Interactive Verification of Distributed Protocols Using Decidable Logic

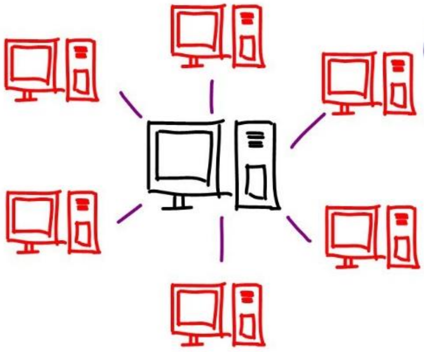
Sharon Shoham, Tel Aviv University

Static Analysis Symposium, 2018

erc Supervised Verification of Infinite-State Systems

Why verify distributed protocols?

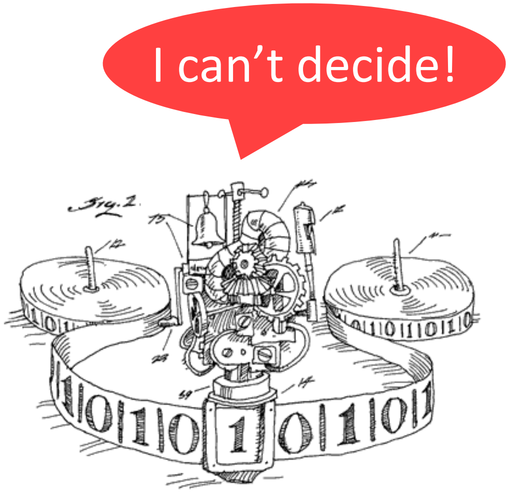
- Distributed systems are everywhere
  - Safety-critical systems
  - Cloud infrastructure



- 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

Verifying distributed protocols is hard

- Infinite state-space
  - unbounded number of threads
  - unbounded number of messages
  - unbounded number of objects
- Asymptotic complexity of verification
  - Rice theorem
  - The ability of simple programs to represent complex behaviors



State of the art in formal verification

- Automatic techniques
  - Abstract Interpretation
  - Model checking

  Limited for infinite state systems due to undecidability

- Deductive techniques
  - SMT-based deduction + manual program annotations (e.g. Dafny)
    - Requires programmer effort to provide inductive invariants
    - SMT solver may diverge (matching loops, arithmetic)
    - Unpredictability, butterfly effect
  - 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

Proof Assistants

Ultimately limited by human

proof/code:

Verdi: ~10

IronFleet: ~4

Ultimately limited by undecidability

Model Checking

Static Analysis

Expressiveness

Automation

State of the art in formal verification

Expressiveness

Proof Assistants

Ultimately limited by human

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

Interactive Verification

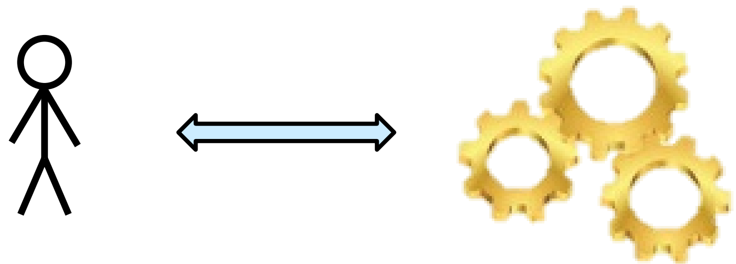
Ultimately limited by undecidability

Model Checking
Static Analysis

Automation

Supervised Verification of Infinite-State Systems

Interactive Verification



Goals

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

Questions

- What is the role of the human?
- What is the role of the machine?
- How do they interact?

This talk

Interactive verification by

(1) Deductive verification with decidable logic
   - Interaction based on candidate inductive invariants & counterexamples to induction

(2) Interactive inference of universal invariants
   - Fine-grained interaction based on counterexamples to induction & diagrams

(3) User-guided inference of phase invariants
   - Coarse-grained interaction based on phase sketches & relaxed traces

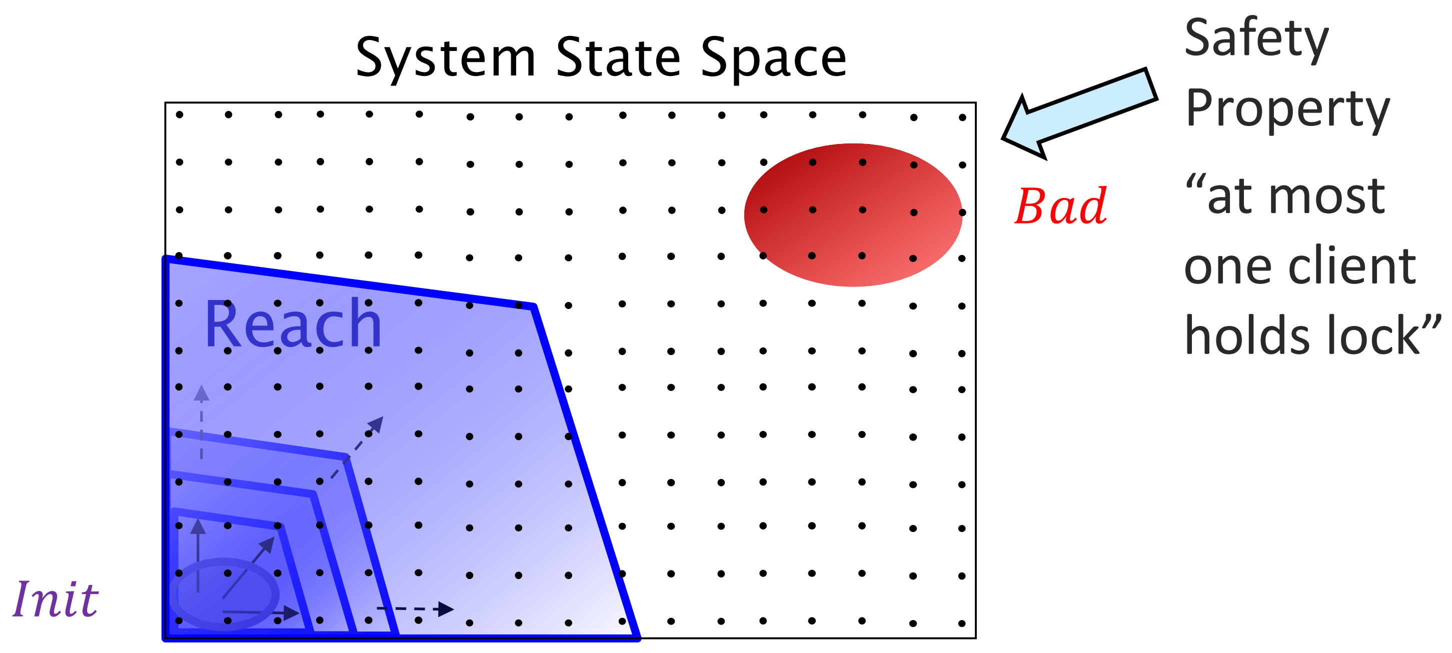
Realization in Ivy https://github.com/Microsoft/ivy

