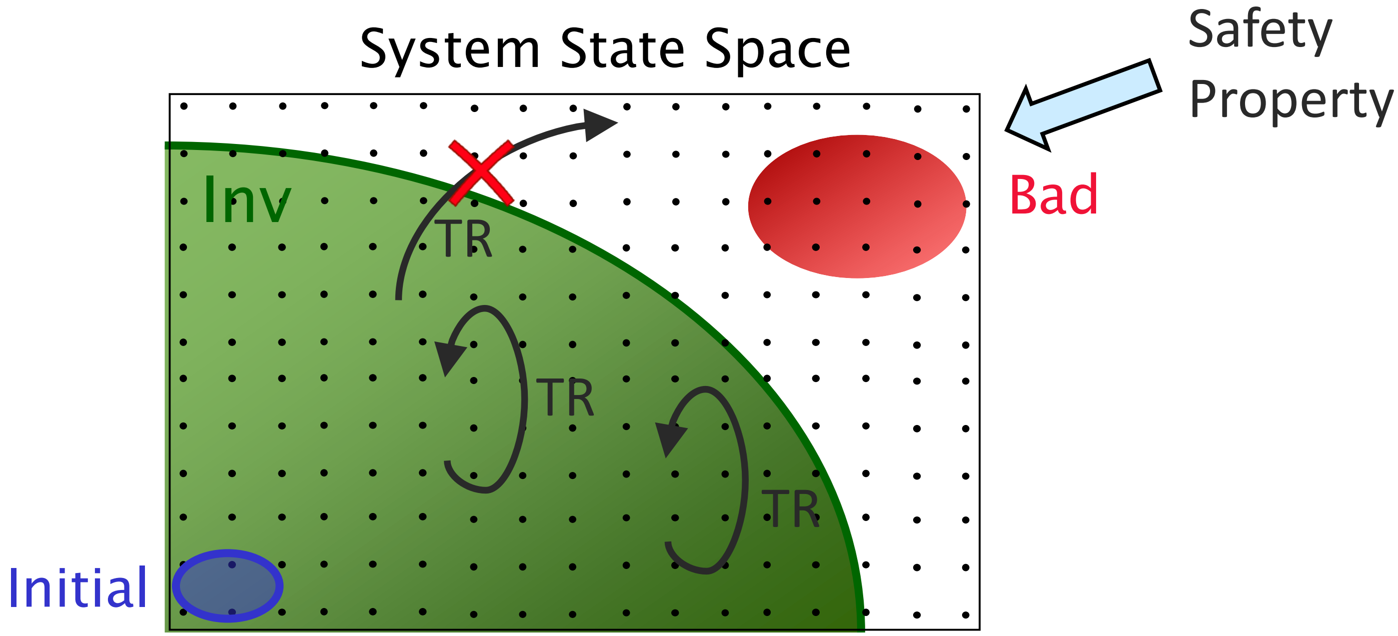
- [PLDI'16] IVy: Safety Verification by Interactive Generalization. O. Padon, K. McMillan, A. Panda, M. Sagiv, S. Shoham
- [OOPSLA'17] Paxos Made EPR: Decidable Reasoning about Distributed Protocols. O. Padon, G. Losa, M. Sagiv, S. Shoham
- [POPL'18] Reducing Liveness to Safety in First-Order Logic. O. Padon, J. Hoenicke, G. Losa, A. Podelski, M. Sagiv, S. Shoham

Safety Verification



System S is **safe** if all the reachable states satisfy the property P = ¬Bad

Safety Verification

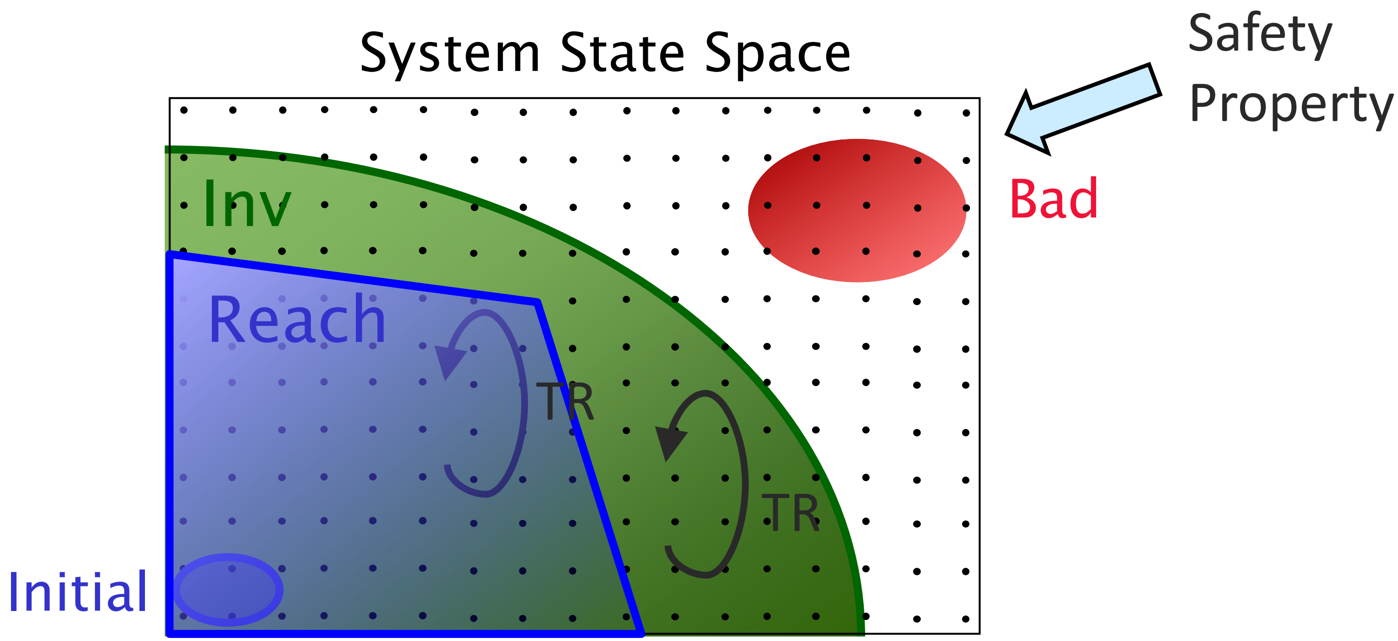


System S is **safe** if all the reachable states satisfy the property P = ¬Bad

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

Inv ⇒ P=¬Bad	(Safety)
Init ⇒ Inv	(Initiation)
if σ ⊨ Inv and TR(σ, σ') then σ' ⊨ Inv	(Consecution)

Safety Verification



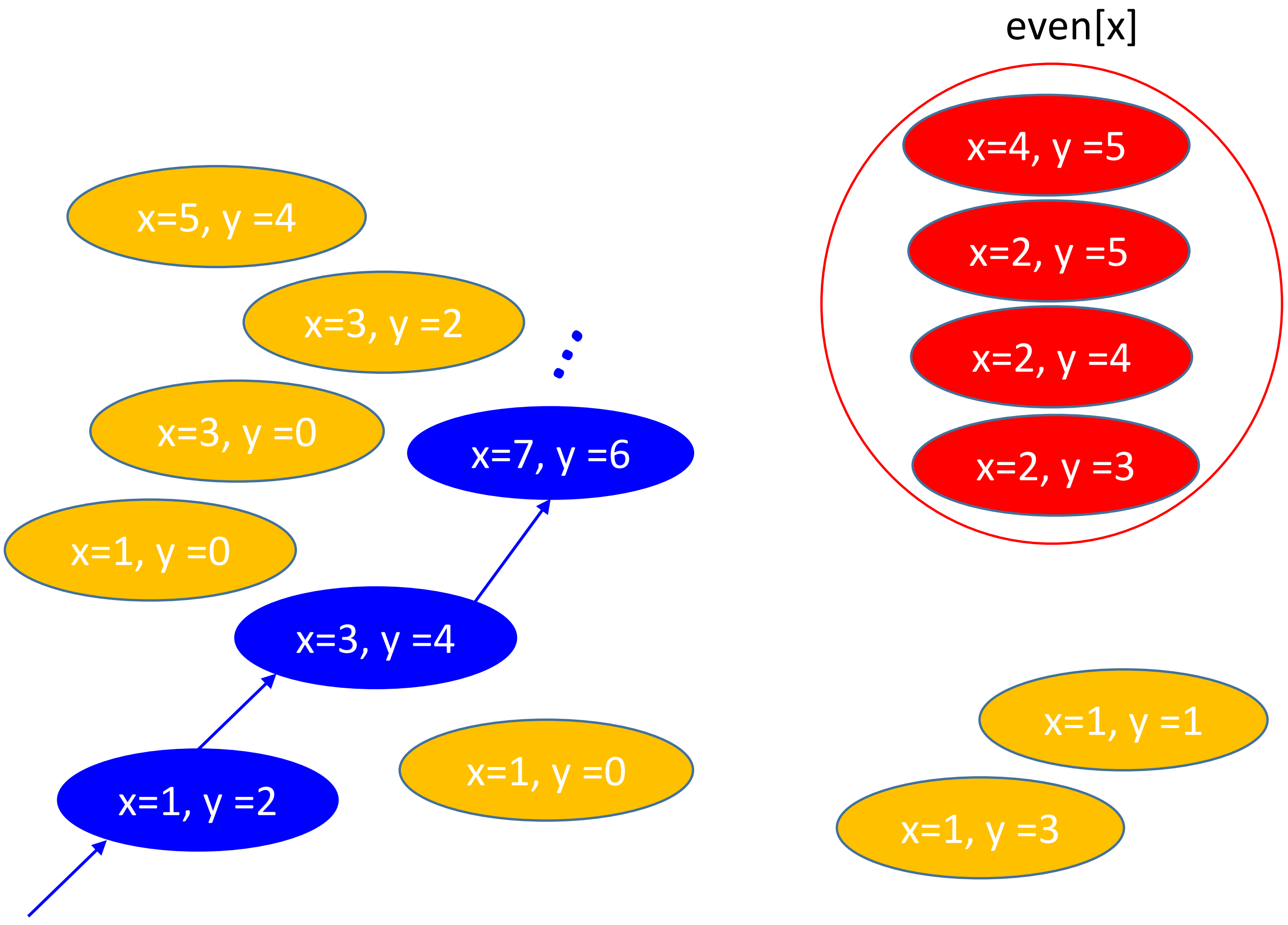
System S is **safe** if all the reachable states satisfy the property P = ¬Bad

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

$Inv \Rightarrow P = \neg Bad$	(Safety)
$Init \Rightarrow Inv$	(Initiation)
if $\sigma \models Inv$ and $TR(\sigma, \sigma')$ then $\sigma' \models Inv$	(Consecution)

