Verification of Distributed Protocols Using Decidable Logic

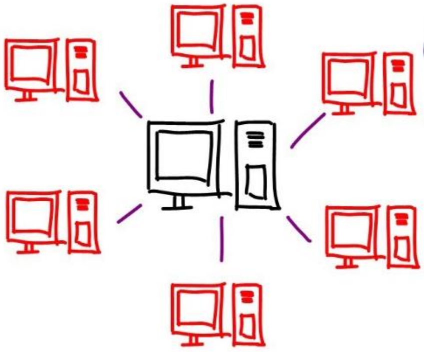
Sharon Shoham

Tel Aviv University

Programming Languages Mentoring Workshop 2019

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS])

Why verify distributed protocols?

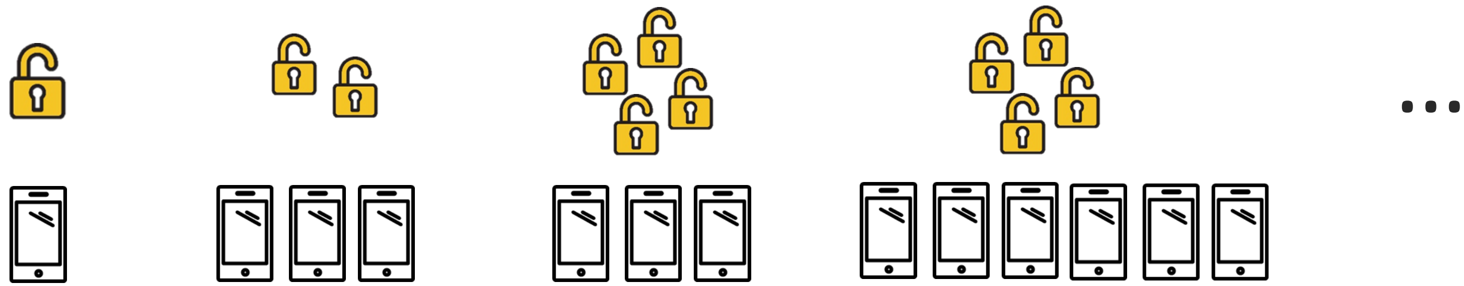


- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure
  - Blockchains

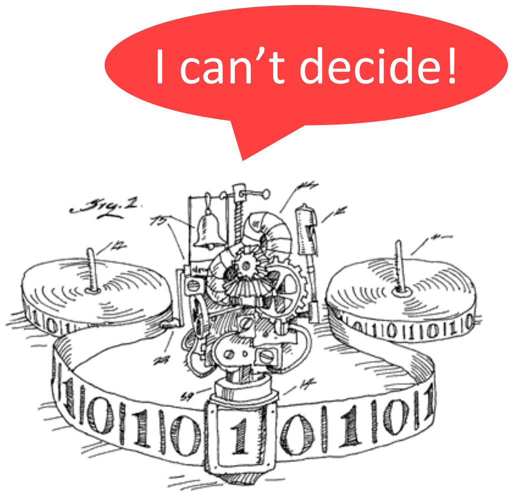
- Distributed protocols are notoriously hard to get right
  - Even small protocols can be tricky
  - Bugs occur on rare scenarios
  - Testing is costly and not sufficient

Verifying distributed protocols is hard

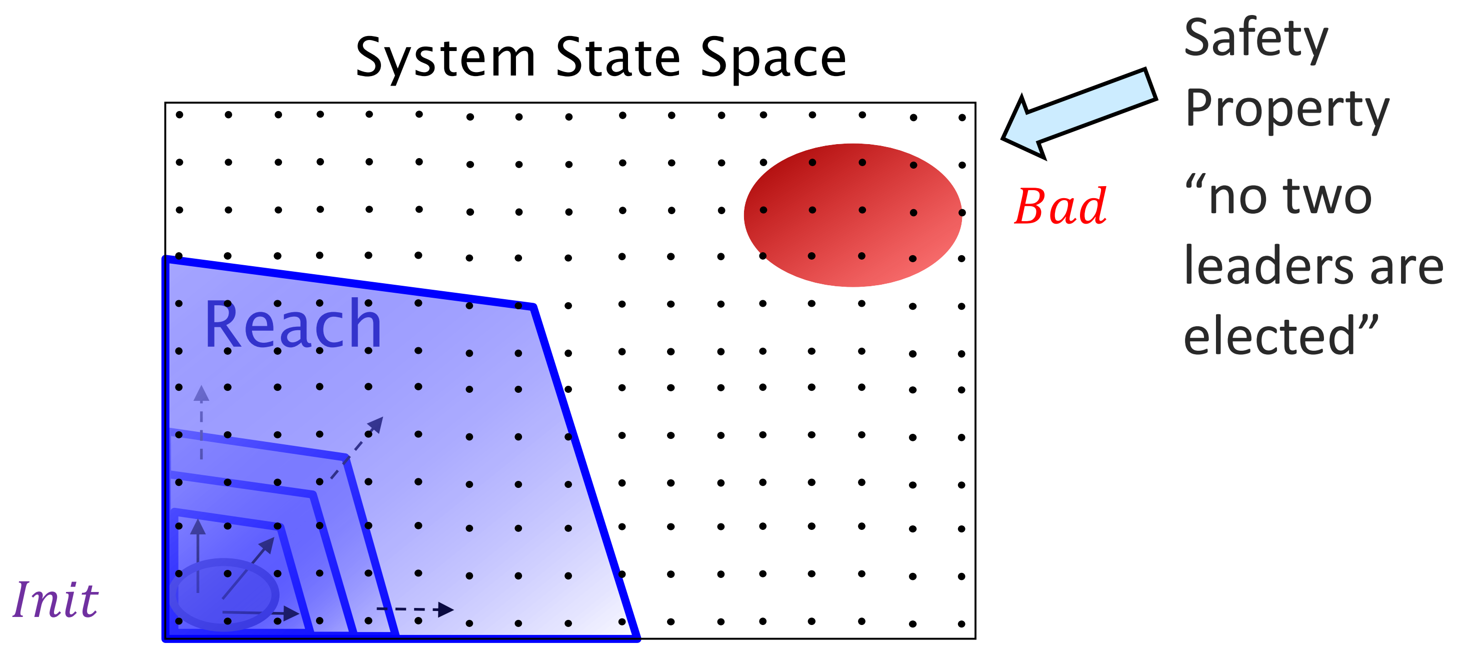
Verify distributed protocols for any number of nodes and resources



- Infinite state-space
  - unbounded #processes
  - unbounded #messages
  - unbounded #objects
- Asymptotic complexity of verification
  - Rice theorem