(2)
- [PLDI'16] IVy: Safety Verification by Interactive Generalization. O. Padon, K. McMillan, A. Panda, M. Sagiv, S. Shoham

(1)
- [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
- [PLDI'18] Modularity for decidability of deductive verification with applications to distributed systems. M. Taube, G. Losa, K. McMillan, O. Padon, M. Sagiv, S. Shoham, J. Wilcox, D. Woos

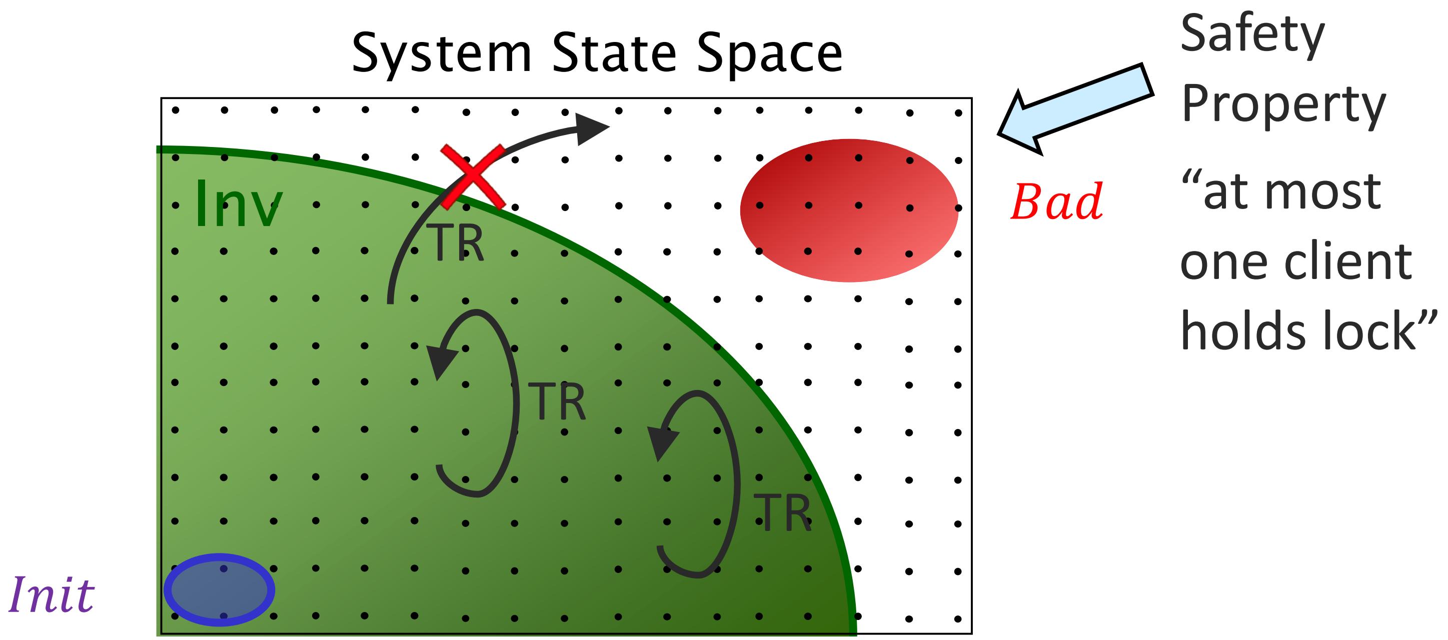
(3)
- [sub] Inferring Phase Invariants from Phase Sketches. Y. Feldman, J. Wilcox, S. Shoham, M. Sagiv