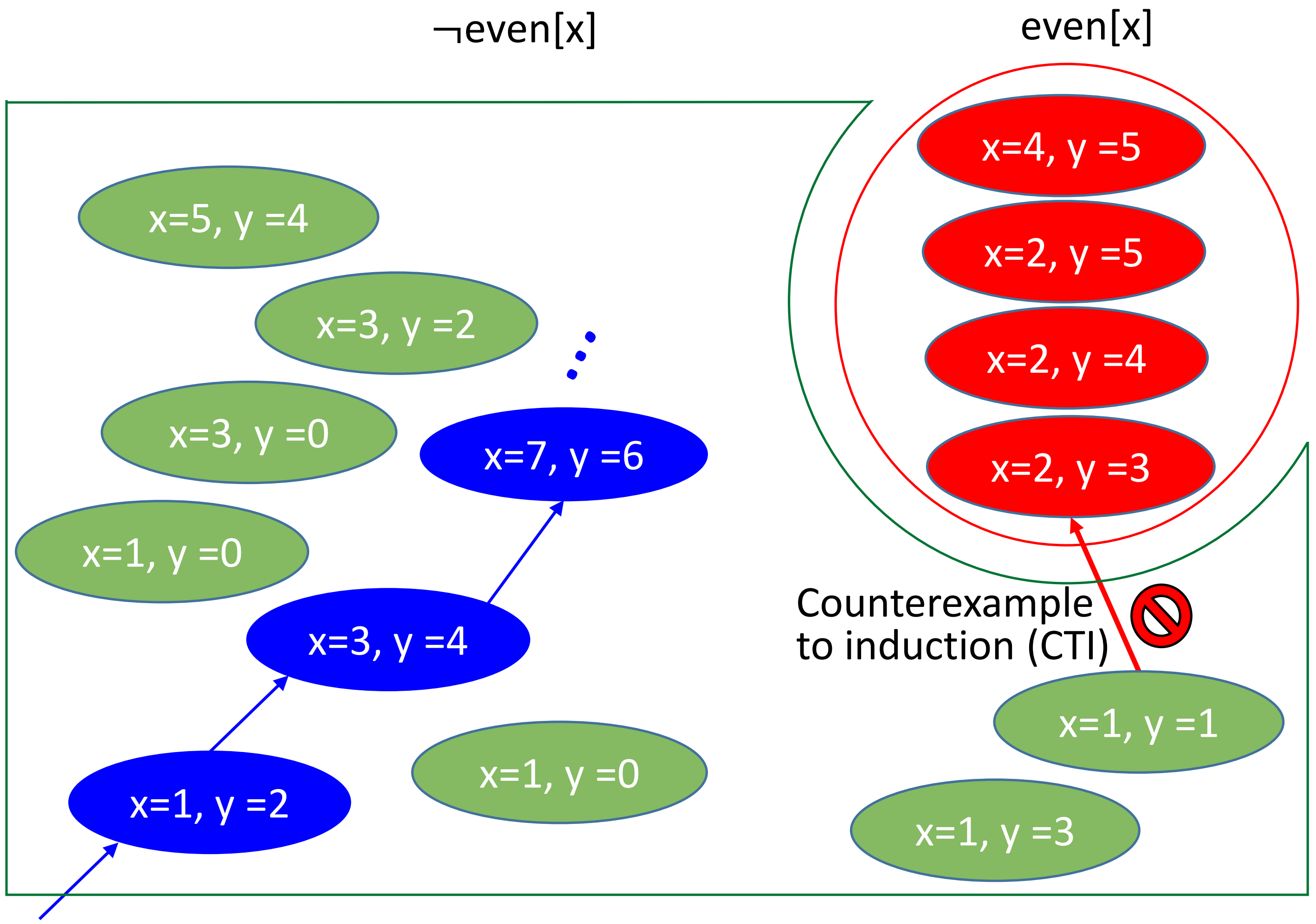
Simple Example: Loop Invariants

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



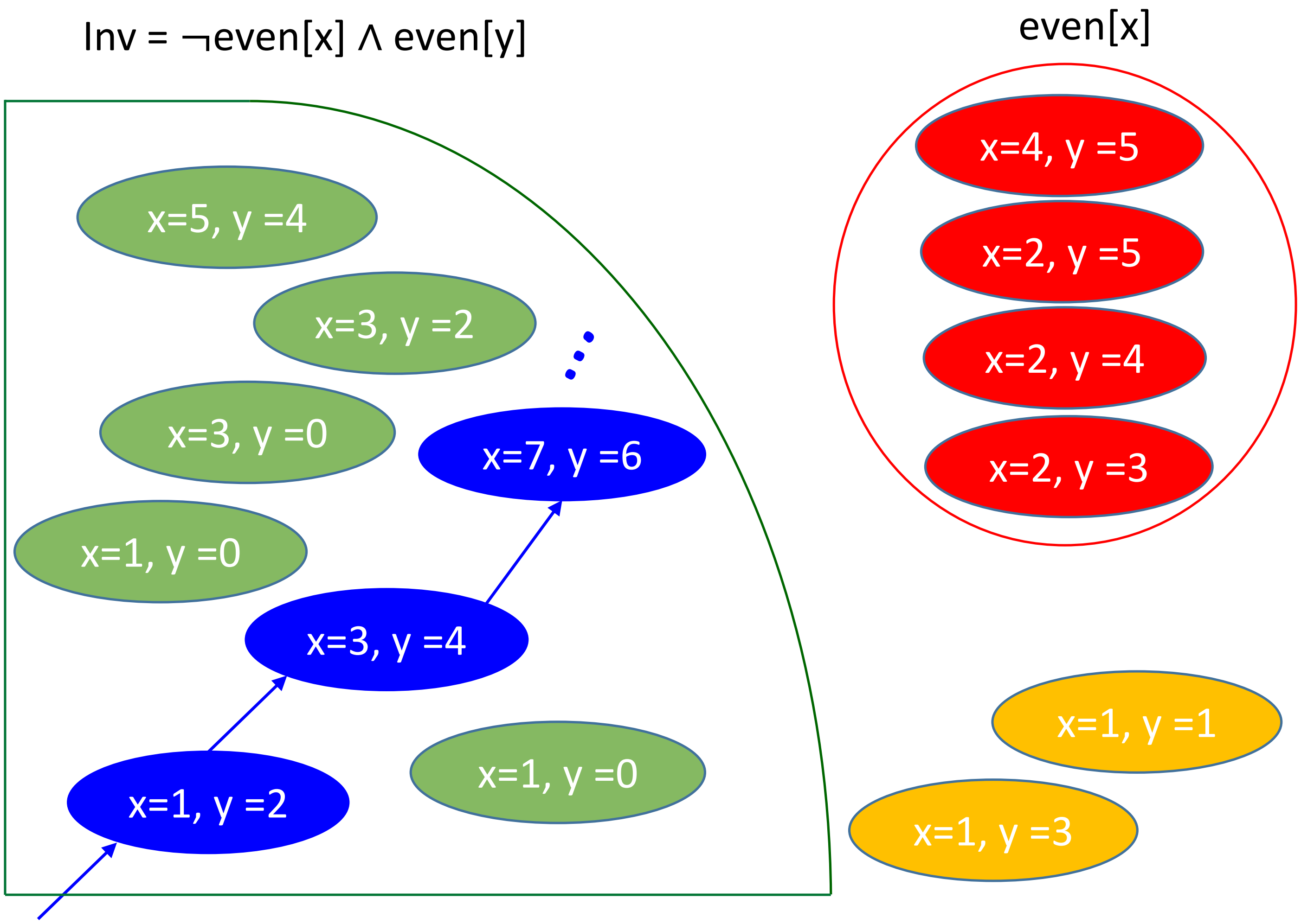
Simple Example: Loop Invariants

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



Simple Example: Loop Invariants

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



Challenges in Safety Verification

Infer inductive invariants for safety verification

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

IVy's Approach: Supervised Verification

Infer inductive invariants for safety verification

User

1. Formal specification: reasoning about infinite-state systems
   - Modeling the system and the property (TR, Init, Bad)

Machine

2. Deduction: checking inductiveness
   - Undecidability of implication checking
     - Unbounded state (threads, messages), arithmetic, quantifiers,…

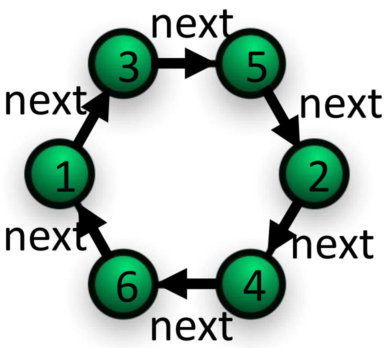
User + Machine

3. Inference: inferring inductive invariants (Inv)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable

How Does it Work?

- Specify systems and properties in decidable fragment of first-order logic (EPR)
  - Allows quantifiers to reason about unbounded sets
  - Decidable to check inductiveness
  - Finite counterexamples to induction, display graphically
- Interact with the user to find inductive invariants
  - by providing graphical UI for gradually strengthening the inductive invariant based on counterexamples to induction
- Logic is mostly hidden
  - Friendly to non-expert users

Example: Leader Election in a Ring



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

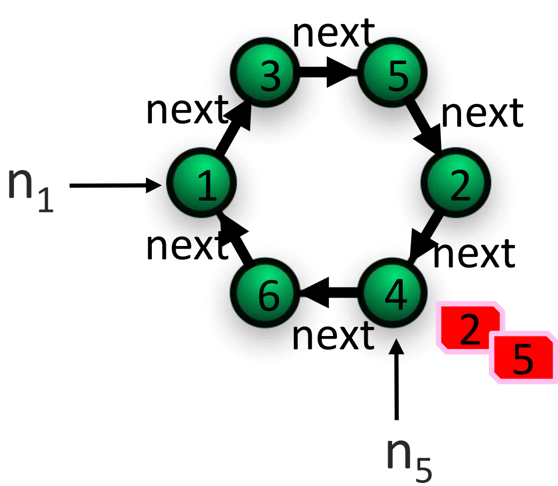
[CACM'79] E. Chang and R. Roberts. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*

Modeling in IVy

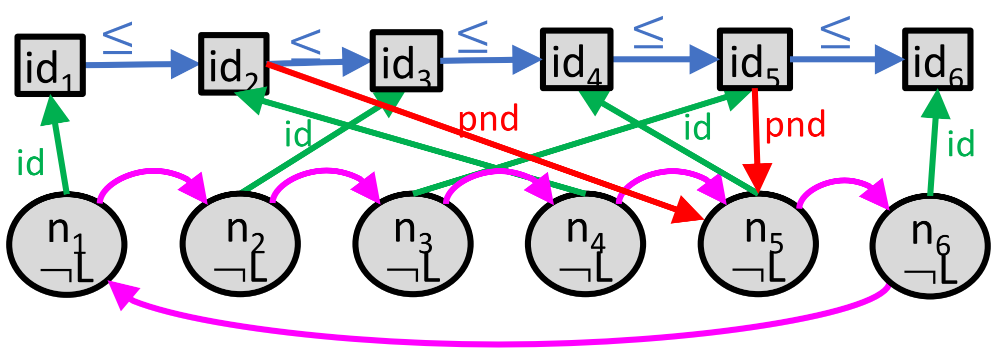
- State: first-order structure over vocabulary V
  - ≤ (ID, ID) – total order on node id's
  - **btw** (Node, Node, Node) – the ring topology
  - **id**: Node → ID – relate a node to its id
  - **pending**(ID, Node) – pending messages
  - **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