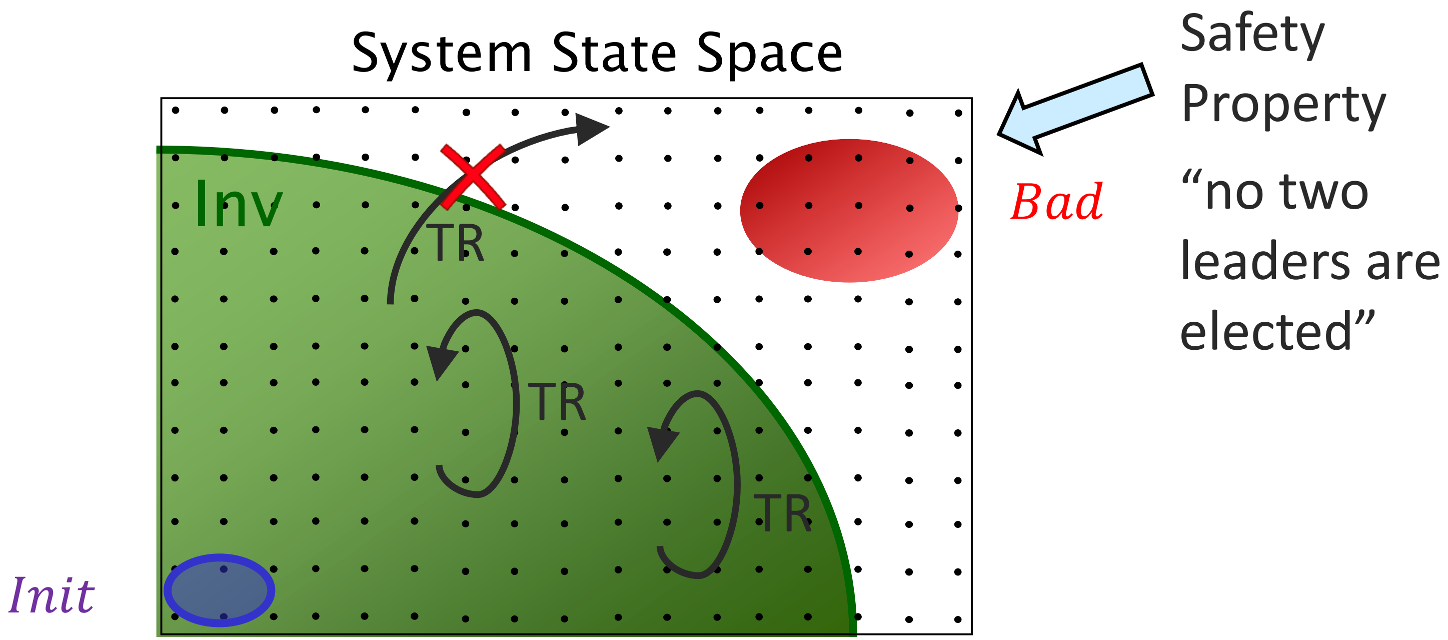


Safety of Infinite State Systems



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

Inductive Invariants



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

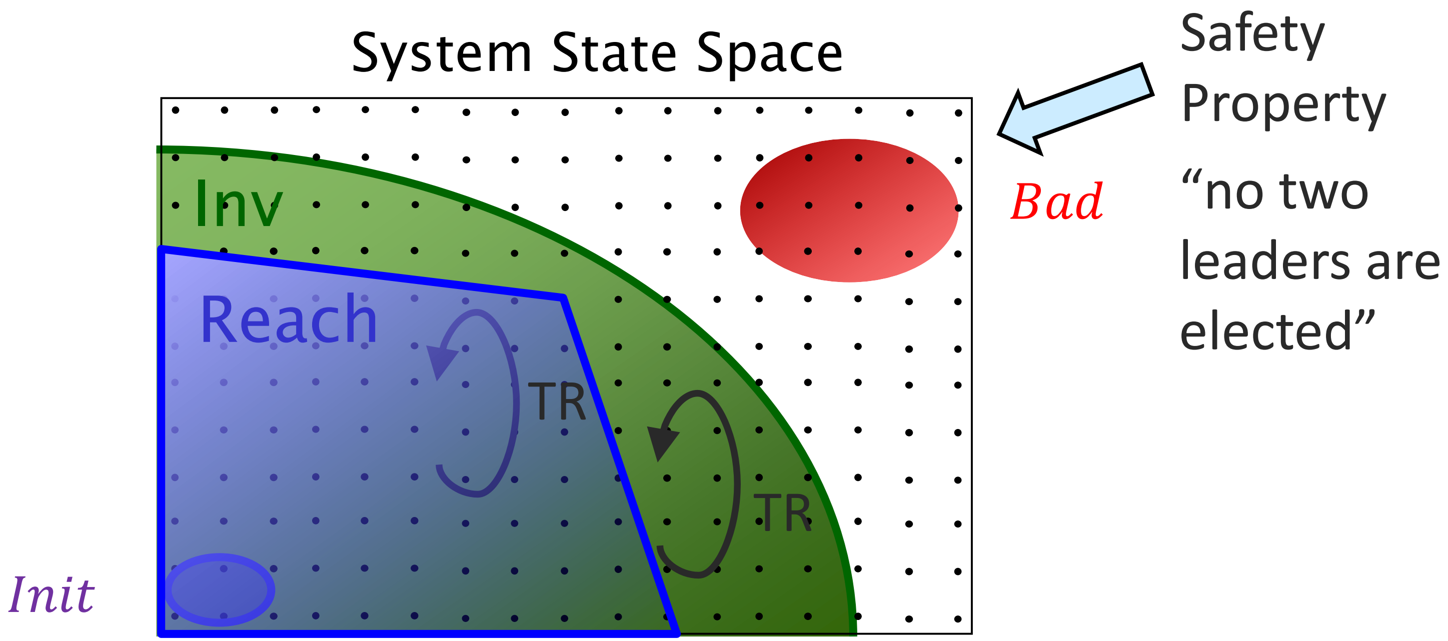
System S is safe iff there exists an **inductive invariant** Inv:

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

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

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

Inductive Invariants



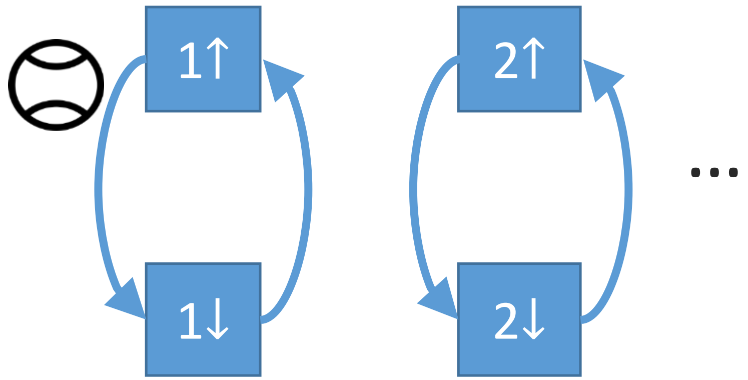
System S is **safe** if all the reachable states satisfy the property P = $\neg Bad$

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**)

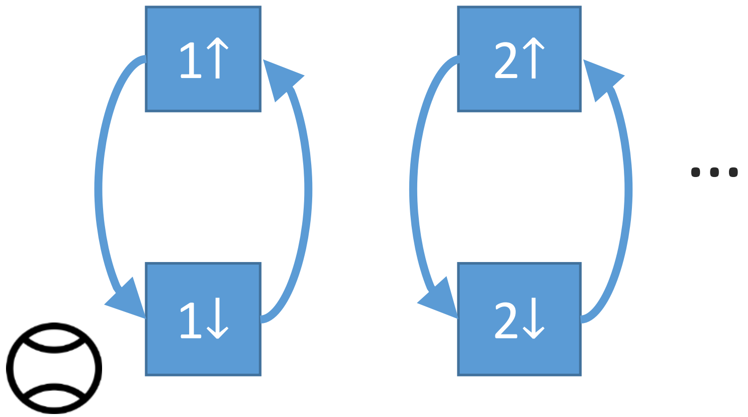
Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ …



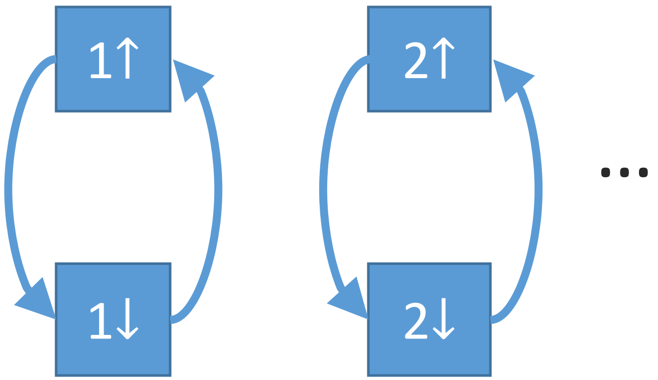
Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ ...



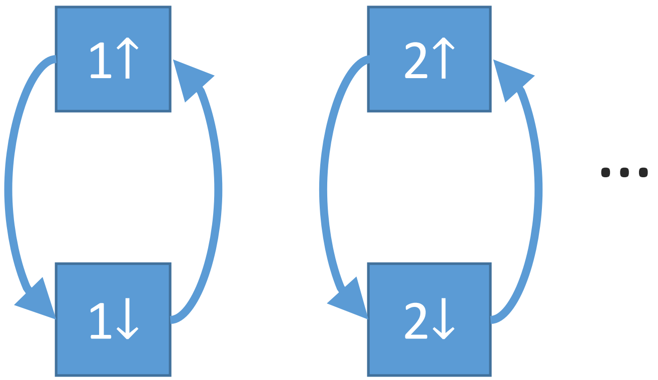
Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ …
- The ball starts at player 1↑
- Can the ball get to 2↓?



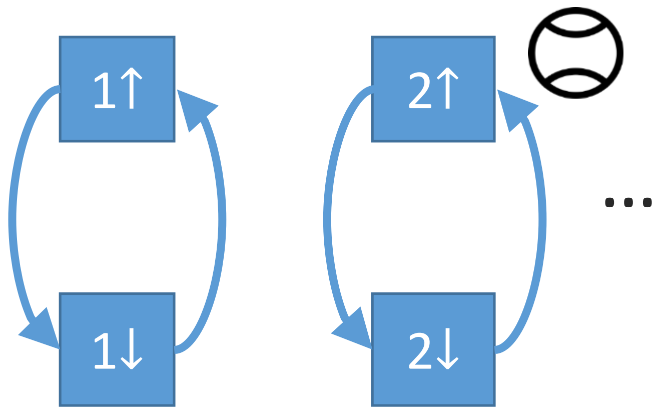
Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ ...
- The ball starts at player 1↑
- Can the ball get to 2↓?
- Is "**the ball is not at 2↓**" an inductive invariant?