Safety Verification



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

Inductive Invariants



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

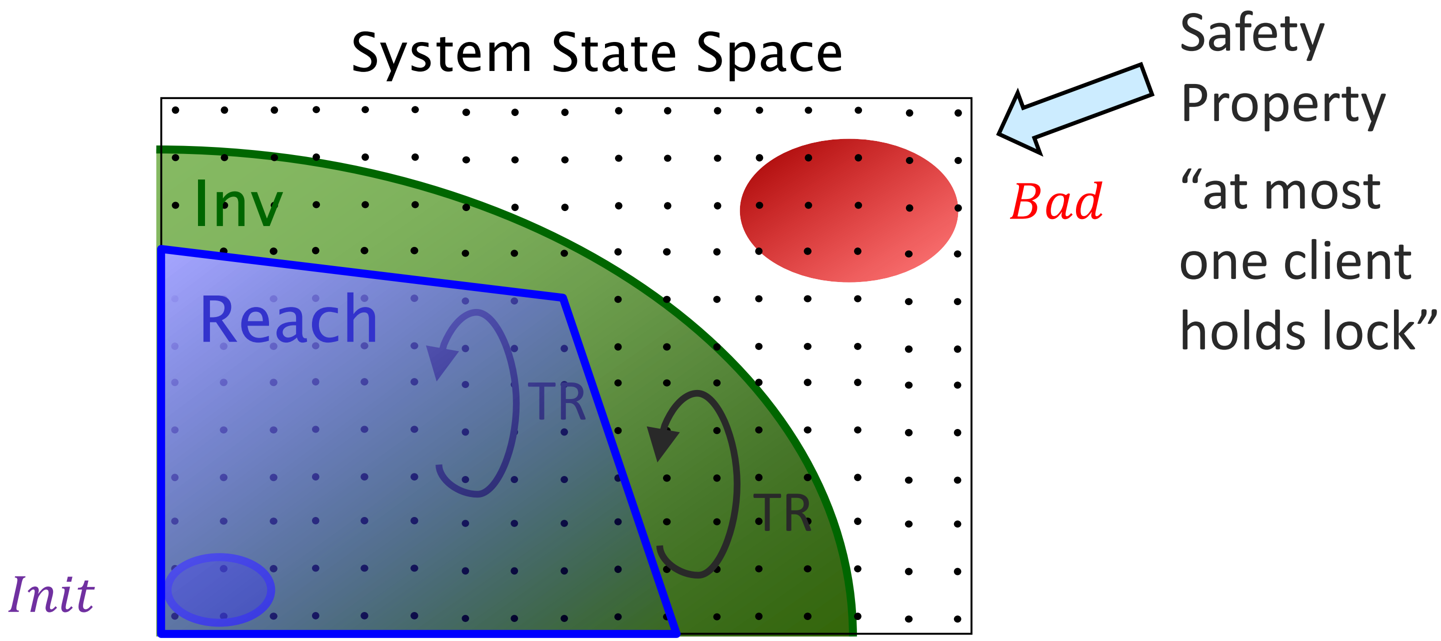
System S is safe iff there exists an **inductive invariant** Inv:

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

$Inv \wedge \text{TR} \Rightarrow Inv'$ **(Consecution)**

$Inv \Rightarrow \neg Bad$ **(Safety)**

Inductive Invariants



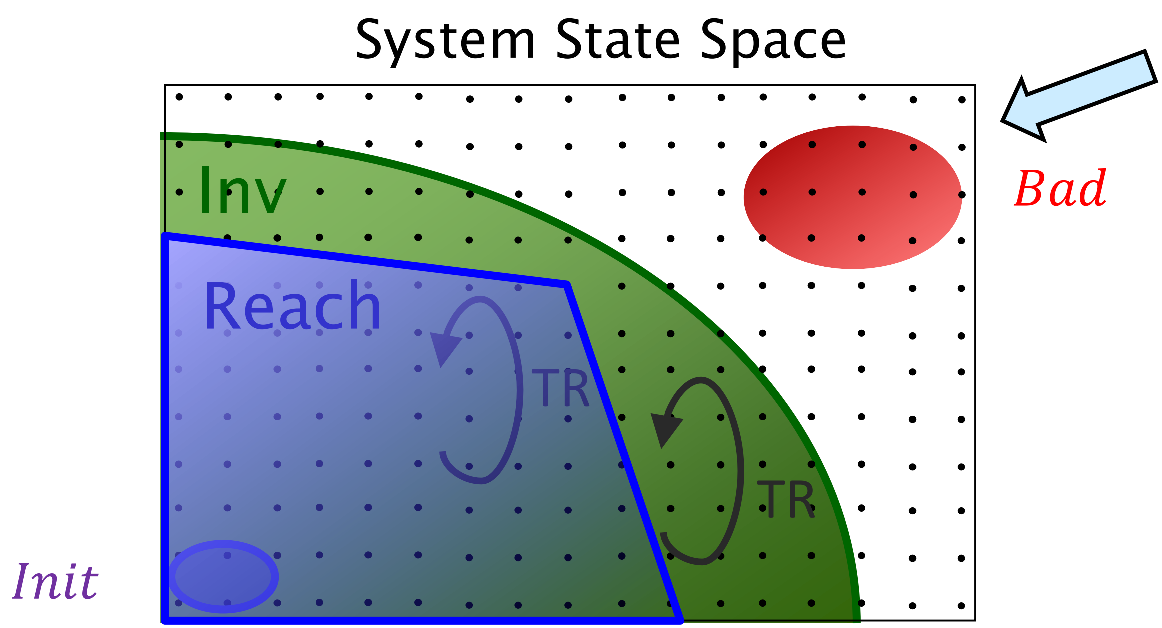
System S is **safe** if all the reachable states satisfy the property P = $\neg Bad$

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

$Init \Rightarrow Inv$	**(Initiation)**
$Inv \wedge TR \Rightarrow Inv'$	**(Consecution)**
$Inv \Rightarrow \neg Bad$	**(Safety)**

} Verification Conditions (VC)

Inductive Invariants



Safety Property
"at most one client holds lock"

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:

$Init \Rightarrow Inv$	**(Initiation)**	$Init \wedge \neg Inv \equiv \bot$	VC
$Inv \wedge TR \Rightarrow Inv'$	**(Consecution)**	$Inv \wedge TR \wedge \neg Inv' \equiv \bot$	
$Inv \Rightarrow \neg Bad$	**(Safety)**	$Inv \wedge Bad \equiv \bot$	