structure



$\langle n_5, n_1, n_3\rangle \in I($**btw**$)$

Modeling in IVy

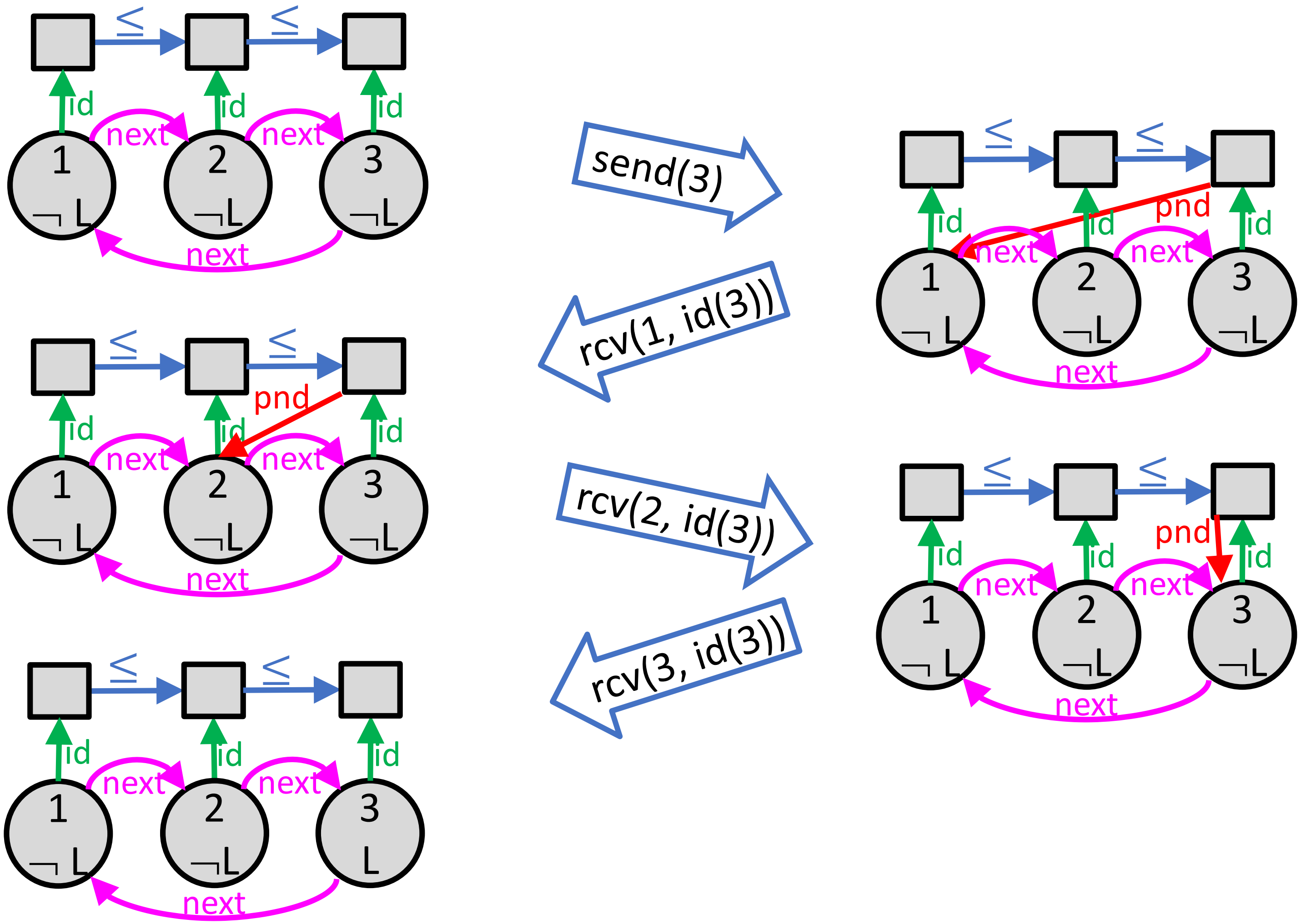
- ≤(ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending msgs
- **leader**(Node) – node is a leader

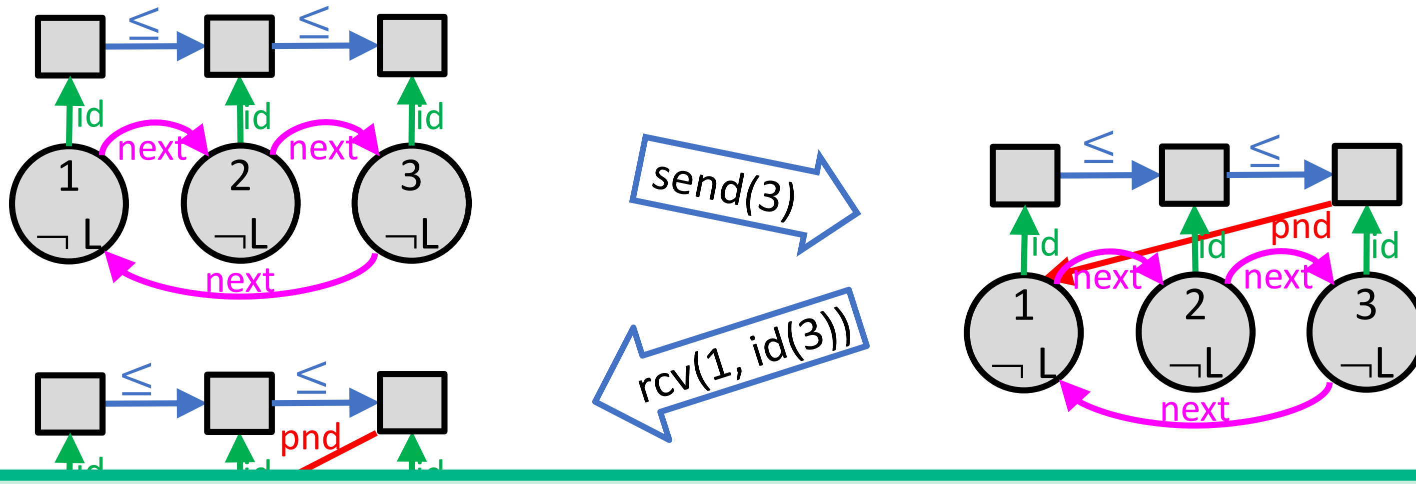
```
action send(n:Node) {
  "s := next(n)"
  pending(id(n),s) := true
}
```

```
action receive(n:Node, m:ID){
  requires pending(m, n)
  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
}
```