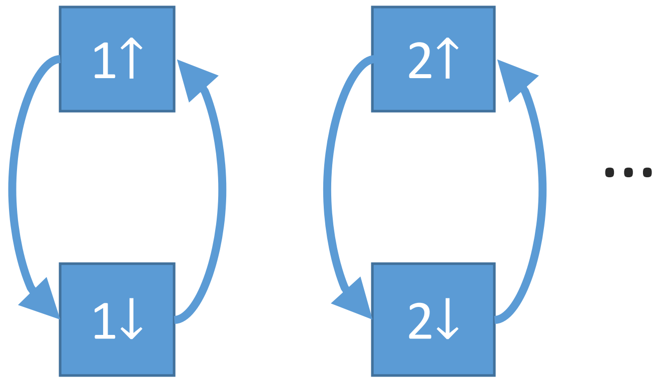
Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ …
- The ball starts at player 1↑
- Can the ball get to 2↓?
- Is "**the ball is not at 2↓**" an inductive invariant? **No!**
  - **Counterexample to induction**



Example

- N pairs of players pass a ball:
  - 1↑ will pass to 1↓
  - 1↓ will pass to 1↑
  - 2↑ will pass to 2↓
  - 2↓ will pass to 2↑ …



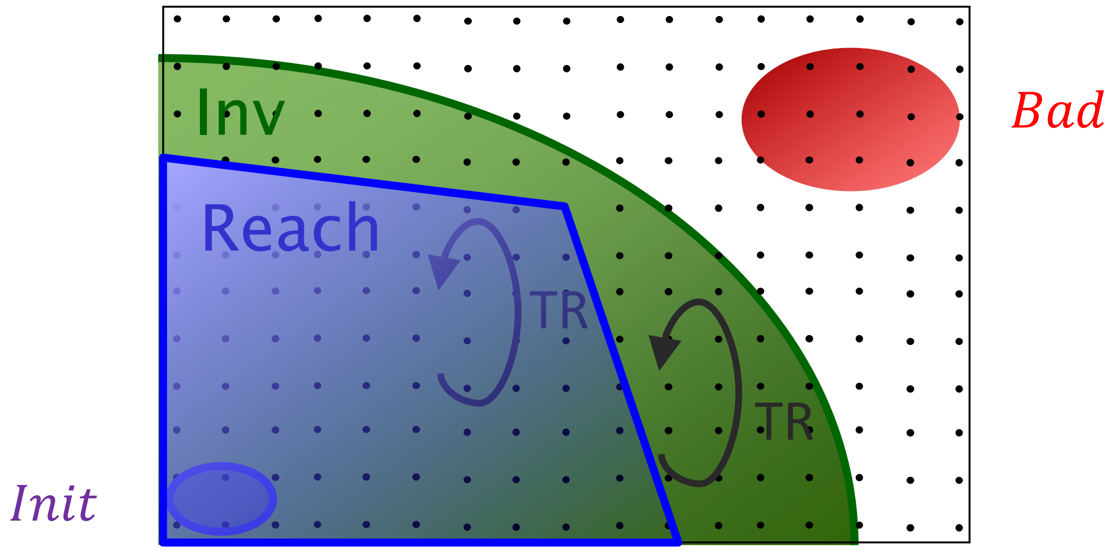
- The ball starts at player 1↑
- Can the ball get to 2↓?
- Is "**the ball is not at 2↓**" an inductive invariant? **No!**
  - **Counterexample to induction**
- Inductive invariant: "**the ball is not at 2↑ nor 2↓**"

Logic-based verification

Provers/solvers for different logics made huge progress

- Propositional logic (SAT) – industrial impact for hardware verification
- Satisfiability modulo theories (SMT) – major trend in software verification
- Automated first-order theorem provers
- Interactive theorem provers
- Z3, CVC4, iProver, Vampire, Coq, Isabelle/HOL ….

Logic-based verification



Represent *Init*, *Tr*, *Bad*, *Inv* by logical formulas: **Formula ⇔ Set of states**

Inv(V) is an inductive invariant if the verification conditions (VCs) are valid:

Initiation	$Init(V) \Longrightarrow Inv(V)$		unsat($Init(V) \wedge \neg Inv(V)$)
Cons.	$Inv(V) \wedge TR(V,V') \Longrightarrow Inv(V')$		unsat($Inv(V) \wedge TR(V,V') \wedge \neg Inv(V')$)
Safety	$Inv(V) \Longrightarrow \neg Bad(V)$		unsat($Inv(V) \wedge Bad(V)$)

Challenges for logic-based verification

Formal specification

Modeling the system and its invariants

Deduction

Checking validity of the VCs

Inference

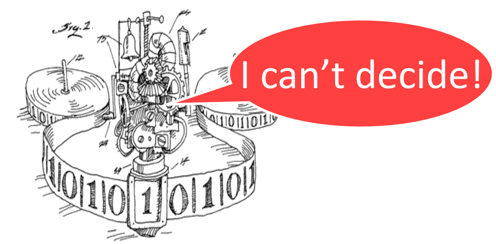
Finding an inductive invariant

Deduction

Inv(V) is an inductive invariant if the following verification conditions are valid:

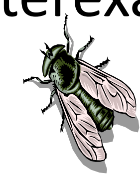
Initiation	$Init(V) \Longrightarrow Inv(V)$	unsat($Init(V) \wedge \neg Inv(V)$)
Cons.	$Inv(V) \wedge TR(V,V') \Longrightarrow Inv(V')$	unsat($Inv(V) \wedge TR(V,V') \wedge \neg Inv(V')$)
Safety	$Inv(V) \Longrightarrow \neg Bad(V)$	unsat($Inv(V) \wedge Bad(V)$)

Church's Theorem



Are the logical VC's valid ?

Counterexample



Unknown / Diverge

Proof

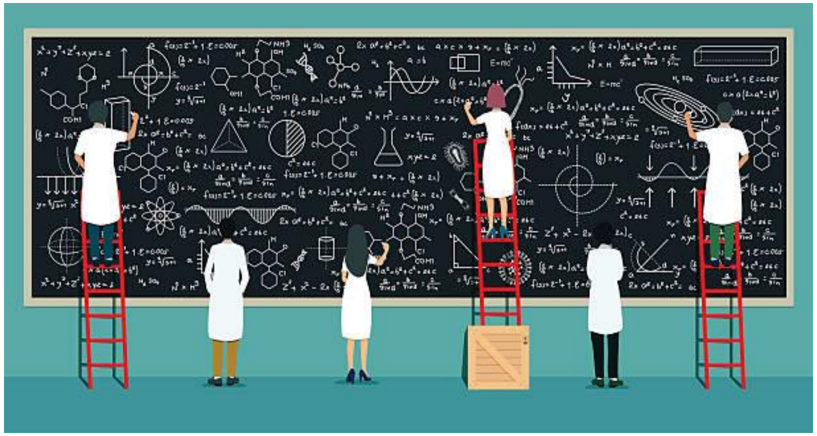
Deduction

Interactive theorem provers

(Coq, Isabelle/HOL, LEAN)

- Programmer proves the inductive invariant
- Huge programmer effort (~10-50 lines of proof per line of code)

e.g. Verdi



Automatic solvers/provers

(e.g. Z3, CVC4, Vampire)

- VCs discharged automatically
- Tools may diverge (for SMT: matching loops, arithmetic)
- Unpredictability (butterfly effect)

e.g. Ironfleet

Logic-based verification approaches

Expressiveness

Interactive theorem provers

- Huge programmer effort (~10-50 lines of proof per line of code)

Automated deductive verification

- SMT solver may diverge (matching loops, arithmetic)
- Unpredictability, butterfly effect

Model Checking, Abstract Interpretation

- Limited due to undecidability

Automation

Logic-based verification approaches

Expressiveness

Interactive theorem provers

- Huge programm
(~10-50 lines o
per line of cod

Desired

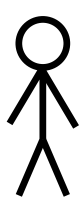
- Expressiveness
- High degree of automation
- Predictability
- Comprehensibility for users
- Efficiency/scalability

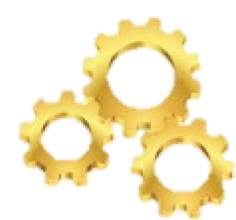
rn

Model
Checking,
Abstract
erpretation

- Limited due to undecidability

Automation

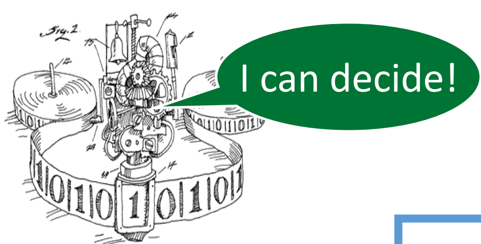




This talk: Restrict VC's to decidable logic

Inv(V) is an inductive invariant if the following verification conditions are valid:

Initiation	Init(V)⟹Inv(V)	unsat(Init(V)∧¬Inv(V))
Cons.	Inv(V)∧TR(V,V') ⟹ Inv(V')	unsat(Inv(V)∧TR(V,V')∧¬Inv(V'))
Safety	Inv(V) ⟹ ¬Bad(V)	unsat(Inv(V)∧Bad(V))

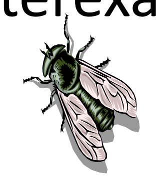


Are the logical VC's valid ?