Challenges in Safety Verification

Formal specification: reasoning about infinite-state systems

- Modeling the system, the property and the inductive invariant

Deduction: checking validity of the VCs

- Undecidability of implication checking (unsatisfiability)
    - Unbounded state (threads, messages), arithmetic, quantifiers,…

Inference: inferring inductive invariants (Inv)

- Hard to specify
- Hard to infer automatically
    - Undecidable even when deduction is decidable

Ivy: Restrict VC's to decidable logic

Effectively Propositional Logic – EPR

Decidable fragment of first order logic

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

☺ Allows quantifiers to reason about unbounded sets

- $\forall x,y.\ \mathbf{holds\_lock}(x) \land \mathbf{holds\_lock}(y) \rightarrow x = y$

☺ 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, Karbyshev et al. CAV'15]
- Software-Defined Networks
  [Ball et al. PLDI'14]
- Distributed Protocols
  [Padon et al. PLDI'16, OOPSLA'17, POPL'18, Taube et al. PLDI'18]
- Concurrent Modification Errors in Java
  [Frumkin et al. VMCAI'17]

More in Ken & Oded's tutorial

Challenges for verification with EPR

✓ Formal specification: reasoning about infinite-state systems

- Modeling the system, the property and the inductive invariant **in EPR**

✓ Deduction: checking validity of the VCs

- ~~Un~~decidability of implication checking (unsatisfiability)
  - Unbounded state (threads, messages), arithmetic, quantifiers,…

Inference: inferring inductive invariants (Inv)

- Hard to specify
- Hard to infer automatically
  - Undecidable even when deduction is decidable

Challenges for verification with EPR

✓ Formal specification: reasoning about infinite-state systems

- Modeling the system, the property and the inductive invariant **in EPR**

✓ Deduction: checking validity of the VCs

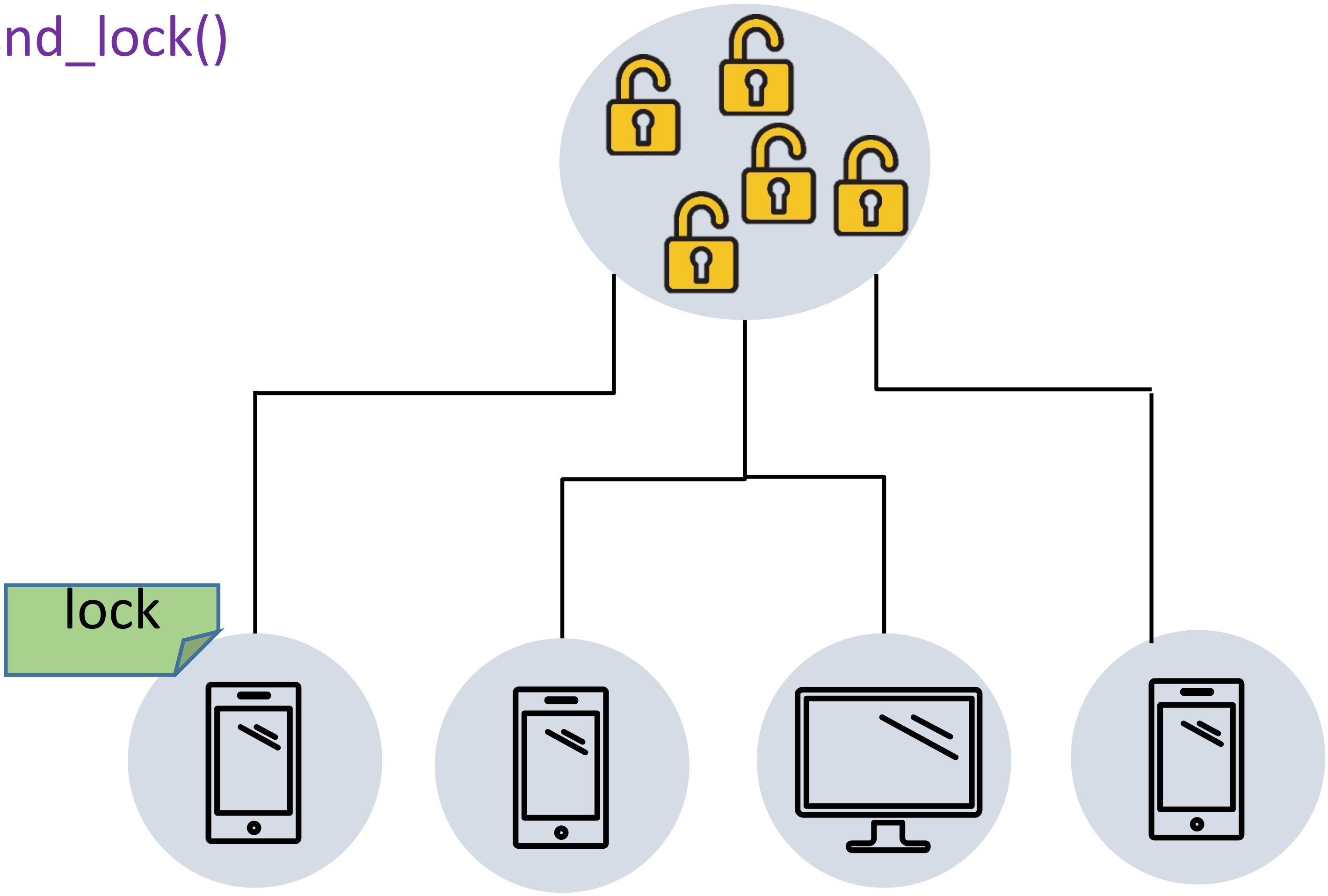
- ~~Un~~decidability of implication checking (unsatisfiability)
  - Unbounded state (threads, messages), arithmetic, quantifiers,…