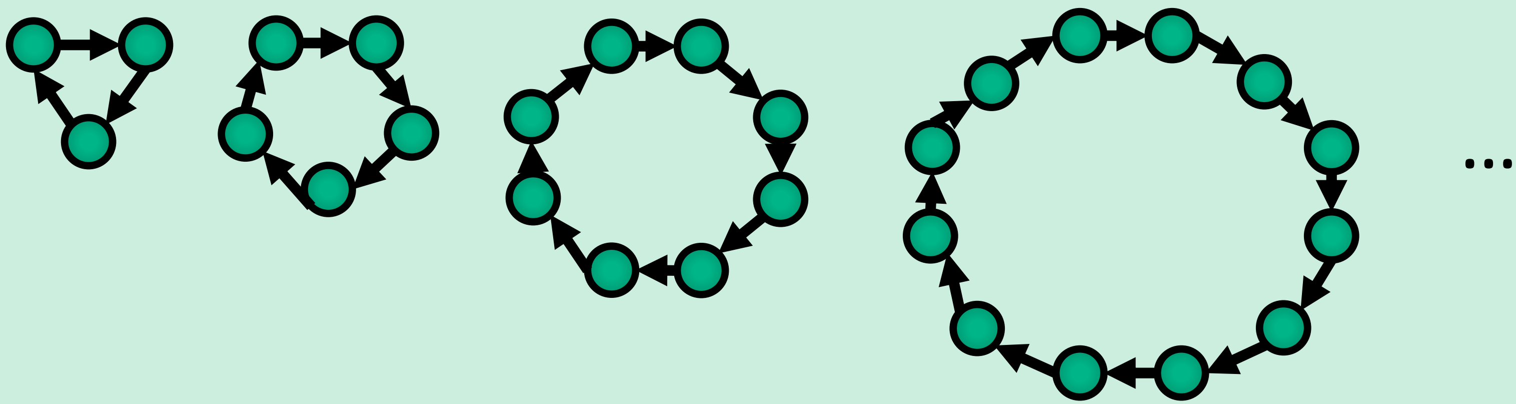
```
protocol = (send | receive)*
```

```
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))
pnd
id
next
≤
1
2
3
¬L
L

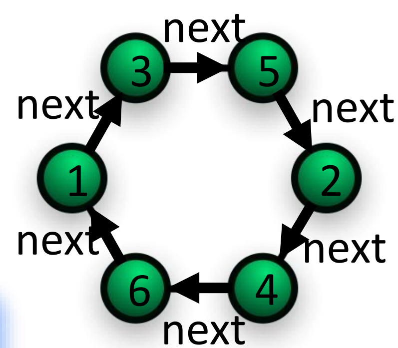


Specify and verify the protocol for **any** number of nodes in the ring



Example: Leader Election in a Ring

- Nodes are organized in a unidirectional ring
- Each node has a unique numeric id
- Protocol:
  - E
  - A next) if the id in t
  - A ler
- The
  - The protocol selects at most one leader



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*

Inductive Invariant for Leader Election

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

Safety property:

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

Inductive invariant: Inv = I0 ∧ I1 ∧ I2 ∧ I3

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

The leader has the highest ID

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

Only highest id can be self-pnd

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

Cannot bypass higher nodes

Inductive Invariant for Leader Election

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

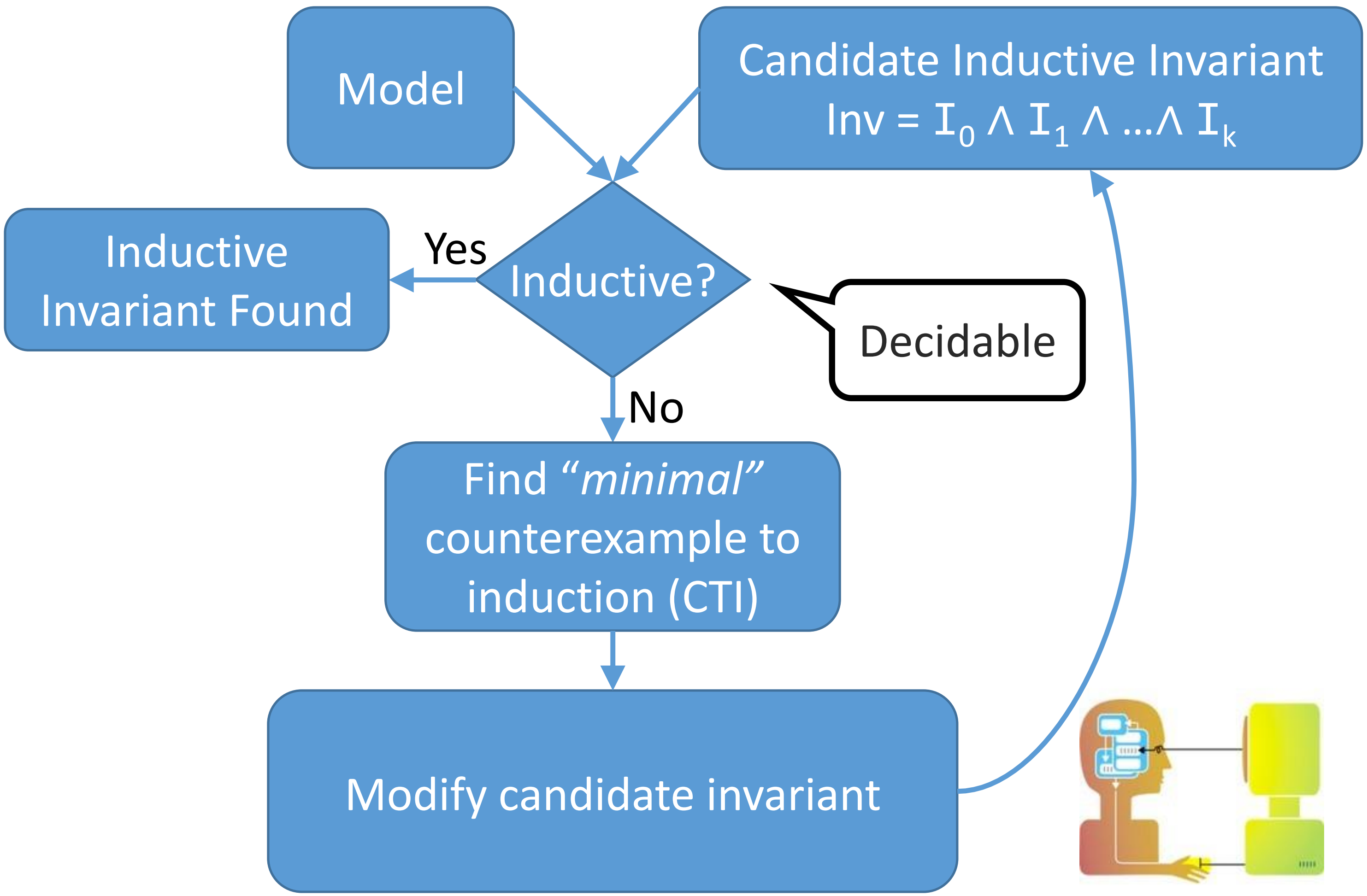
Safety property:

$I_0 = \neg Bad = \forall x,y: Node.\ \mathbf{leader}(x) \land \mathbf{leader}(y) \rightarrow x = y$

Inductive invariant: $Inv = I0 \land I1 \land I2 \land I3$

How can we come up with an inductive invariant?

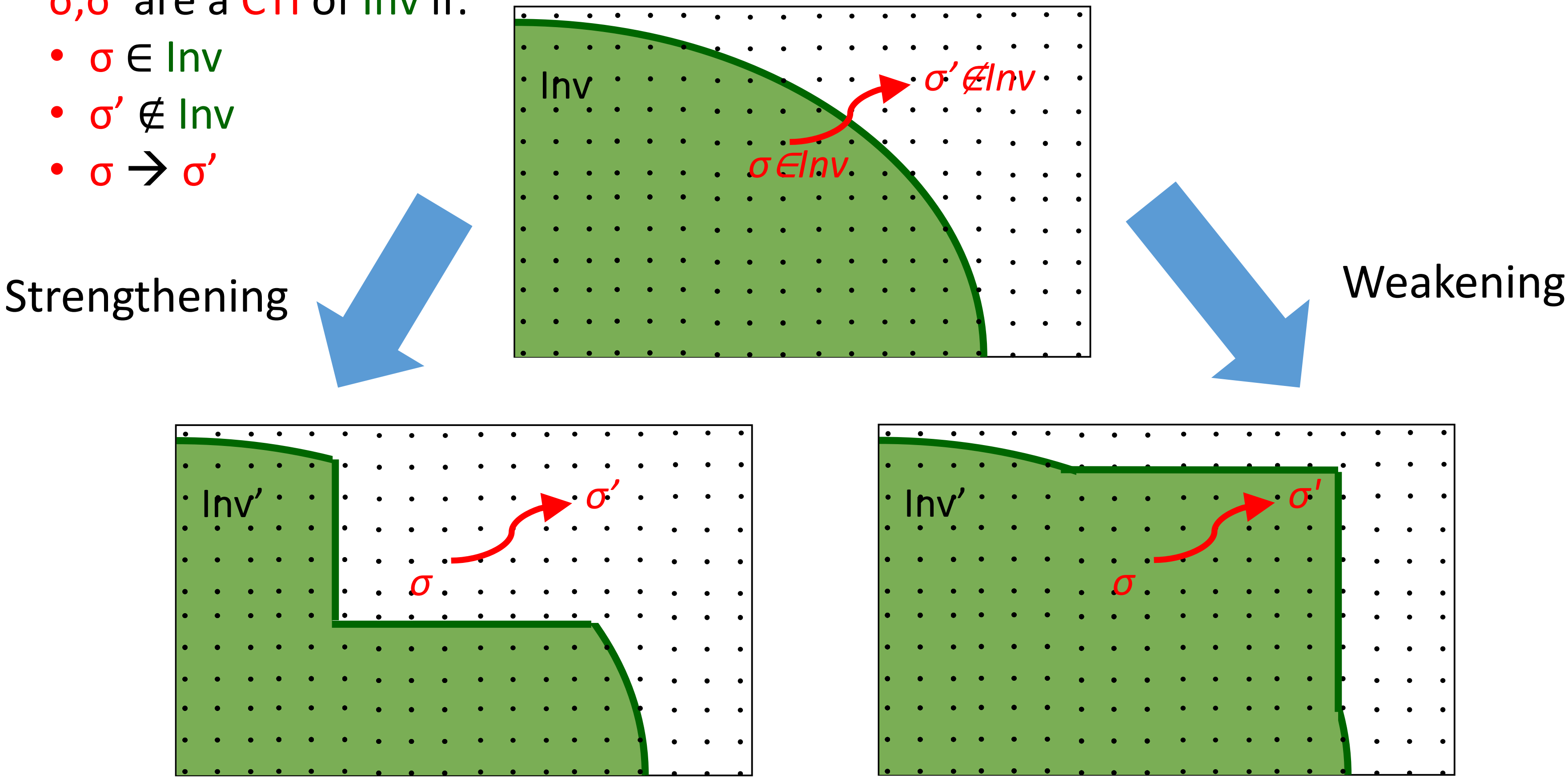
Invariant Inference in IVy



Strengthening & Weakening from CTI

σ,σ’ are a CTI of Inv if:

- σ ∈ Inv
- σ’ ∉ Inv
- σ → σ’



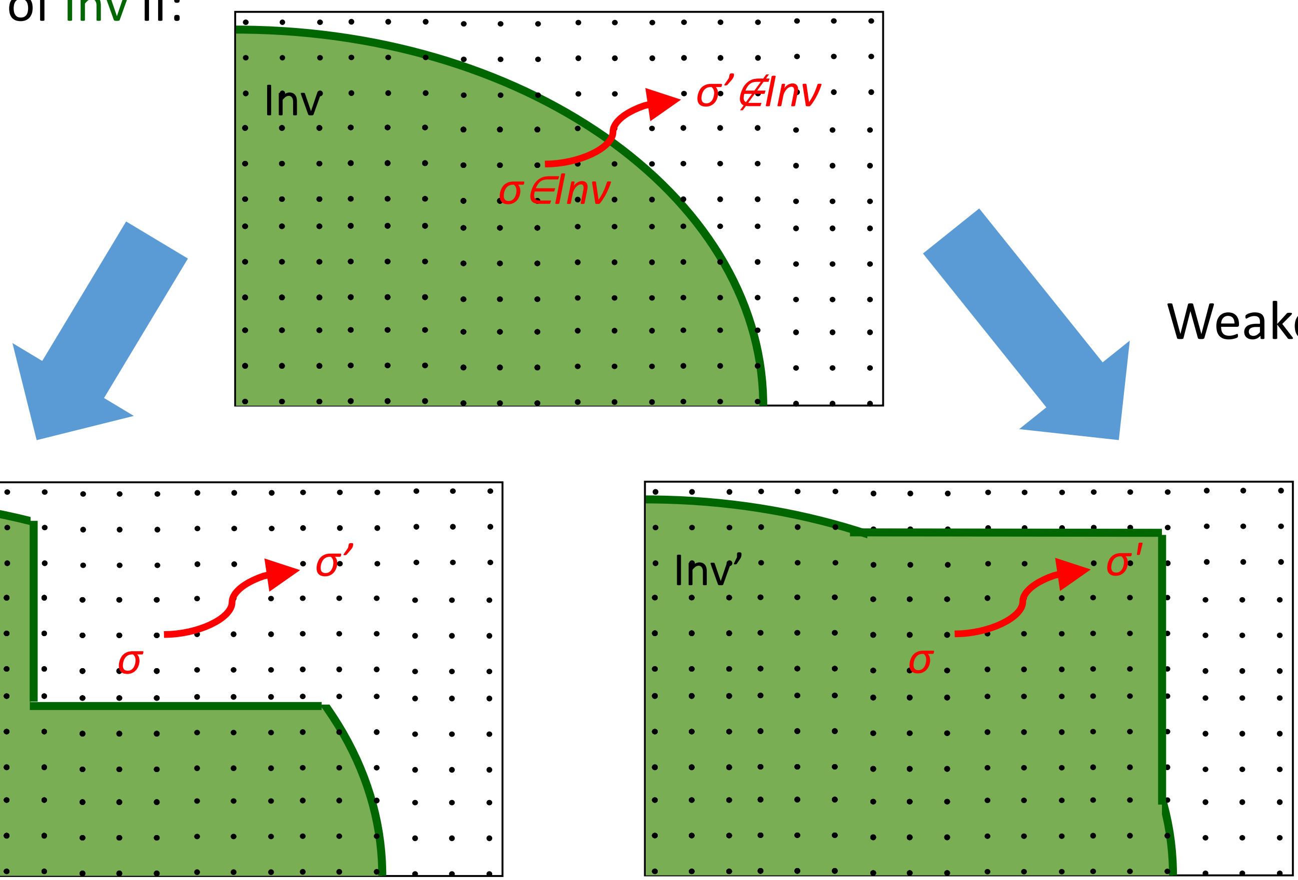
Strengthening & Weakening from CTI

σ,σ’ are a CTI of Inv if:

- σ ∈ Inv
- σ’ ∉ Inv
- σ → σ’

Strengthening

Weakening



Add a conjecture
Inv’ := Inv ∧ “avoid(σ)”

Key Challenge: Generalization

Generalization using Diagram

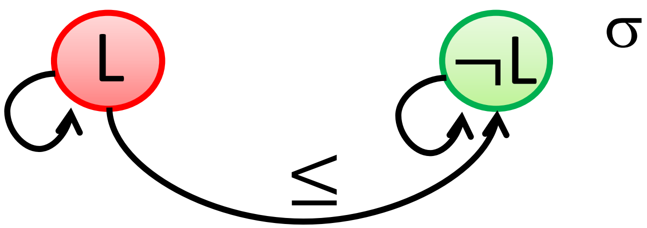
Use **diagrams** to generalize from states

- state σ is a finite first-order structure

$$\mathrm{Diag}(\sigma) = \exists\, x\, y.\ x \neq y \wedge L(x) \wedge \neg L(y) \wedge \leq(x, y) \wedge \neg\leq(y, x) \wedge \leq(x, x) \wedge \leq(y, y)$$

σ' ⊨ Diag(σ) iff σ is a substructure of σ'

σ is obtained from σ' by removing elements and projecting relations on remaining elements



[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Proving Their Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

Generalization using Diagram

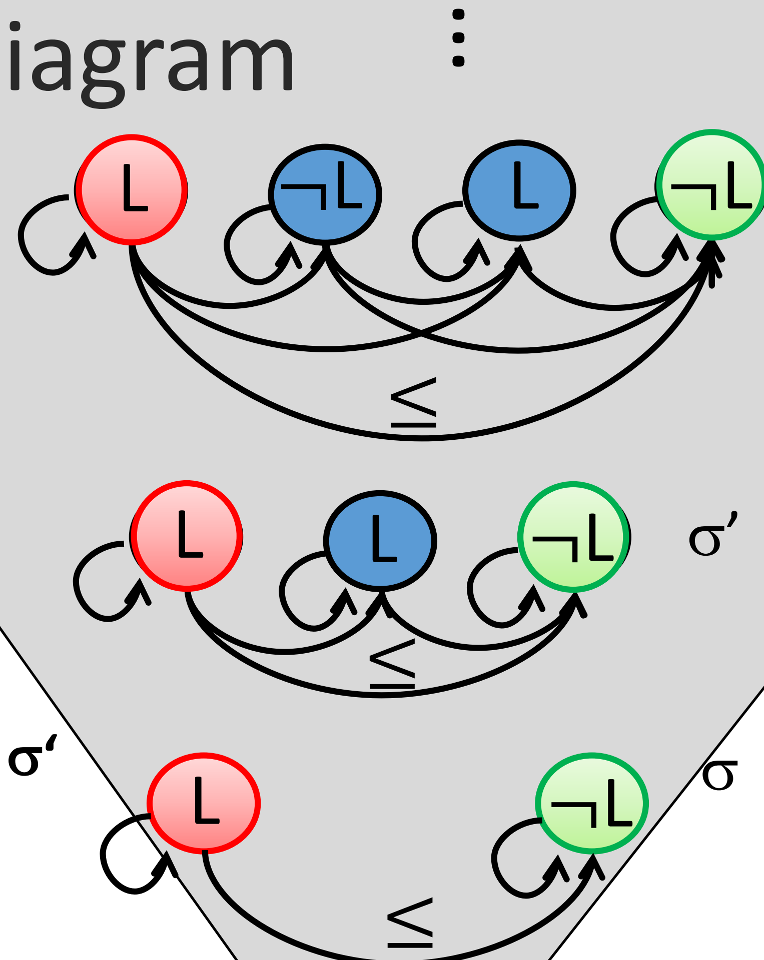
Use **diagrams** to generalize