∈ Decidable logic

With good tool support

Counterexample



Proof

Challenges for verification with decidable logic

Formal specification

Modeling in a decidable logic

Deduction

Checking validity of the VC's ✓

Invariant inference

Finding an inductive invariant

This talk

Logic: EPR – decidable fragment of first order logic

Formal specification

- Surprisingly expressive

Invariant inference

- Automatic (based on PDR)
  - Semi-algorithm: may diverge
- Interactive
  - Based on graphically displayed counterexamples to induction

Effectively Propositional Logic – EPR

Decidable fragment of first order logic

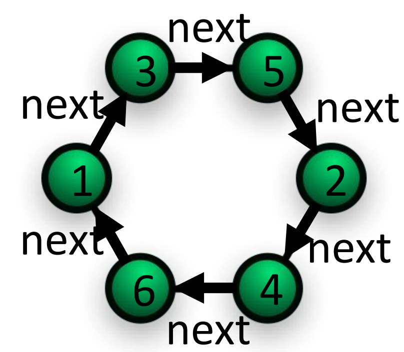
+ Quantification (∃*∀*)   - Theories (e.g., arithmetic)

- ☺ Allows quantifiers to reason about unbounded sets
  - $\forall x,y.\ \mathbf{leader}(x) \land \mathbf{leader}(y) \rightarrow x = y$
- ☺ Satisfiability is decidable => Deduction is decidable
- ☺ Small model property => Finite cex to induction
- ☺ Turing complete modeling language
- ☹ Limited language for safety and inductive invariants
  - ➢ Suffices for many infinite-state systems

Successful verification with EPR

- Shape Analysis
  [Itzhaky et al. CAV'13, POPL'14, CAV'14, CAV'15]
- Software-Defined Networks
  [Ball et al. PLDI'14]
- Distributed protocols
  [Padon et al. PLDI'16, OOPSLA'17, POPL'18, PLDI'18]
- Concurrent Modification Errors in Java programs
  [Frumkin et al. VMCAI'17]

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

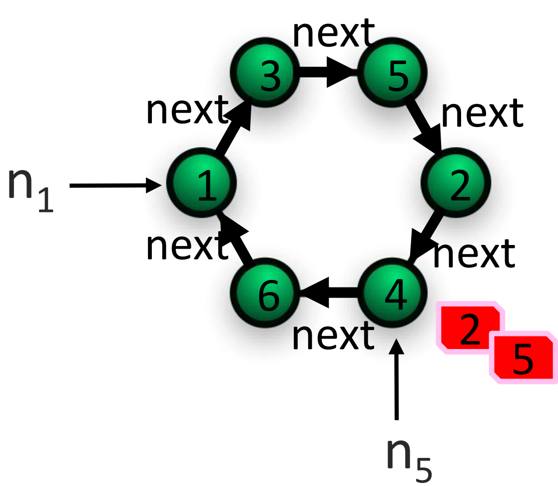
[CACM'79] E. Chang and R. Roberts. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*

Modeling with EPR

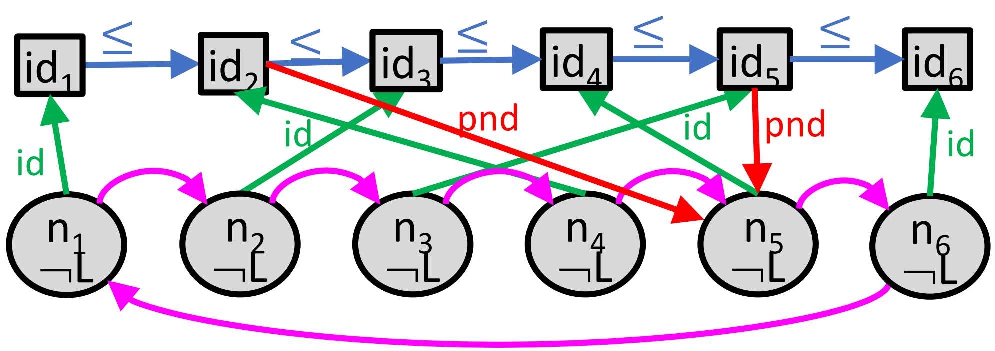
- State: finite first-order structure over vocabulary V
  - $\leq$ (ID, ID) – total order on node id's
  - **id**: Node → ID – relate a node to its id
  - **btw** (Node, Node, Node) – the ring topology
  - **pending**(ID, Node) – pending messages
  - **leader**(Node) – leader(n) means n is the leader

Axiomatized in EPR (≤, id, btw)

protocol state



structure



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

Modeling with EPR

- State: finite first-order structure over vocabulary V (+ axioms)

- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall$ id, n. ¬**pending**(id, n)
  - Bad(V) – bad states, e.g., $\exists n_1,n_2.\textbf{leader}(n_1) \land \textbf{leader}(n_2) \land n_1 \neq n_2$

- Transition relation: expressed as EPR formula TR(V, V'), e.g.:

  ∃n,s. "s = **next**(n)"∧ ∀x,y. **pending'**(x,y)↔ (**pending**(x,y) ∨ (x=**id**[n]∧y=s))

  ∨ ∃n. **pending** (**id**[n],n) ∧ ∀x. **leader'**(x) ⟷ (**leader**(x) ∨ x=n)

  …

Modeling with EPR

- State: finite first-order structure over vocabulary V (+ axioms)

Propose(n): send(id(n), next(n))

Recv(n,msg): if msg = id(n) then leader(n) := true

if msg > id(n) then send(msg,next(n))

- Transition relation: expressed as EPR formula TR(V, V'), e.g.:

$\exists n,s.\ \text{“}s = \textbf{next}(n)\text{”} \land \forall x,y.\ \textbf{pending}'(x,y) \leftrightarrow (\textbf{pending}(x,y) \lor (x=\textbf{id}[n] \land y=s))$

$\lor\ \exists n.\ \textbf{pending}(\textbf{id}[n],n) \land \forall x.\ \textbf{leader}'(x) \longleftrightarrow (\textbf{leader}(x) \lor x=n)$

…

Modeling with EPR

- **State**: finite first-order structure over vocabulary V (+ axioms)

- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall$ id, n. $\neg$**pending**(id, n)
  - Bad(V) – bad states, e.g., $\exists\, n_1, n_2.\ \textbf{leader}(n_1) \land \textbf{leader}(n_2) \land n_1 \neq n_2$

- Transition relation: expressed as EPR formula TR(V, V'), e.g.:

  ∃n,s. "s = **next**(n)" ∧ ∀x,y. **pending'**(x,y) ↔ (**pending**(x,y) ∨ (x=**id**[n]∧y=s))

  ∨ ∃n. **pending** (**id**[n],n) ∧ ∀x. **leader'**(x) ⟷ (**leader**(x) ∨ x=n)

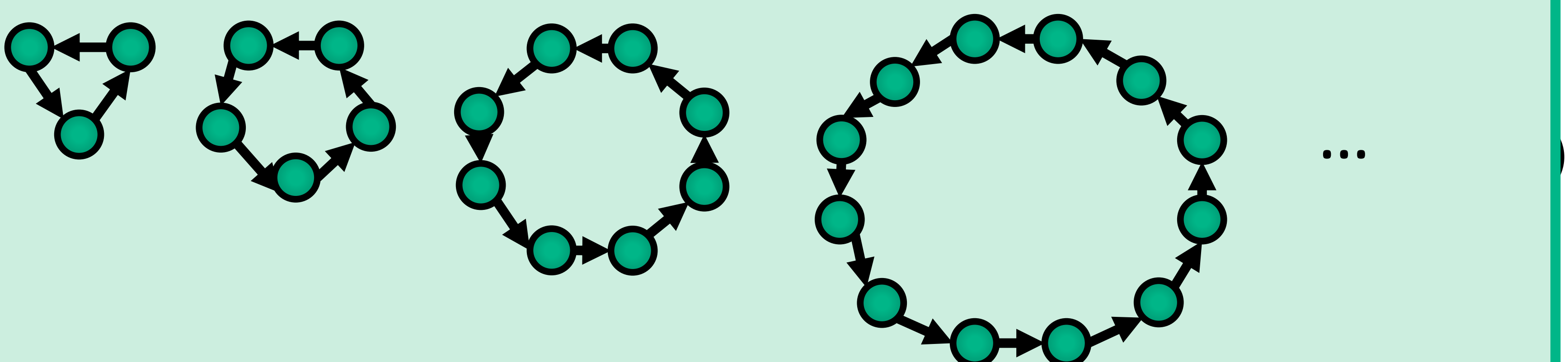
  …

Modeling with EPR

- State: finite first-order structure over vocabulary V (+ axioms)

- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall$ id, n. $\neg$**pending**(id, n)
  - Bad(V) – bad states, e.g., $\exists\, n_1, n_2.$ **leader**$(n_1) \wedge$ **leader**$(n_2) \wedge n_1 \neq n_2$

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



Using EPR for Verification

- System model $Init(V), Bad(V), TR(V, V') \in EPR$
- Inductive invariant $Inv(V) \in \forall^*$
- Verification conditions

Initiation	$Init(V) \Longrightarrow Inv(V)$	unsat($Init(V) \wedge \neg Inv(V)$)
Cons.	$Inv(V) \wedge TR(V,V') \Longrightarrow Inv(V')$	unsat($Inv(V) \wedge TR(V,V') \wedge \neg Inv(V')$)
Safety	$Inv(V) \Longrightarrow \neg Bad(V)$	unsat($Inv(V) \wedge Bad(V)$)

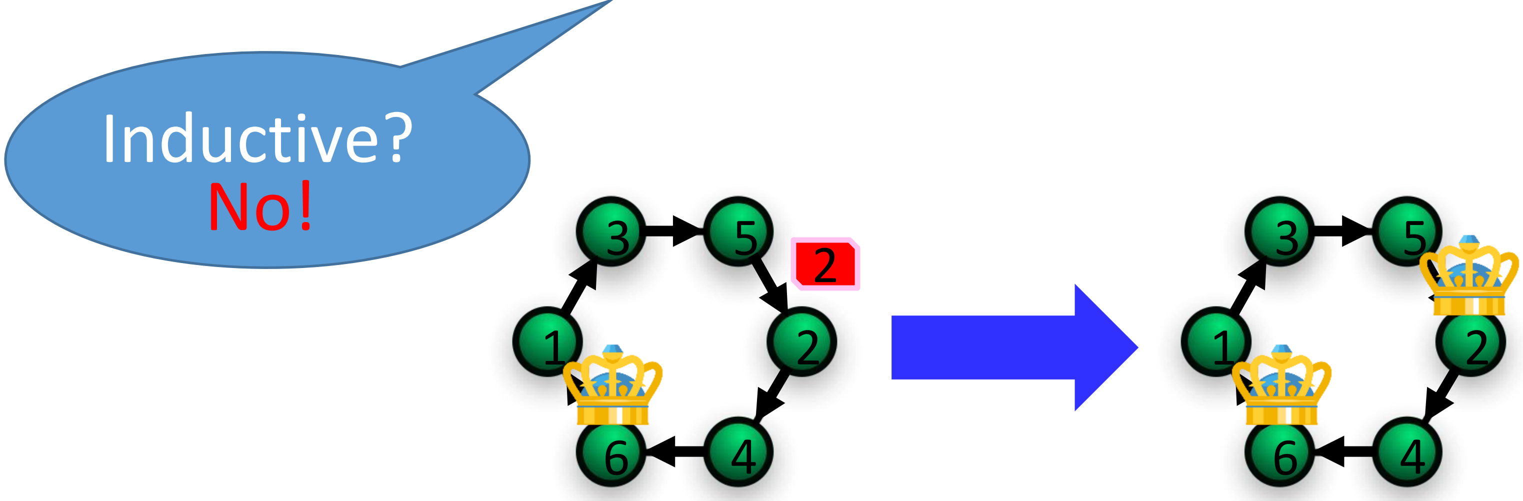
Verification conditions ∈ EPR

➔ Decidable to check

Inductive Invariant for Leader Election

Safety property:

I_0 = ¬Bad = ∀x,y: Node. **leader**(x) ∧ **leader**(y) → x = y



- ≤ (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

Inductive Invariant for Leader Election

Safety property:

I_0 = ¬Bad = ∀x,y: Node. **leader**(x) ∧ **leader**(y) → x = y

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

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

The leader has the highest id

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

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

The reason for using "btw" instead of "next"

- ≤ (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

Inductive Invariant for Leader Election

Safety property:

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

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

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

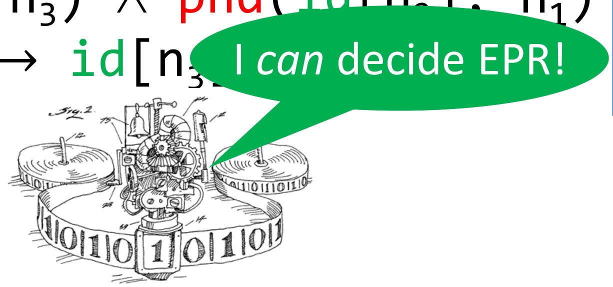
The leader has the highest id

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

Only highest id can be self-pnd

$I_3 = \forall n_1,n_2,n_3\text{: Node. } \mathrm{btw}(n_1,n_2,n_3) \land \mathrm{pnd}(\mathrm{id}[n_{\ldots}], n_1)$
$\to \mathrm{id}[n_3 \ldots$

Cannot bypass higher nodes



$Init(V) \land \neg Inv(V)$
$Inv(V) \land TR(V,V') \land \neg Inv(V')$
$Inv(V) \land Bad(V)$

→ **EPR Solver** → Proof

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.** $btw(x, y, z) \rightarrow btw(y, z, x)$ Circular shifts **∀x, y, z, w: Node.** $btw(w, x, y) \land btw(w, y, z) \rightarrow btw(w, x, z)$ Transitive **∀x, y, w: Node.** $btw(w, x, y) \rightarrow \neg btw(w, y, x)$ Anti-Symmetric **∀x, y, z, w: Node.** $distinct(x, y, z) \rightarrow btw(w, x, y) \lor btw(w, y, x)$
		"`next(a)=b`" ≡ **∀x: Node.** $x \neq a \land x \neq b \rightarrow$ `btw(a,b,x)`

So far

Formal specification with EPR

- Surprisingly expressive
  - Integers: numeric id’s expressed with $\leq$
  - Transitive closure: ring topology expressed with btw
  - Network semantics: pending messages
  - Sets and cardinalities (for consensus protocols) [OOPSLA’17]
  - Liveness properties [POPL’18, FMCAD’18]
  - Implementations [PLDI’18]

Not in this talk

Next

Invariant inference: finding inductive invariants

(1) Automatically

- Adapt techniques from finite-state model checking (PDR)

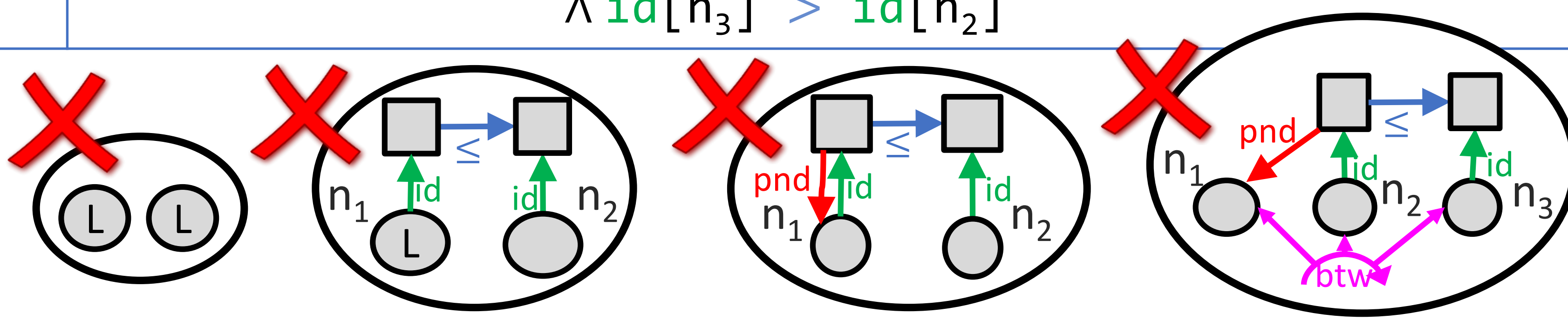
(2) Interactively

- Based on graphically displayed counterexamples to induction

How can we find a universally quantified inductive invariant?