Inference: inferring inductive invariants (Inv) **interactively**

- Hard to specify
- Hard to infer automatically
  - Undecidable even when deduction is decidable

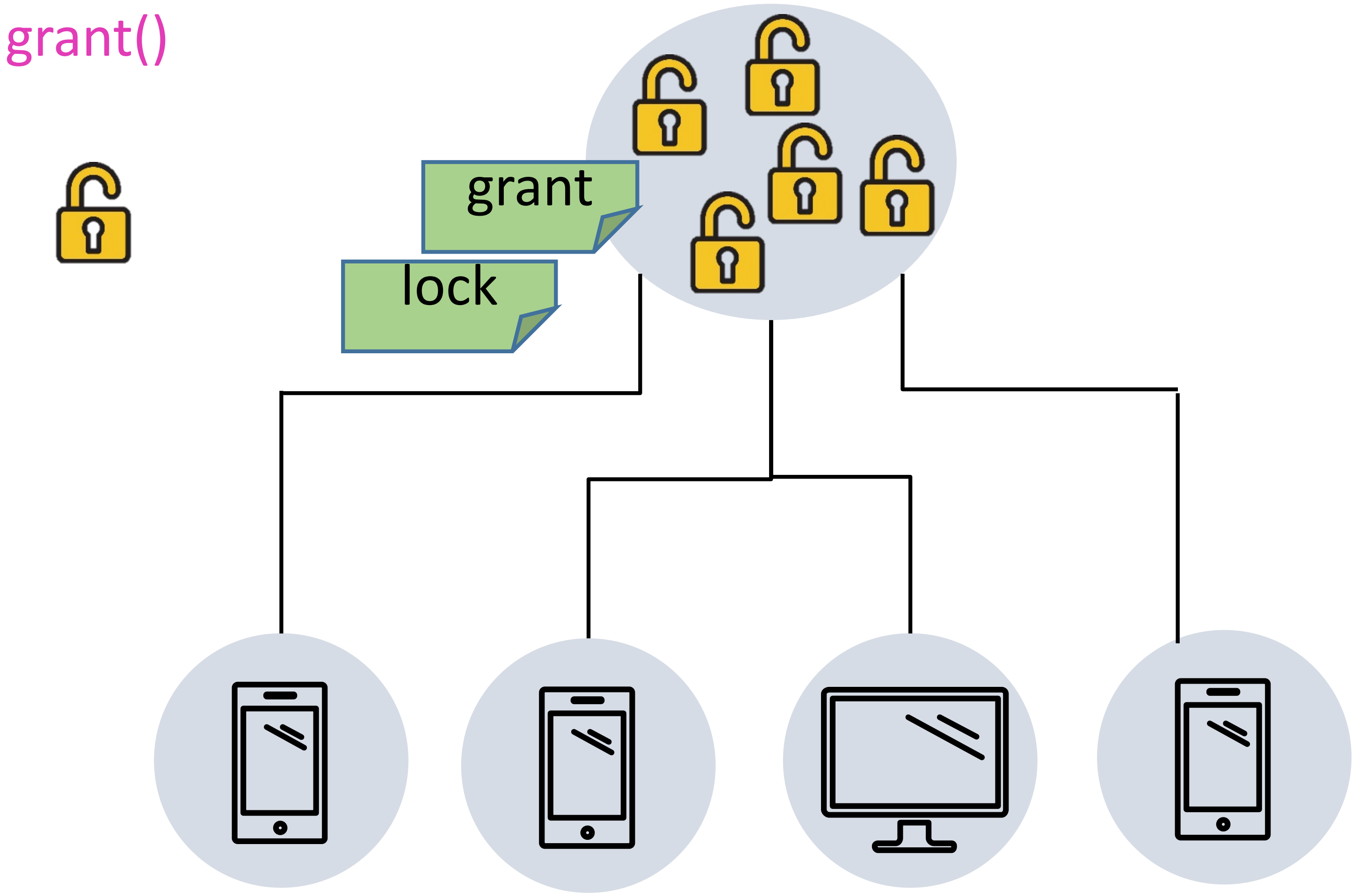
Example: Lock Server

snd_lock()