- state σ is a finite first-or

$$\mathrm{Diag}(\sigma) = \exists\, x\, y.\ x \neq y \wedge L(x) \wedge \neg L(y) \wedge \leq(x, y) \wedge \neg{\leq}(y, x) \wedge \leq(x, x) \wedge \leq(y, y)$$

σ' ⊨ Diag(σ) iff σ is a substructure of σ'

σ is obtained from σ' by removing elements and projecting relations on remaining elements



[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Proving Their Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

Generalization using Diagram

Can generalize more

- remove facts/conjuncts

$\mathrm{Diag}(\sigma) = \exists\, x\, y.\ x \neq y \wedge$ ~~$L(x) \wedge \neg L(y)$~~

$\qquad \wedge \leq(x, y) \wedge \neg\leq(y, x)$

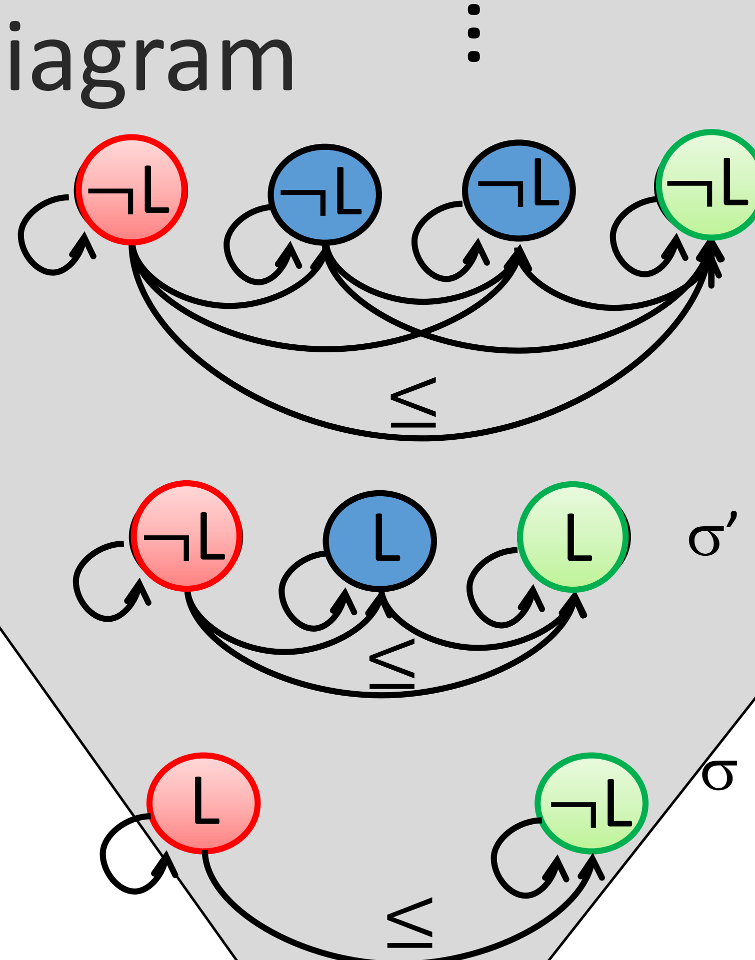
$\qquad \wedge \leq(x, x) \wedge \leq(y, y)$

$\mathrm{gen}(\mathrm{Diag}(\sigma)) = \exists\, x\, y.\ x \neq y$

$\qquad \wedge \leq(x, y) \wedge \neg\leq(y, x)$

$\qquad \wedge \leq(x, x) \wedge \leq(y, y)$

$\mathrm{avoid}(\sigma) = \neg\mathrm{gen}(\mathrm{Diag}(\sigma))$

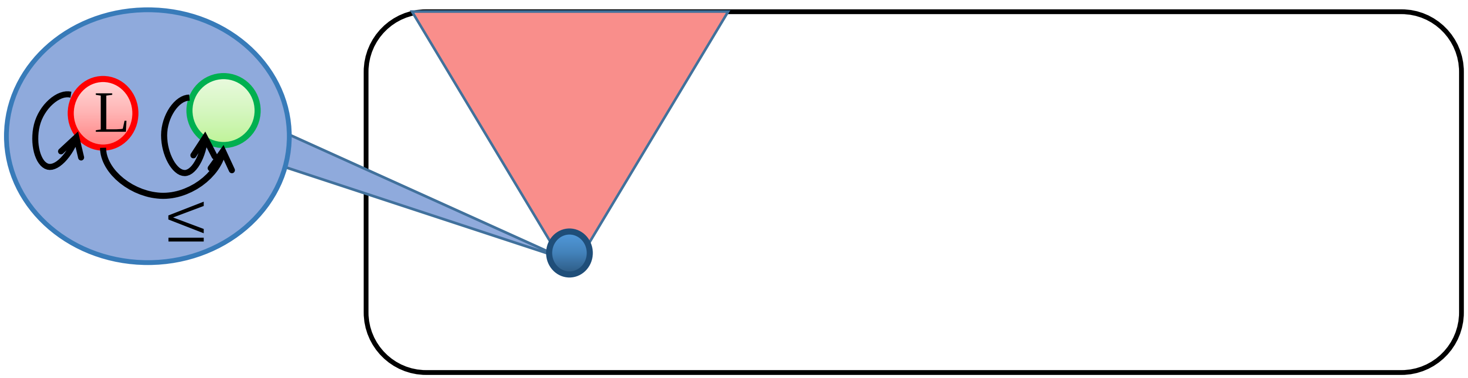


[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Proving Their Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

From Diagrams to Invariants

$\exists \overline{x}. (\neg \mathrm{I}_{1,1}(\bar{x}) \wedge \ldots \wedge \neg \mathrm{I}_{1,m}(\bar{x}))$

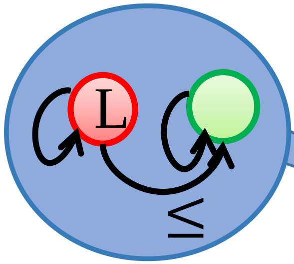
Diagram

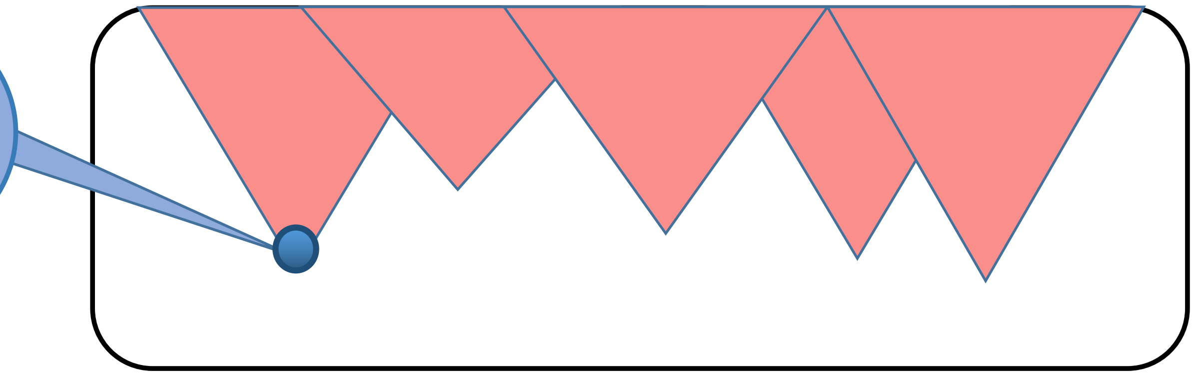


From Diagrams to Invariants

conjecture

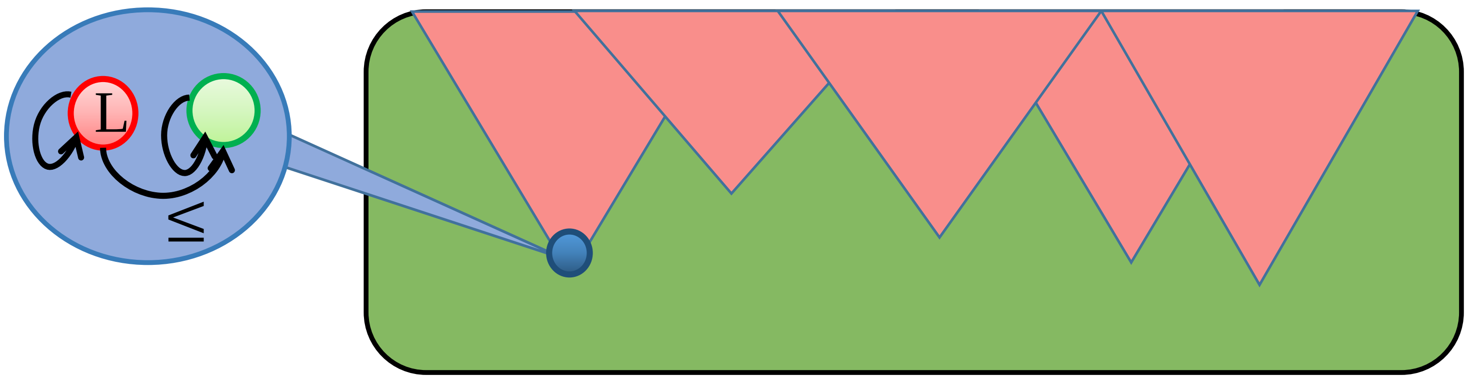
$$\neg\, \exists \bar{x}.\, (\neg l_{1,1}(\bar{x}) \wedge \ldots \wedge \neg l_{1,m}(\bar{x}))$$





From Diagrams to Invariants

$$Inv \equiv \overbrace{\neg \exists \bar{x}.\, (\neg l_{1,1}(\bar{x}) \wedge \ldots \wedge \neg l_{1,m}(\bar{x}))}^{\text{conjecture}} \wedge \ldots \wedge \neg \exists \bar{x}.\, (\neg l_{n,1}(\bar{x}) \wedge \ldots \wedge \neg l_{n,m}(\bar{x}))$$



Q: How to select which facts to remove in the generalization?

IVy: **interact** with the user to identify **irrelevant facts**

Leader Election: Iteration 3

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

1. Each node **sends** its id to the next
2. A node that **receives** a msg passes it to the next node in the ring if the id in the msg $\geq$ the node's id
3. A node that receives its own id becomes the **leader**

Safety property:

$$I_0 = \neg Bad = \forall x,y: Node.\ \mathbf{leader}(x) \land \mathbf{leader}(y) \rightarrow x = y$$

Inductive invariant: $Inv = I0 \land I1 \land I2 \land I3$

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