Inductive Invariant for Leader Election

I_0 $\neg$Bad	$\forall n_1,n_2 : \text{Node}.\ \textbf{leader}(n_1) \land \textbf{leader}(n_2) \rightarrow n_1 = n_2$ $\neg\exists n_1,n_2 : \text{Node}.\ \textbf{leader}(n_1) \land \textbf{leader}(n_2) \land n_1 \neq n_2$	At most one leader elected
I_1	$\forall n_1,n_2 : \text{Node}.\ \textbf{leader}(n_1) \rightarrow id[n_2] \leq id[n_1]$ $\neg\exists n_1,n_2 : \text{Node}.\ \textbf{leader}(n_1) \land id[n_2] > id[n_1]$	The leader has the highest id
I_2	$\forall n_1,n_2 : \text{Node}.\ pnd(id[n_1],n_1) \rightarrow id[n_2] \leq id[n_1]$ $\neg\exists n_1,n_2 : \text{Node}.\ pnd(id[n_1],n_1) \land id[n_2] > id[n_1]$	Only highest id can be self-pnd
I_3	$\forall n_1,n_2,n_3 : \text{Node}.\ btw(n_1,n_2,n_3) \land pnd(id[n_2],n_1) \rightarrow id[n_3] \leq id[n_2]$ $\neg\exists n_1,n_2,n_3 : \text{Node}.\ btw(n_1,n_2,n_3) \land pnd(id[n_2],n_1) \land id[n_3] > id[n_2]$	Cannot bypass higher nodes



Construct Inv by excluding “bad” states

1. How to find these states?
2. How to generalize into conjectures?

Generalization using Diagram

Use **diagrams** as generaliza

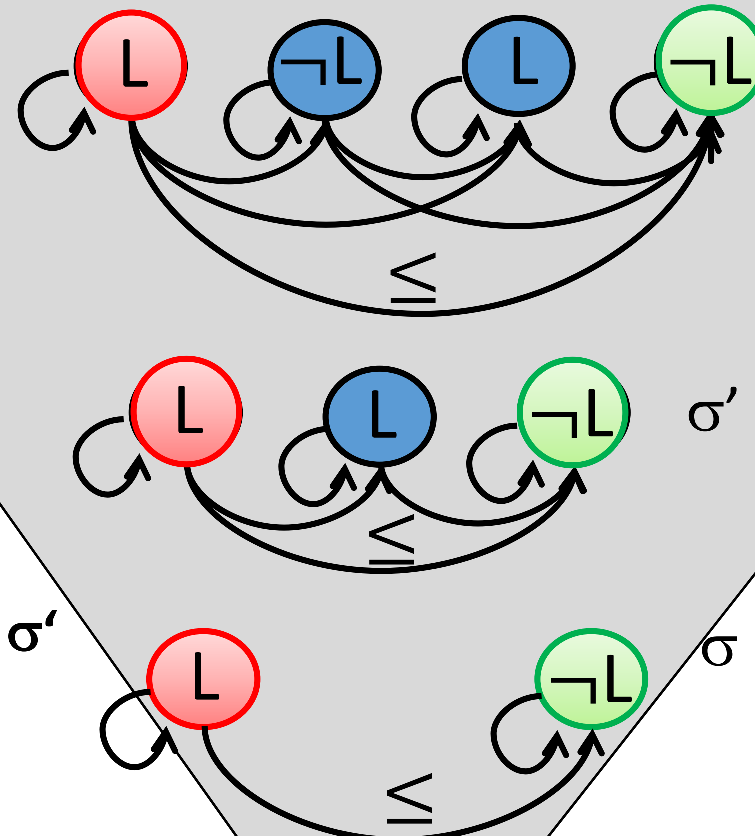
- state σ is a finite first-or

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

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

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

exclude(σ) = ¬Diag(σ)



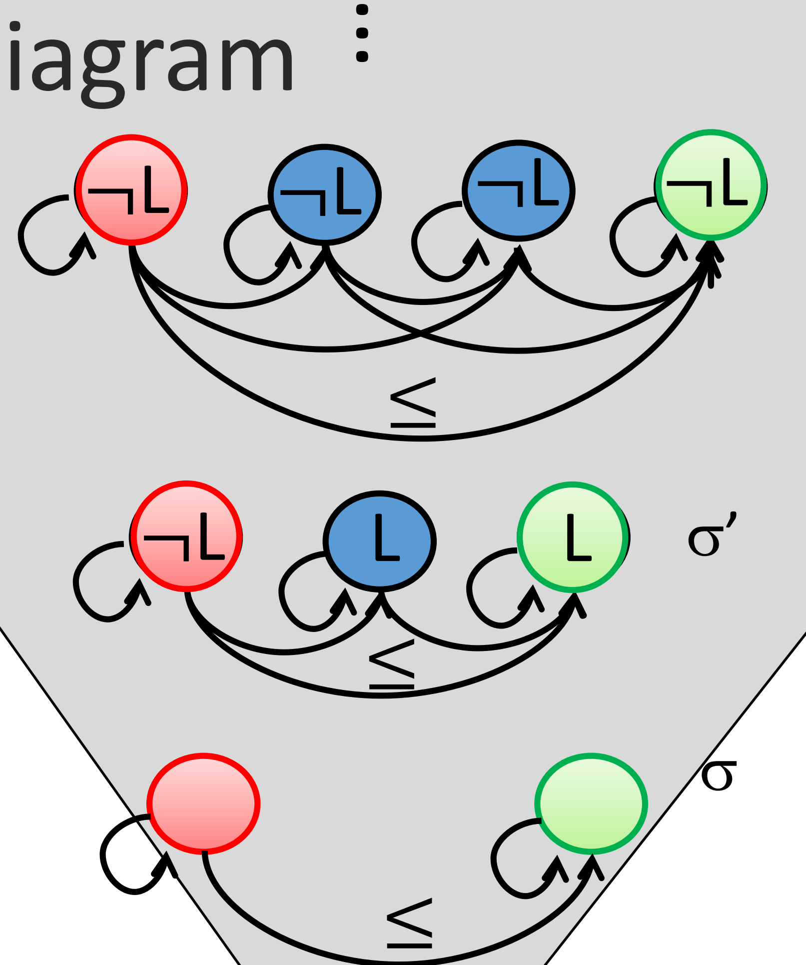
[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Proving Their Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

Generalization using Diagram

Generalize even more if σ is a partial structure

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

$$\mathrm{exclude}(\sigma) = \neg\mathrm{Diag}(\sigma)$$

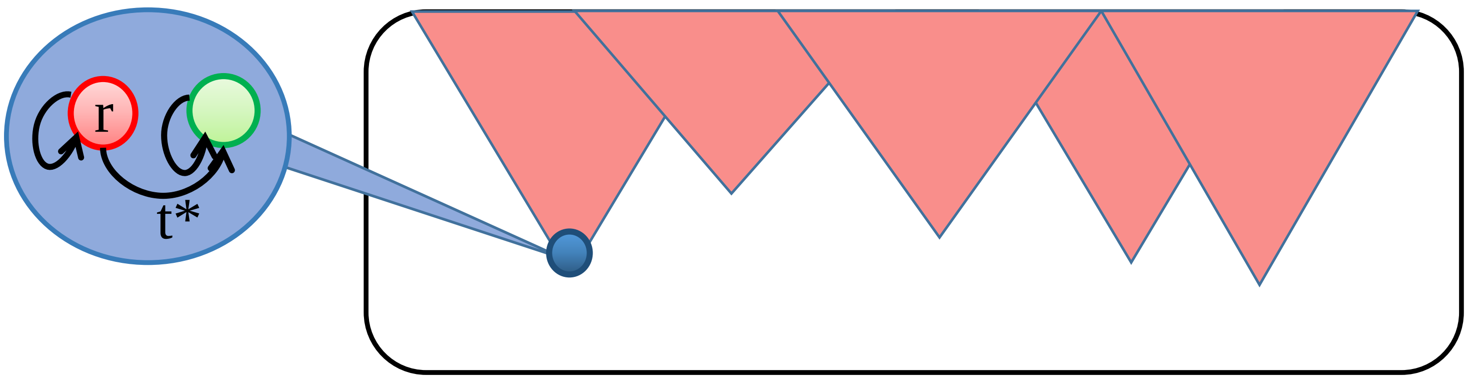


[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Proving Their Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