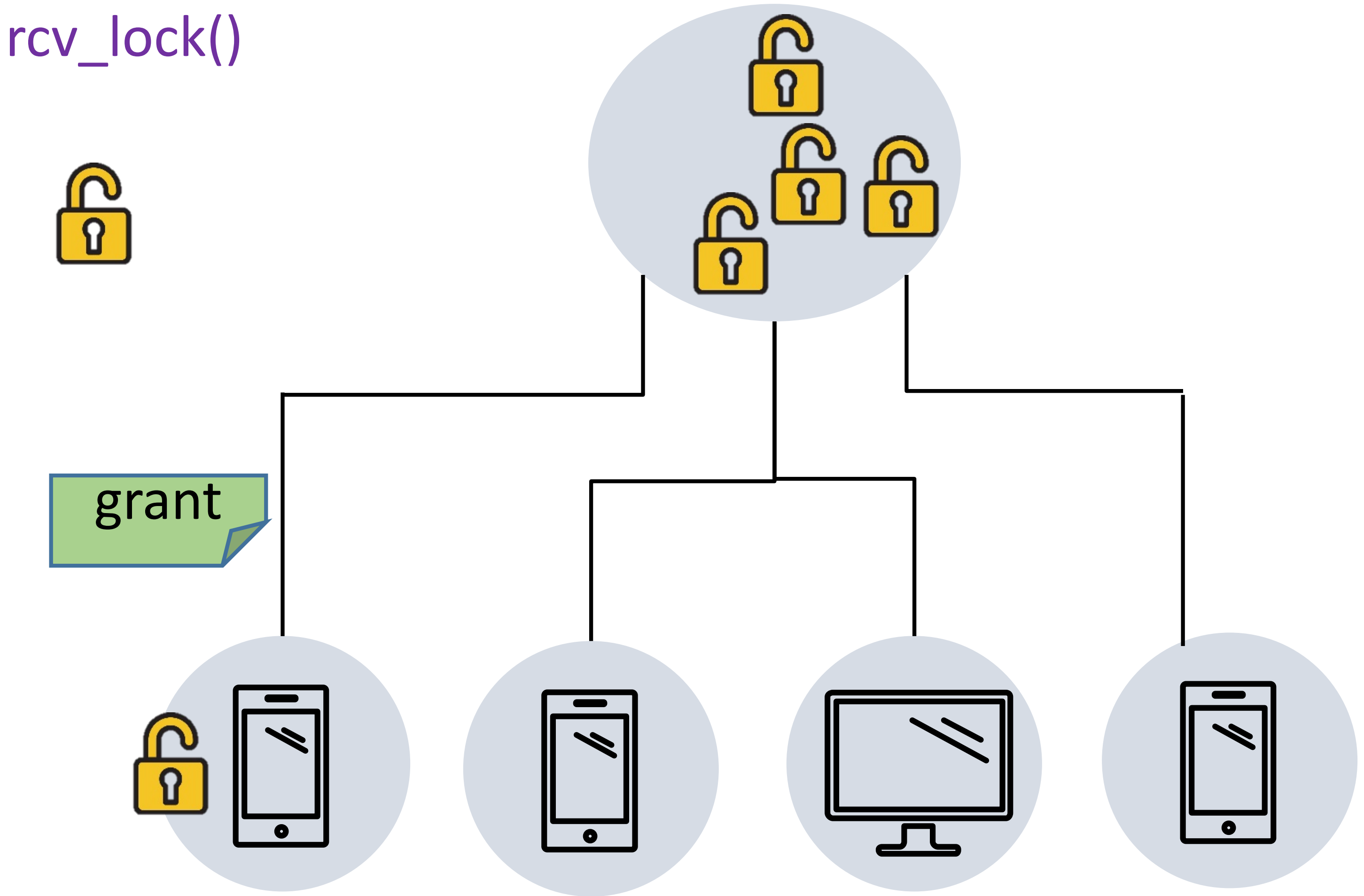
[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

Example: Lock Server



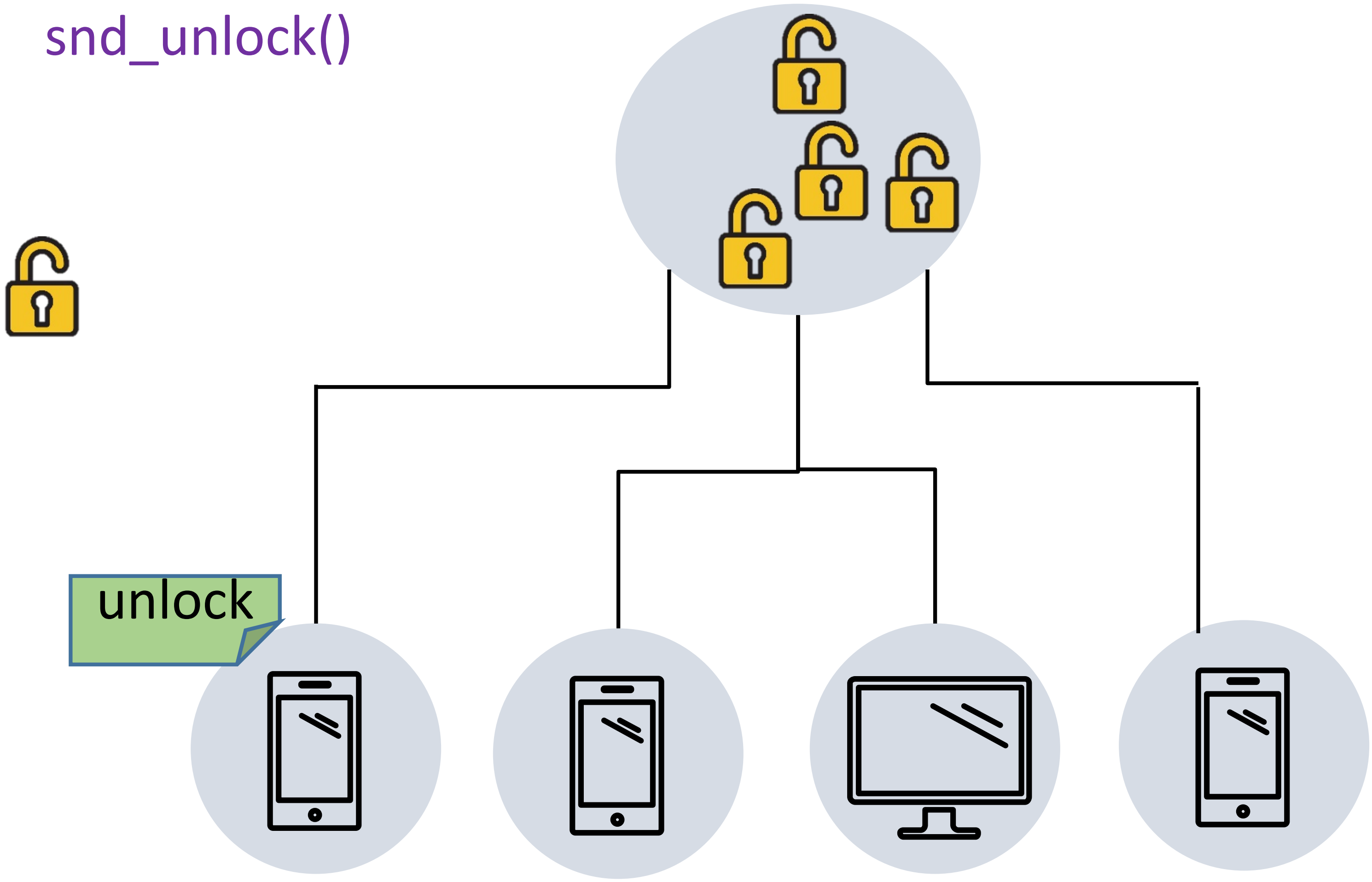
[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