The leader has the highest ID

$$I_2 = \forall n_1,n_2: Node.\ pnd(id[n_2], n_2) \rightarrow id[n_1] \leq id[n_2]$$

Only highest id can be self-pnd

$$I_3 = \forall n_1,n_2,n_3: Node.\ btw(n_1,n_2,n_3) \land pnd(id[n_2], n_1) \rightarrow id[n_3] \leq id[n_2]$$

Cannot bypass higher nodes

Leader Election: Iteration 3

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

1. Each node **sends** its id to the next
2. A node that **receives** a msg passes it to the next node in the ring if the id in the msg $\geq$ the node's id
3. A node that receives its own id becomes the **leader**

Safety property:

$$I_0 = \neg Bad = \forall x,y: Node.\ \mathbf{leader}(x) \land \mathbf{leader}(y) \rightarrow x = y$$

Inductive invariant: Inv = I0 ∧ I1 ∧ I2 ∧ I3

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

The leader has the highest ID

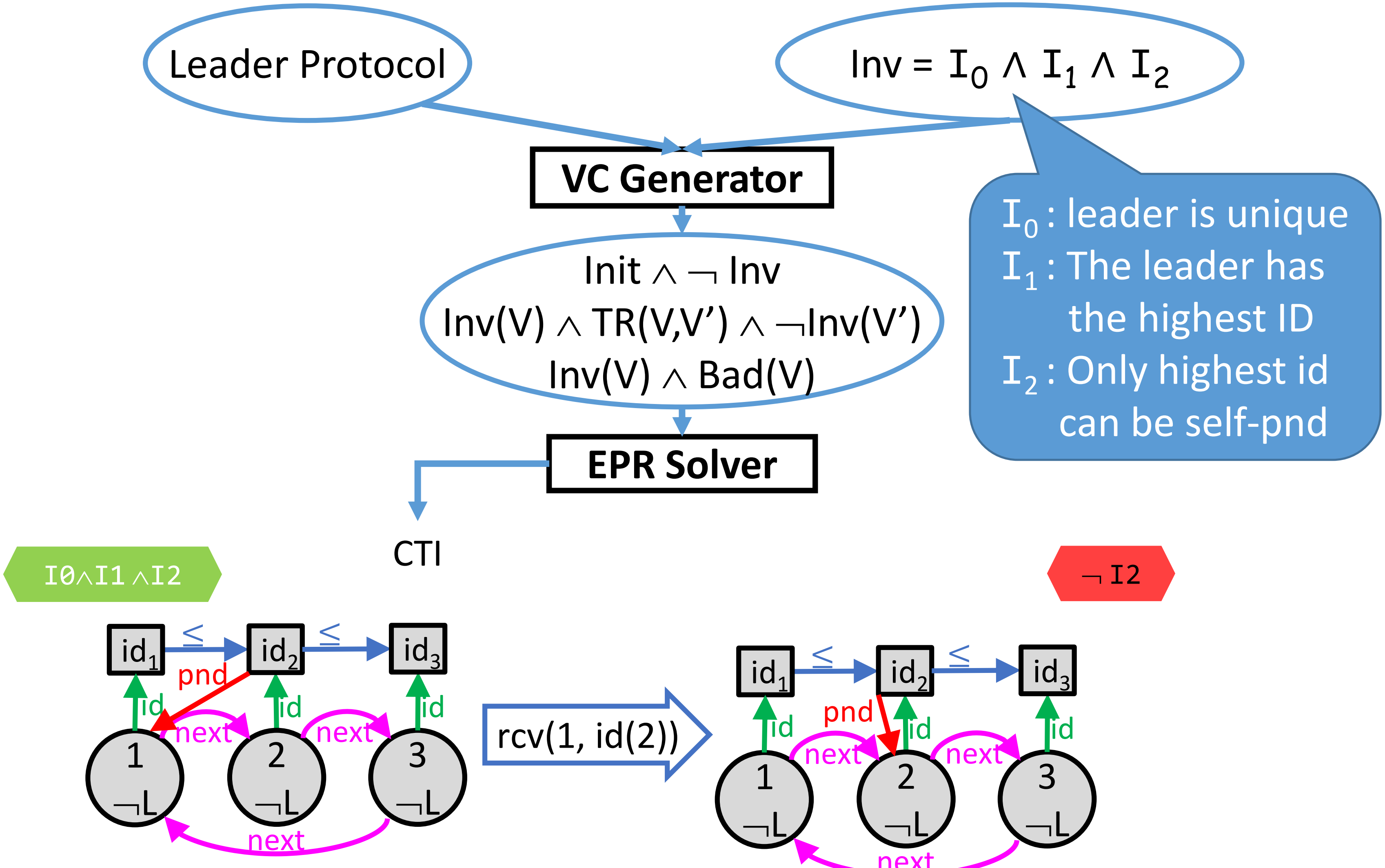
$$I_2 = \forall n_1,n_2: Node.\ pnd(id[n_2], n_2) \rightarrow id[n_1] \leq id[n_2]$$

Only highest id can be self-pnd

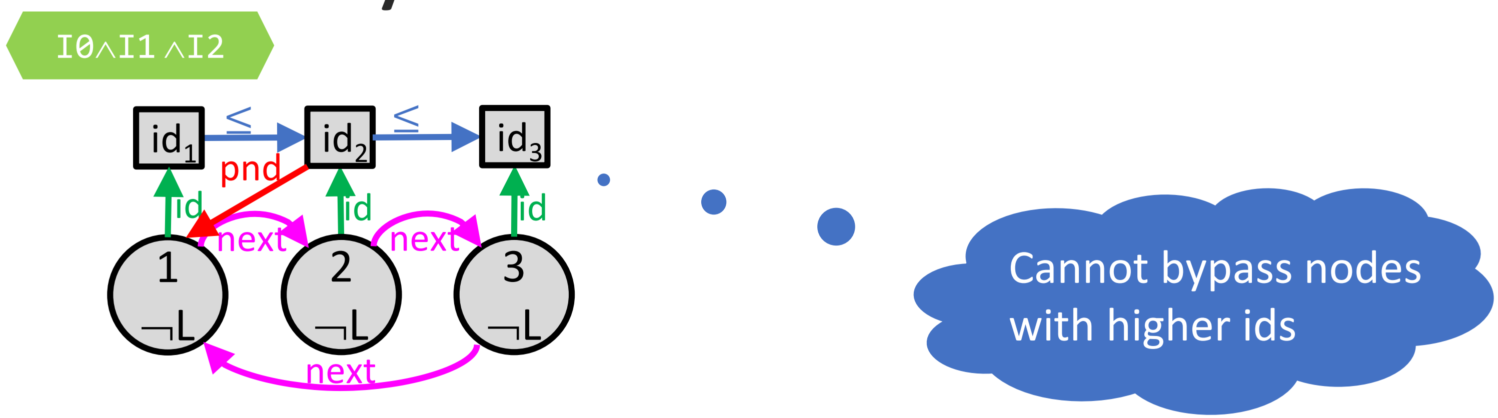
$$I_3 = \forall n_1,n_2,n_3: Node.\ btw(n_1,n_2,n_3) \land pnd(id[n_2], n_1) \rightarrow id[n_3] \leq id[n_2]$$

Cannot bypass higher nodes

IVy: Check Inductiveness



IVy: Generalize from CTI

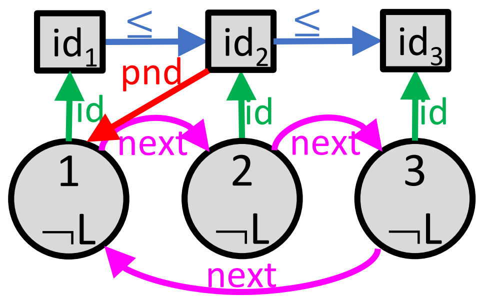


id[n_2] is pending for n_1, had to go through n_3

1. Each node **sends** its id to the next
2. A node that **receives** a msg passes it to the next node in the ring if the id in the msg ≥ the node's id
3. A node that receives its own id becomes the **leader**

IVy: Generalize from CTI

I0∧I1 ∧I2



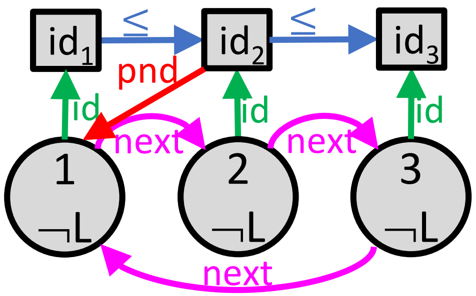
User's Generalization

- ☑ ≤
- ☑ btw
- ☑ id
- ☑ pnd
- ☑ L

Cannot bypass nodes with higher ids

IVy: Generalize from CTI

I0∧I1 ∧I2

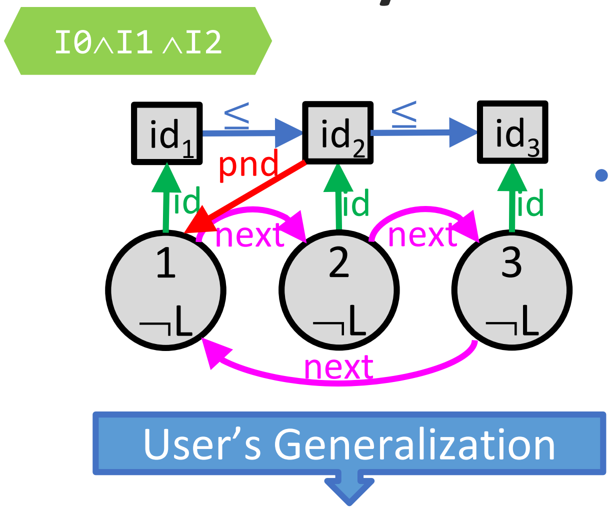


User's Generalization

- [x] ≤
- [x] btw
- [x] id
- [x] pnd
- [ ] L

Cannot bypass nodes with higher ids

IVy: Generalize from CTI

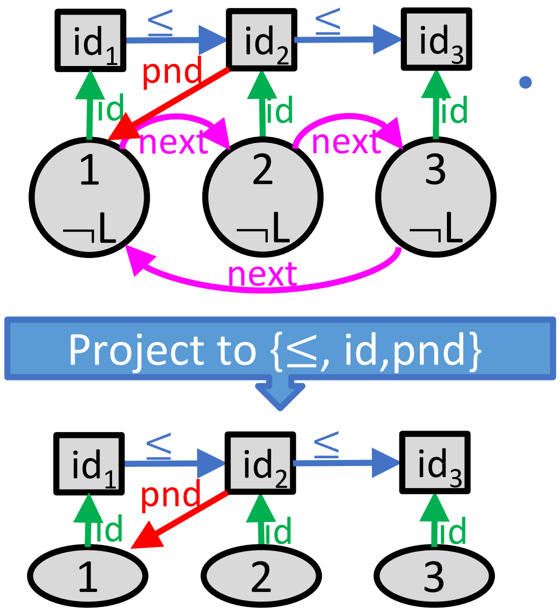


- [x] ≤
- [ ] btw
- [x] id
- [x] pnd
- [ ] L

Cannot bypass nodes with higher ids

IVy: Generalize from CTI

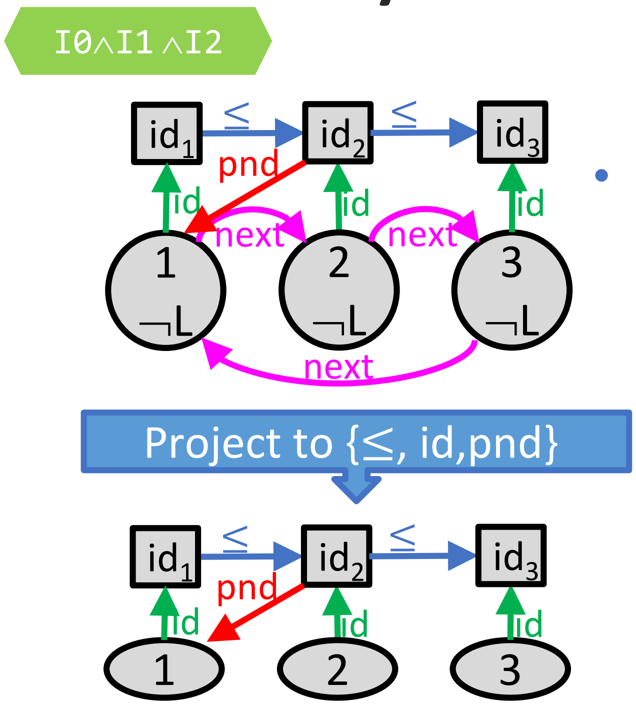
I0∧I1 ∧I2



- ☑ ≤
- ☐ btw
- ☑ id
- ☑ pnd
- ☐ L

Cannot bypass nodes with higher ids

IVy: Generalize from CTI

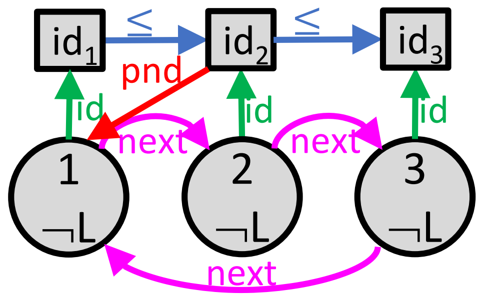


Cannot bypass nodes with higher ids

$$
\begin{aligned}
\neg\exists n_1,n_2,n_3 : \text{Node}.\ & \neq(n_1,n_2,n_3) \wedge \\
& \neq(id[n_1],id[n_2],id[n_3]) \wedge \\
& id[n_1] \leq id[n_2] \leq id[n_3] \wedge \\