$\forall^*$ Invariant - excluded substructures

$\text{Inv} \equiv \forall\bar{x}.\ (\text{l}_{1,1}(\bar{x}) \vee \ldots \vee \text{l}_{1,m}(\bar{x})) \wedge \ldots \wedge \forall\bar{x}.\ (\text{l}_{n,1}(\bar{x}) \vee \ldots \vee \text{l}_{n,m}(\bar{x}))$

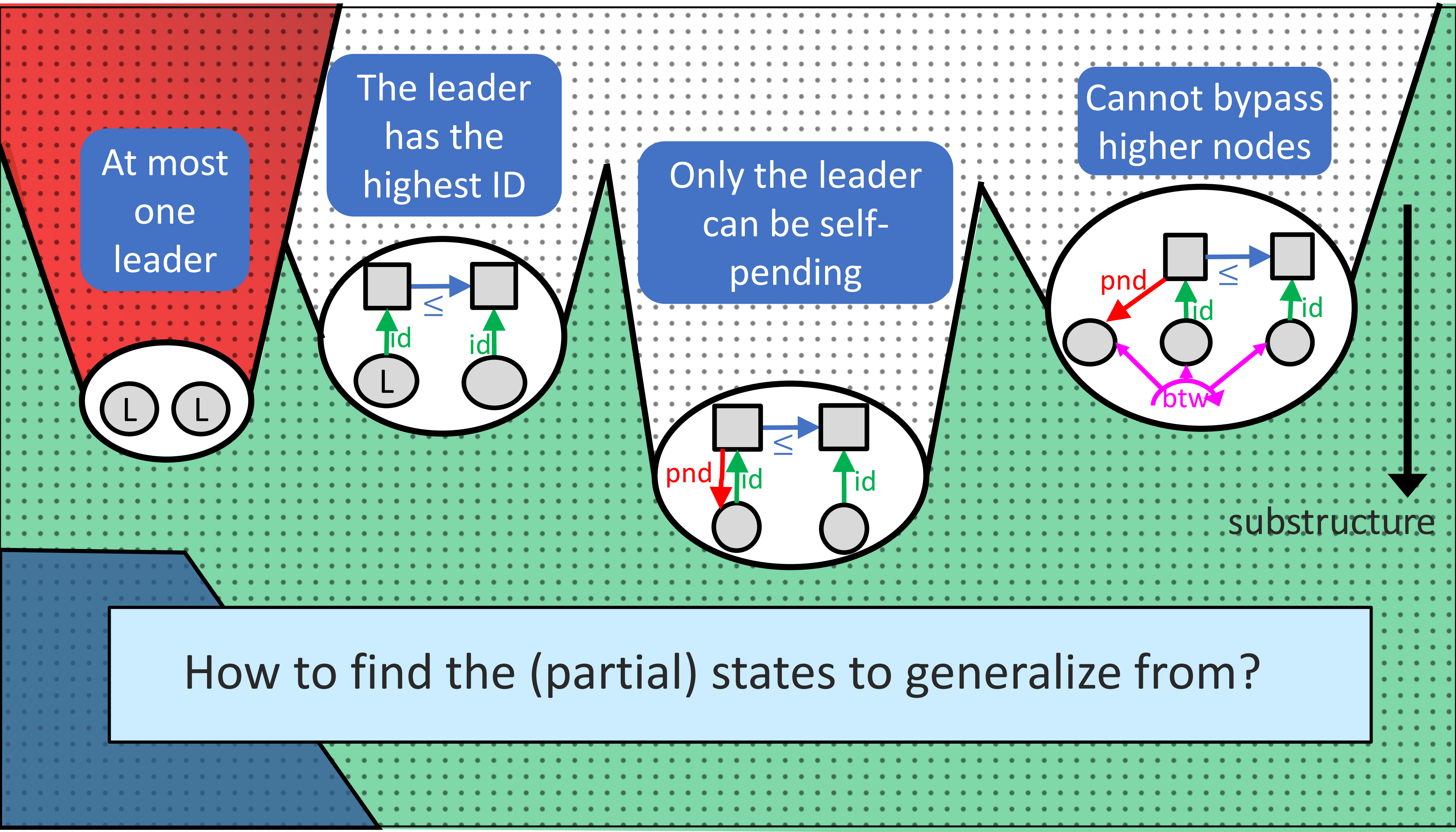
clause / conjecture



$\text{Inv} \equiv \neg\,\exists\bar{x}.\ (\neg\text{l}_{1,1}(\bar{x}) \wedge \ldots \wedge \neg\text{l}_{1,m}(\bar{x})) \wedge \ldots \wedge \neg\exists\bar{x}.\ (\neg\text{l}_{n,1}(\bar{x}) \wedge \ldots \wedge \neg\text{l}_{n,m}(\bar{x}))$

cube

Leader election example



How to find the (partial) states to generalize from?

(1) Automatic inference: UPDR

- Based on Bradley's IC3/PDR [VMCAI11,FMCAD11]
  - SAT-based verification of finite-state systems
- Abstracts concrete states using their logical diagram
- Backward traversal performed over diagrams
- Blocking of CTI excludes a *generalization* of its diagram ➔ generates universally quantified lemmas

---

- [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.
- [VMCAI'17] Property Directed Reachability for Proving Absence of Concurrent Modification Errors, A. Frumkin, Y. Feldman, O. Lhoták, O. Padon, M. Sagiv and S. Shoham.

UPDR: Possible outcomes

- Universal inductive invariant found
  - System is safe

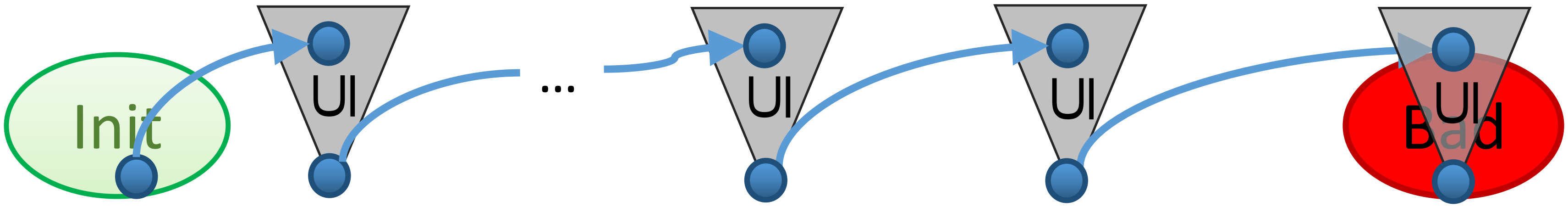
Used to infer inductive invariants / procedure summaries of:

- Heap-manipulating programs, e.g.
  - Singly/Doubly/Nested linked list
  - Iterators in Java - Concurrent modification error (CME)
- Distributed protocols
  - Spanning tree
  - Learning switch
  - …

No need for user-defined predicates/ templates!

UPDR: Possible outcomes

- Universal inductive invariant found
  - System is safe
- Proof that no universal inductive invariant exists
  - Safety not determined*



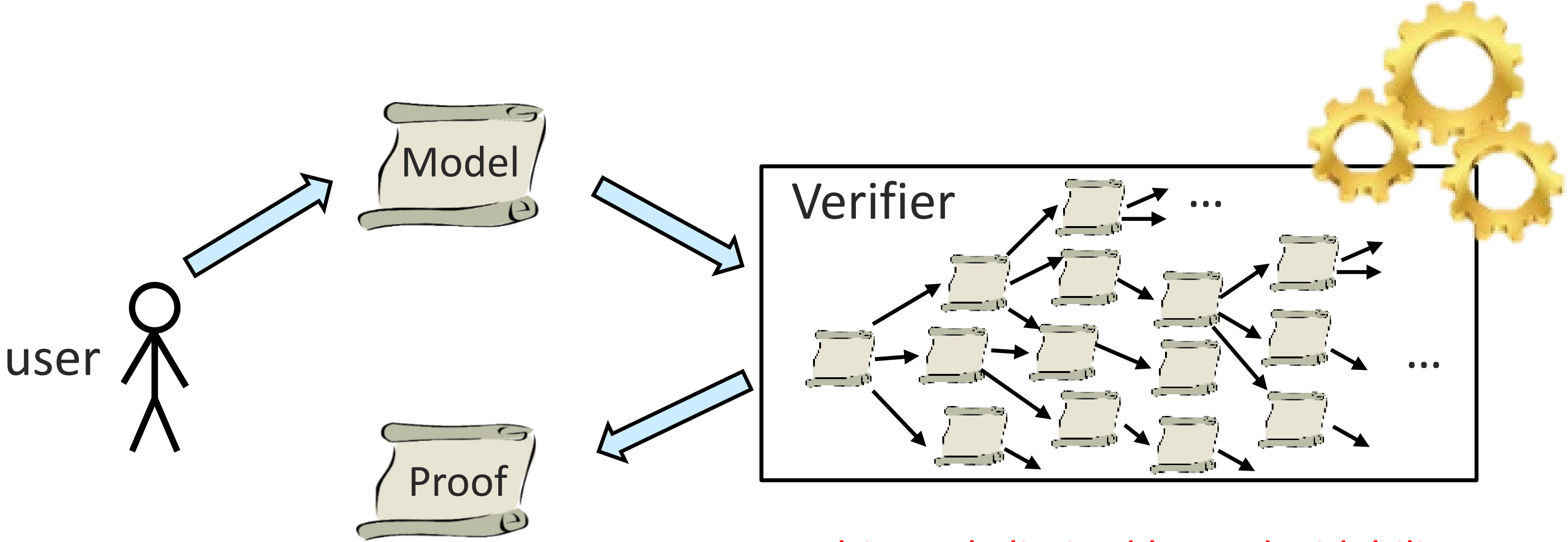
* can use Bounded Model Checking to find real counterexamples