Example: Lock Server



[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

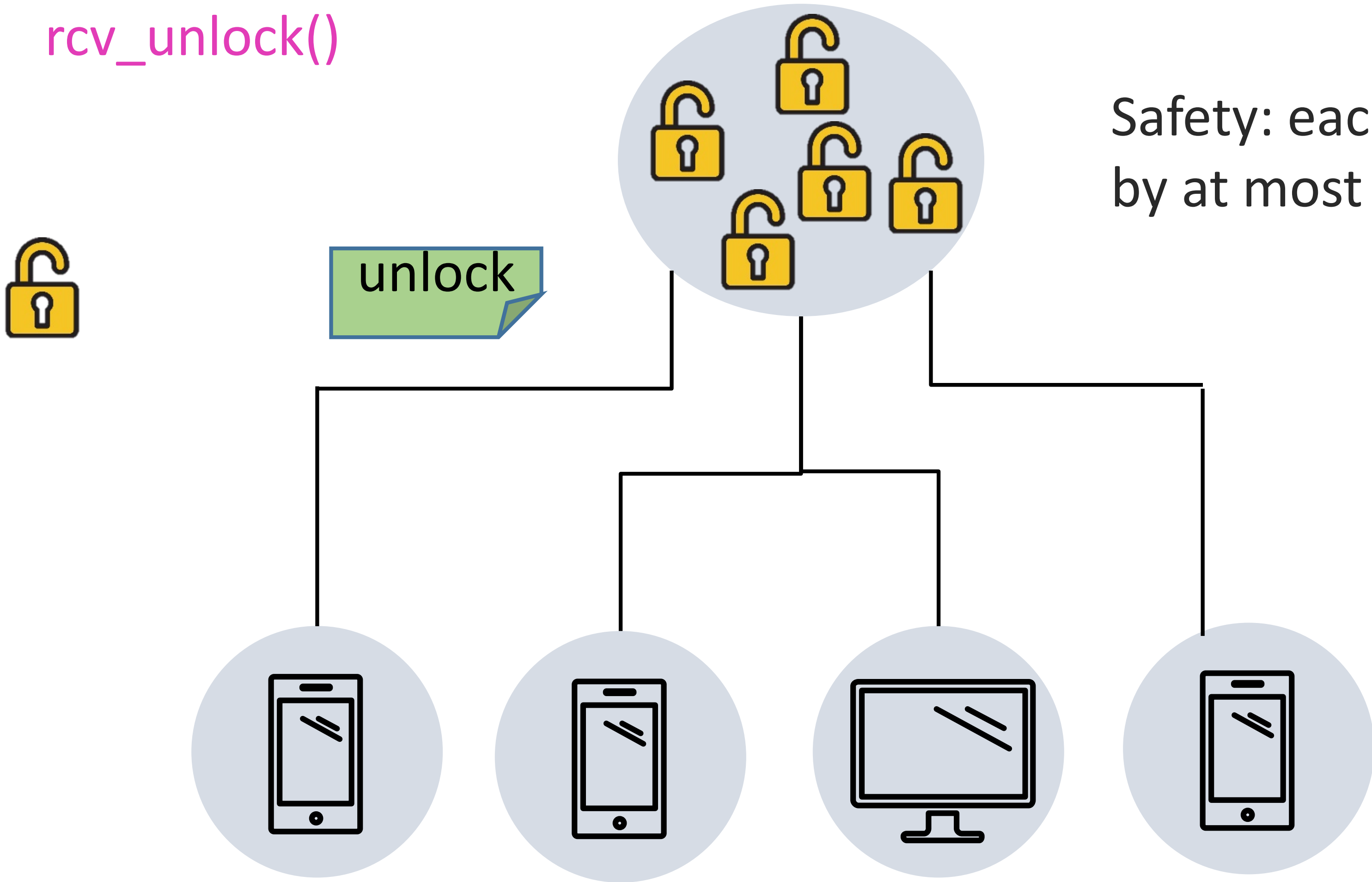
Example: Lock Server



[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

Example: Lock Server

rcv_unlock()



Safety: each lock held by at most one client

[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

Modeling in Ivy (EPR)

server
clients
network

- State: finite first-order structure over vocabulary V
    - free (LOCK)
    - held_by (LOCK, CLIENT)
    - lock_msg (CLIENT, LOCK)
    - grant_msg (CLIENT, LOCK)
    - unlock_msg (CLIENT, LOCK)

Global state of messages in flight

Modeling in Ivy (EPR)

server
clients
network

- State: finite first-order structure over vocabulary V
- Transition relation: EPR formula TR(V, V')

```
action snd_lock(c: C, ℓ: L){
  lock_msg(c, ℓ) := true
}
```

||

```
action grant (c: C, ℓ: L) { … }
```

```
action rcv_lock (c: C, ℓ: L){ … }
```

```
action snd_unlock (c: C, ℓ: L){ … }
```

```
action rcv_unlock(c: C, ℓ: L){ … }
```

$$\exists c, \ell.\ \forall x,y.\ \text{lock\_msg}'(x,y) \leftrightarrow (\text{lock\_msg}(x,y) \lor (x=c \land y=\ell))$$
$$\land\ \text{grant\_msg}'(x,y) \leftrightarrow \text{grant\_msg}(x,y)$$
$$\land\ \text{free}'(y) \leftrightarrow \text{free}(y) \ \ldots.$$
$$\lor\ \exists c, \ell.\ \ldots$$