& pnd(id[n_2], n_1)
\end{aligned}
$$

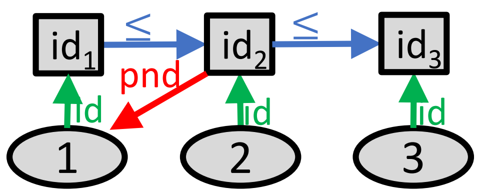
IVy: Generalize from CTI

I0∧I1∧I2



Cannot bypass nodes with higher ids

Project to {≤, id,pnd}



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

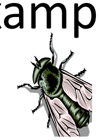
BMC(3)

BMC VC Generator (K=3, $\neg C_3$)

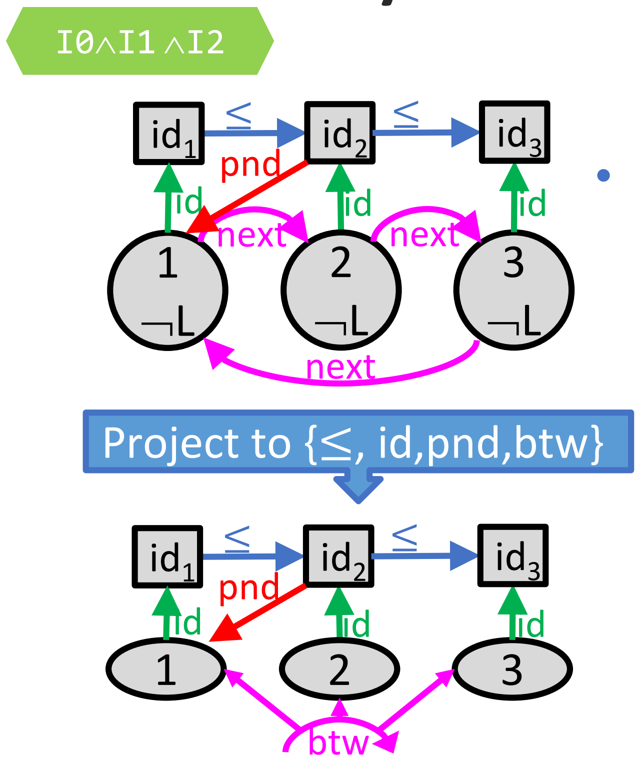
$$Init(V_0) \wedge TR(V_0,V_1) \wedge TR(V_1,V_2) \wedge TR(V_1,V_3) \wedge \neg C_3(V_3)$$

EPR Solver

Counterexample Trace



IVy: Generalize from CTI

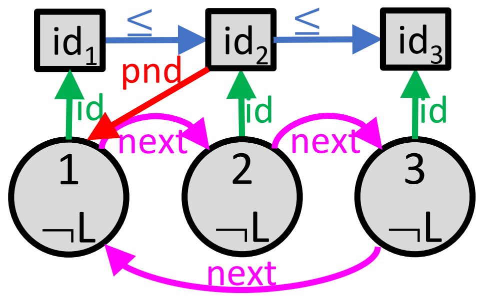


Cannot bypass nodes with higher ids

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

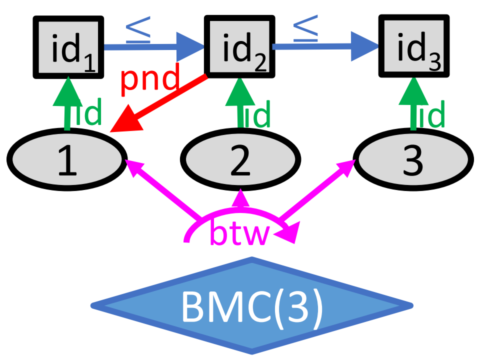
IVy: Generalize from CTI

$I0 \wedge I1 \wedge I2$



Cannot bypass nodes with higher ids

Project to {≤, id,pnd,btw}



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

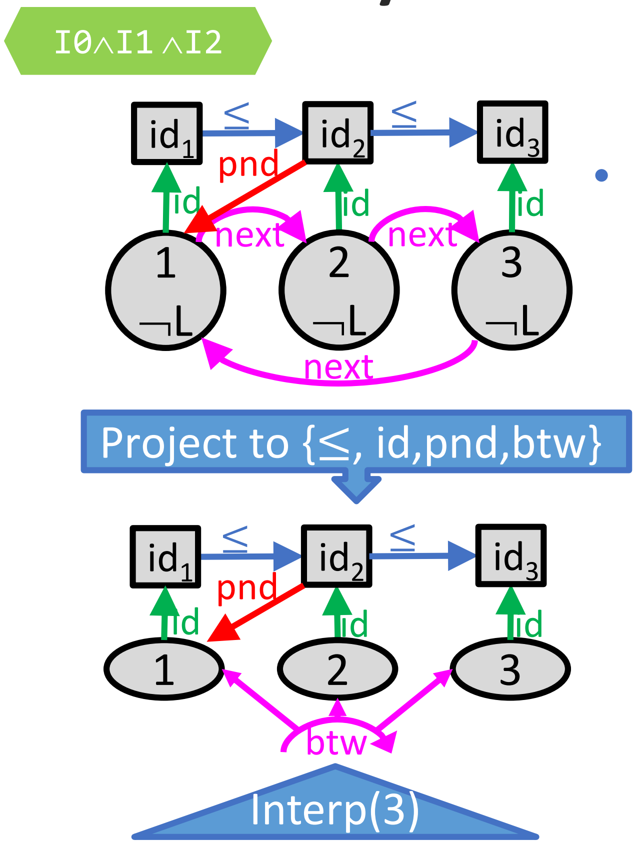
BMC VC Generator (K=3, $\neg C_3$)

$$Init(V_0) \wedge TR(V_0,V_1) \wedge TR(V_1,V_2) \wedge TR(V_1,V_3) \wedge \neg C_3(V_3)$$

EPR Solver

Proof

IVy: Generalize from CTI

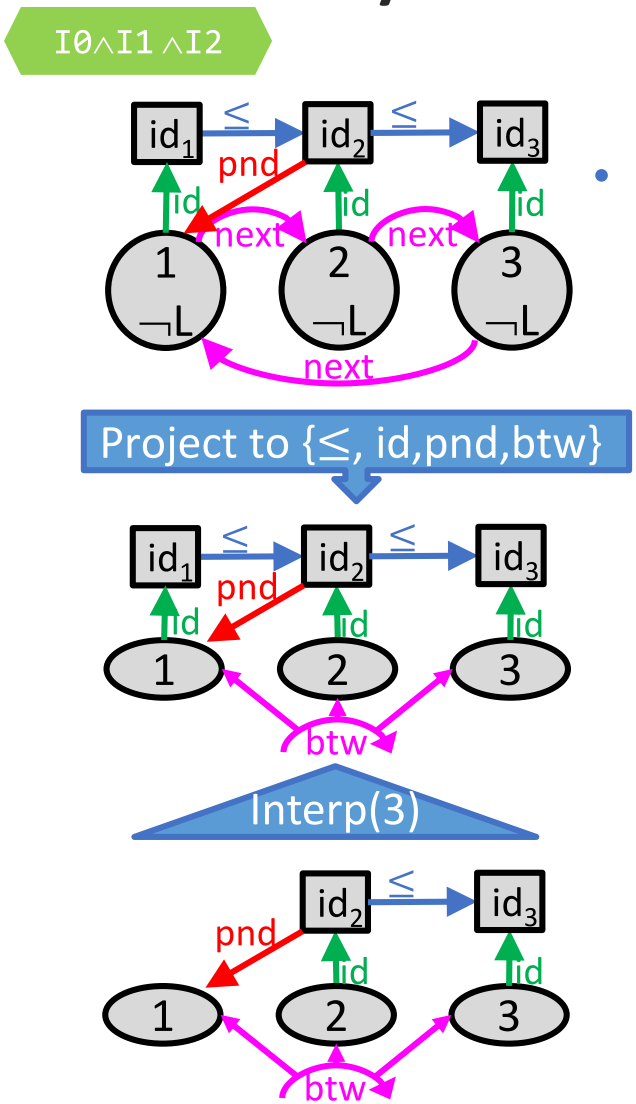


Cannot bypass nodes with higher ids

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

Proof

IVy: Generalize from CTI



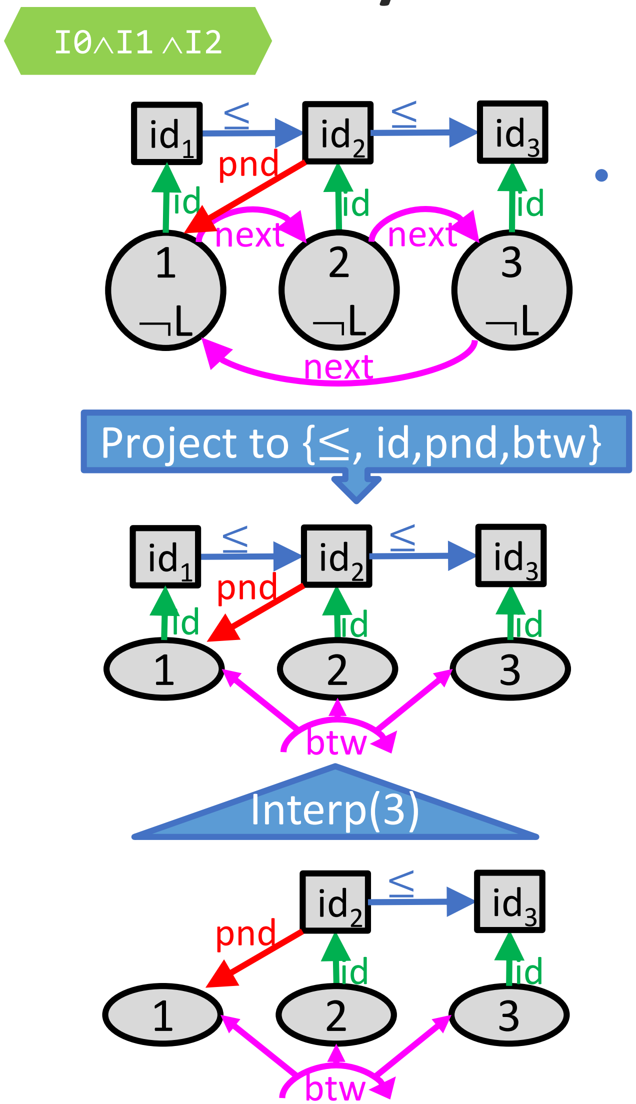
Cannot bypass nodes with higher ids

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

 Proof

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

IVy: Generalize from CTI



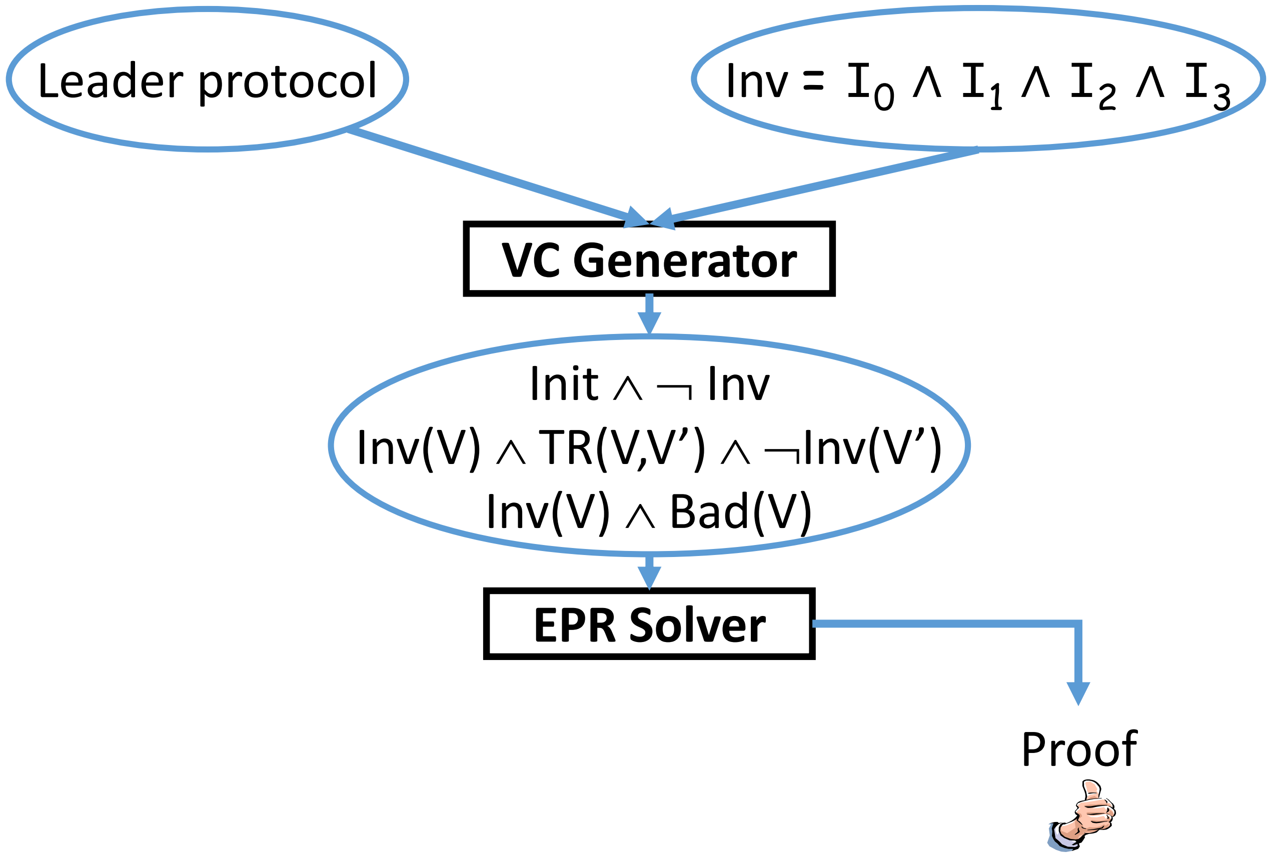
Cannot bypass nodes with higher ids

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

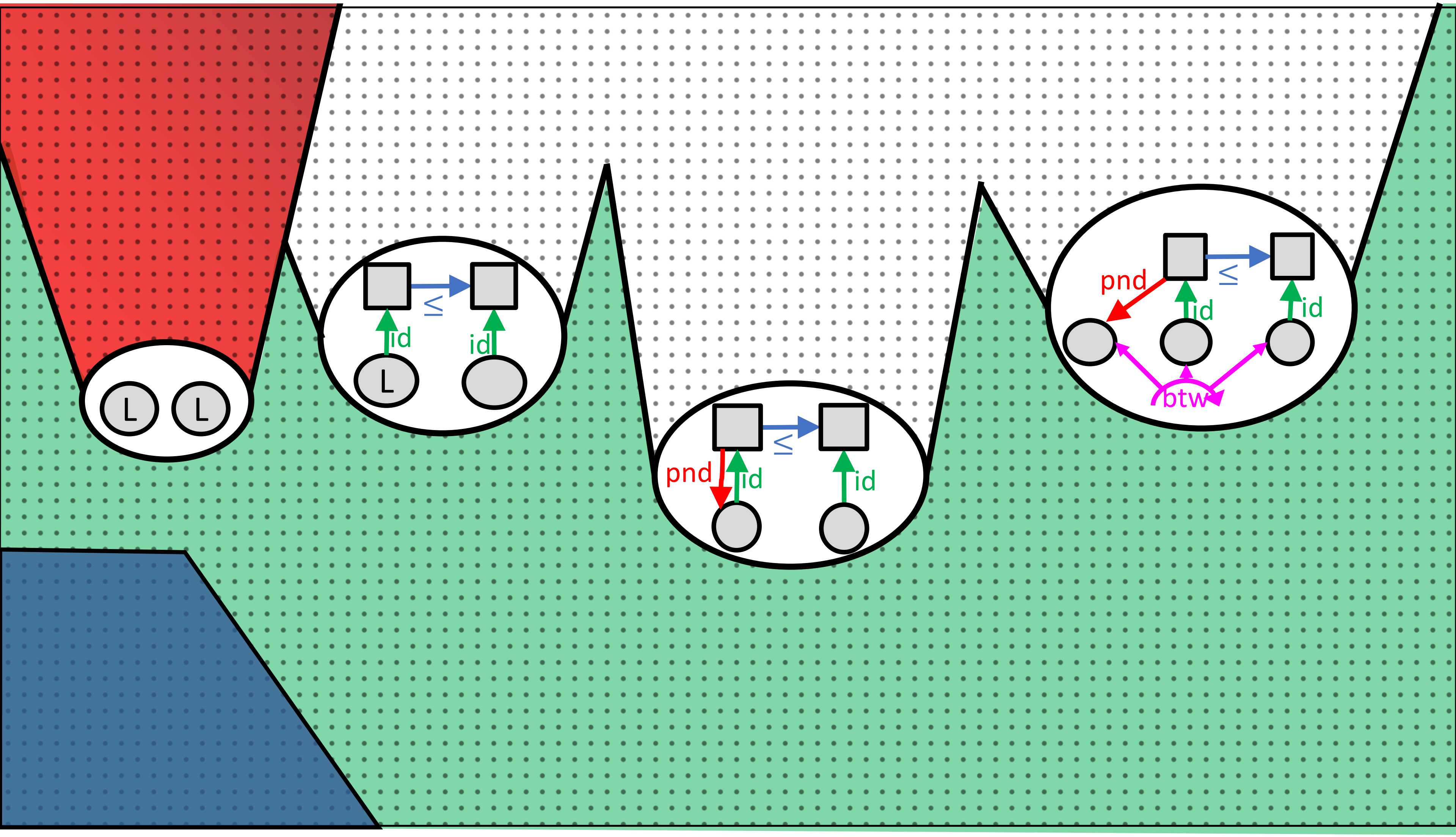
Looks good, add to the invariant as I_3

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

IVy: Check Inductiveness



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



L
L
L
id
id
≤
pnd
id
id
≤
pnd
id
id
≤
btw

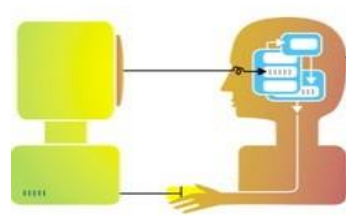
Recap: Supervised Verification in IVy

Check inductiveness

BMC

Interpolation

⋮



Projection of relevant facts

BMC bounds

Examining conjectures

⋮

Decidable Problems

Predictable Automation

Proof intuition and creativity

Graphical interaction

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

(1) Limitations of first-order logic

- Expressing transitive closure
  - Ring protocols
- Expressing arithmetic
  - Node id’s

Domain knowledge and axioms

Axioms: Leader Election Protocol

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

	Intention	EPR Modeling