UPDR: Possible outcomes

- Universal inductive invariant found
  - System is safe
- Proof that no universal inductive invariant exists
  - Safety not determined*
- Divergence
  - In general, inferring universal ind. inv. is undecidable
  - For linked lists it is decidable, UPDR will also terminate
    - Proof uses well-quasi-order and Kruskal's tree theorem

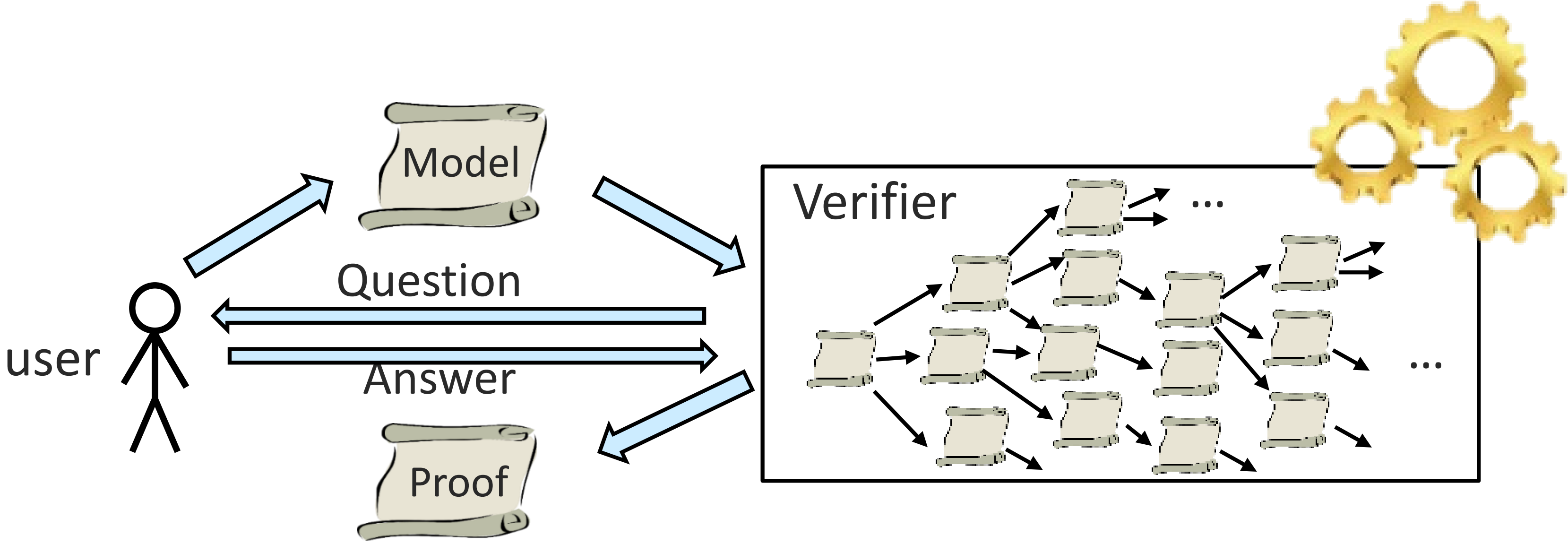
- [POPL'16] Decidability of Inferring Inductive Invariants, O. Padon, N. Immerman, S. Shoham, A. Karbyshev, and M. Sagiv.

Automatic Inference (e.g., UPDR)



Ultimately limited by undecidability

(2) Interactive Inference



- Let the user guide the tool
  - User has intuition about the essence of the proof
  - Computer is good at handling corner cases

Interactive Inference

Expressiveness

Deductive verification

Ultimately limited by human

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

What is the human's role?
What is the machine's role?
How do they interact?

Ultimately limited by undecidability

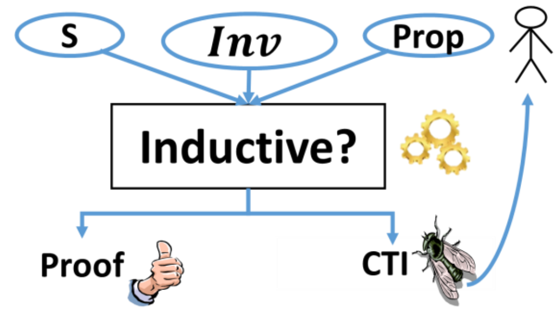
Model Checking
Static Analysis

Automation

Supervised Verification of Infinite-State Systems

Ivy: Interactive Generalization

$Inv = I_0 \wedge \cdots \wedge I_k$



Displays "minimal" CTI to exclude

Generalizes to a partial state

- removes "irrelevant" facts (graphical interface - checkboxes)

Translates to universally quantified conjecture (via diagram)

Provides auxiliary automated checks:

1. BMC(K): uses SAT solver to check if conjecture is true up to K
   - User determines the right K to use
2. ITP(K): uses SAT solver to discover more facts to remove

Examines the proposed conjecture – it could be wrong

Adds I_{k+1}

[PLDI'16] IVy: Safety Verification by Interactive Generalization. O. Padon, K. McMillan, A. Panda, M. Sagiv, S. Shoham https://github.com/Microsoft/ivy

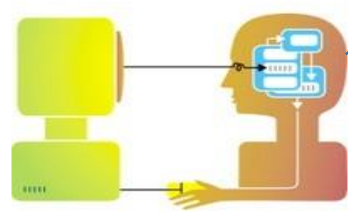
Interactive Verification in IVy

Check inductiveness

BMC

Interpolation

⋮



Projection of relevant facts

BMC bounds

Examining conjectures

⋮

Decidable Problems

Predictable Automation

Proof intuition and creativity

Graphical interaction

Summary 1

Verification with decidable logic

- EPR - decidable fragment of FOL
  - Deduction is decidable
  - Finite counterexamples

- Domain knowledge and axioms
- Derived relations
- Modularity
- Prophecy

- Can be made surprisingly powerful
  - Transitive closure: linked lists, ring topology [PLDI’16]
  - Paxos, Multi-Paxos, [OOPSLA’17]
  - Liveness and Temporal Properties [POPL’18]
  - Developing verified implementations [PLDI’18]

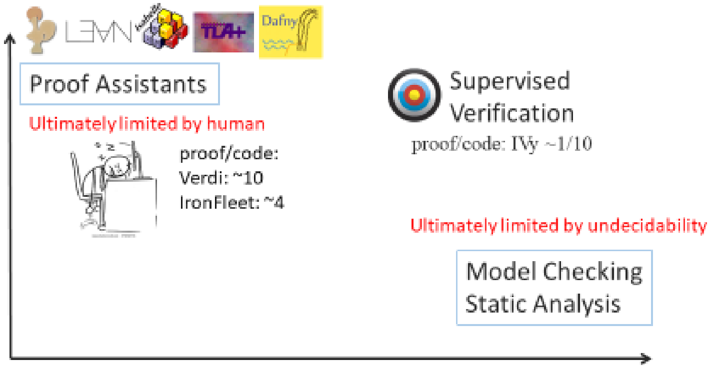
Summary 2

Invariant Inference

- Automatic inference: UPDR [CAV'15,JACM]
- Interactive inference: Ivy [PLDI'16]

- Use logical diagram to infer Inv ∈ $\forall^*$
- Can also prove absence of Inv ∈ $\forall^*$

Take away



- Decidable logic is useful
  - facilitates automation
- We need ways to guide verification tools

- How to divide the problem between human and machine?
- Different inference schemes
- Different Forms of interaction
- Other logics
- Theoretical understanding of limitations and tradeoffs

Supervised Verification of Infinite-State Systems