Node ID's	Integers	**∀i:ID.** $i \leq i$ **Reflexive** **∀i, j, k: ID.** $i\leq j \land j\leq k \rightarrow i\leq k$ **Transitive** **∀i, j: ID.** $i\leq j \land j\leq I \rightarrow i=j$ **Anti-Symmetric** **∀i, j: ID.** $i\leq j \lor j \leq i$ **Total** **∀x, y: Node.** $id(x) = id(y) \rightarrow x=y$ Injective
Ring Topology	Next edges + Transitive closure	**∀x, y, z: Node.** btw(x, y, z) → btw(y, z, x) Circular shifts **∀x, y, z, w: Node.** btw(w, x, y) ∧btw(w, y, z) → btw(w, x, z) Transitive **∀x, y, w: Node.** btw(w, x, y) → ¬btw(w, y, x) Anti-Symmetric **∀x, y, z, w: Node.** distinct(x, y, z) → btw(w, x, y) ∨ btw(w, y, x)
		"`next(a)=b`" ≡ **∀x: Node.** $x\neq a \land x\neq b\rightarrow$ `btw(a,b,x)`

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

(1) Limitations of first-order logic
- Expressing transitive closure
  - Ring protocols
- Expressing arithmetic
  - Node id's
- Expressing Consensus
  - Paxos, Multi-Paxos, Reconfiguration

Domain knowledge and axioms

(2) Restrictions for decidability
- Restricted quantification

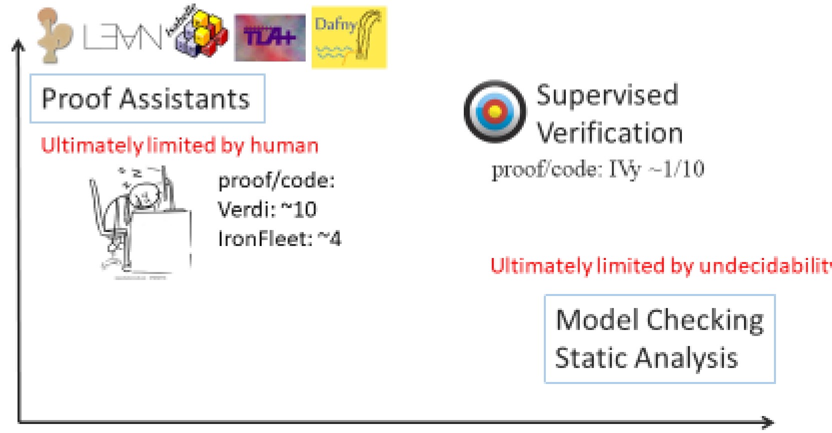
Derived relations and rewrites

[OOPSLA'17] Paxos Made EPR: Decidable Reasoning about Distributed Protocols. O. Padon, G. Losa, M. Sagiv, S. Shoham

IVy: Verified Protocols

Protocol	Model (# LOC)	Property (# Literals)	Invariant (# Literals)
Leader in Ring	59	3	12
Learning Switch	50	11	18
DB Chain Replication	143	11	35
Chord	155	35	46
Lock Server (500 Coq lines [Verdi])	122	3	21
Distributed Lock (1 week [IronFleet])	41	3	26
Single Decree Paxos	85	3	32
Multi Paxos	102	3	38
Vertical Paxos	123	3	65
Fast Paxos	117	3	59
Flexible Paxos	88	3	32
Stoppable Paxos	130	6	60
Virtually Synchronous Paxos	Work in progress		

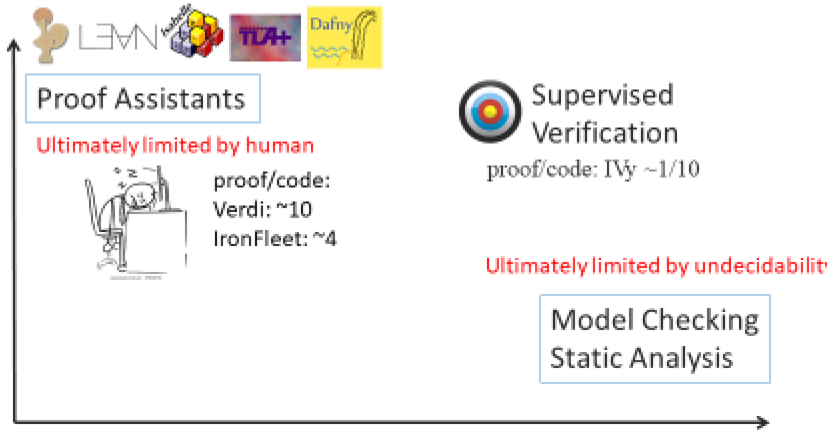
Summary



- Safety verification by
  - Automatic deduction
  - Interactive inference of invariants, graphical interaction
- Use decidable fragment of FOL
  - Deduction is decidable
  - Finite Counterexamples
- Interact with a user based on counterexamples to induction
- Surprisingly powerful
  - Paxos, Multi-Paxos, Reconfiguration, … [OOPSLA'17]
  - Liveness and Temporal Properties [POPL'18]

erc Supervised Verification of Infinite-State Systems

Future Work



- More distributed systems
- Other logics
- Other inference schemes
- Other forms of interaction
- More automation in inferring inductive invariants
- Theoretical understanding of limitations and tradeoffs

Seeking postdocs and students

erc Supervised Verification of Infinite-State Systems