Modeling in Ivy (EPR)

server
clients
network

- State: finite first-order structure over vocabulary V
- Transition relation: EPR formula TR(V, V')
- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall c, \ell. \neg \text{lock\_msg}(c, \ell)$
  - Bad(V) – bad states, e.g.,

$$\exists\, \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \land \text{held\_by}(\ell, c_2) \land c_1 \neq c_2$$

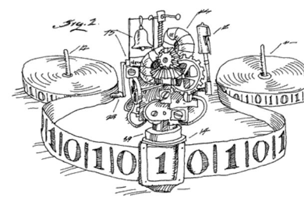
Verification in Ivy (EPR)

server
clients
network

- State: finite first-order structure over vocabulary V
- Transition relation: EPR formula TR(V, V')
- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall c, \ell. \neg \text{lock\_msg}(c, \ell)$
  - Bad(V) – bad states, e.g.,

$$\exists\, \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \wedge \text{held\_by}(\ell, c_2) \wedge c_1 \neq c_2$$

- Inductive invariant s.t. VC ∈ EPR



I *can* decide EPR!

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

→ **EPR Solver** → Proof / CTI

Proof

CTI

server
clients
network

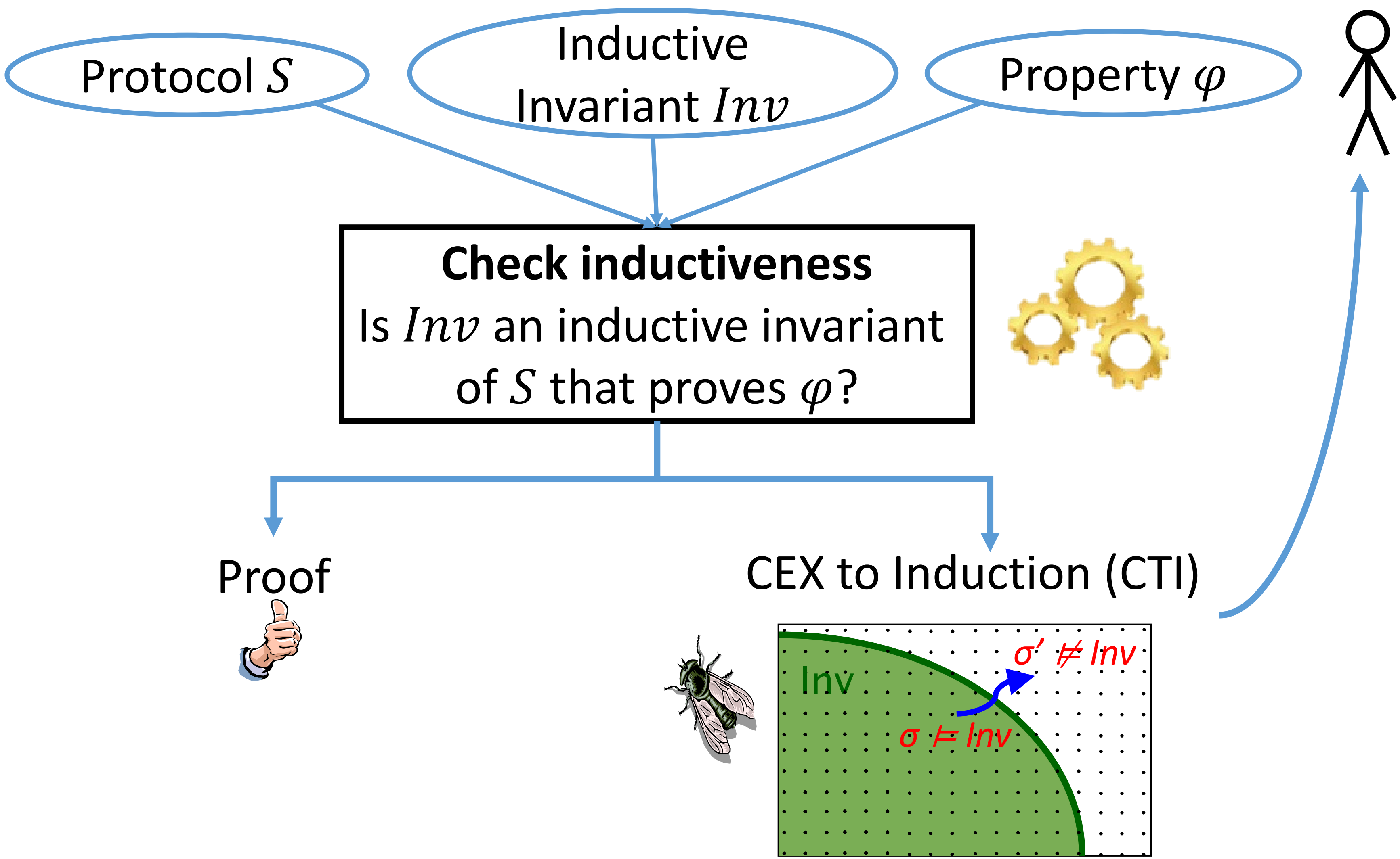
Verification in Ivy (EPR)

- State: finite first-order structure over vocabulary V
- Transition relation: EPR formula TR(V, V')
- Initial states and safety property: EPR formulas over V
  - Init(V) – initial states, e.g., $\forall c, \ell. \neg \text{lock\_msg}(c, \ell)$
  - Bad(V) – bad states, e.g.,

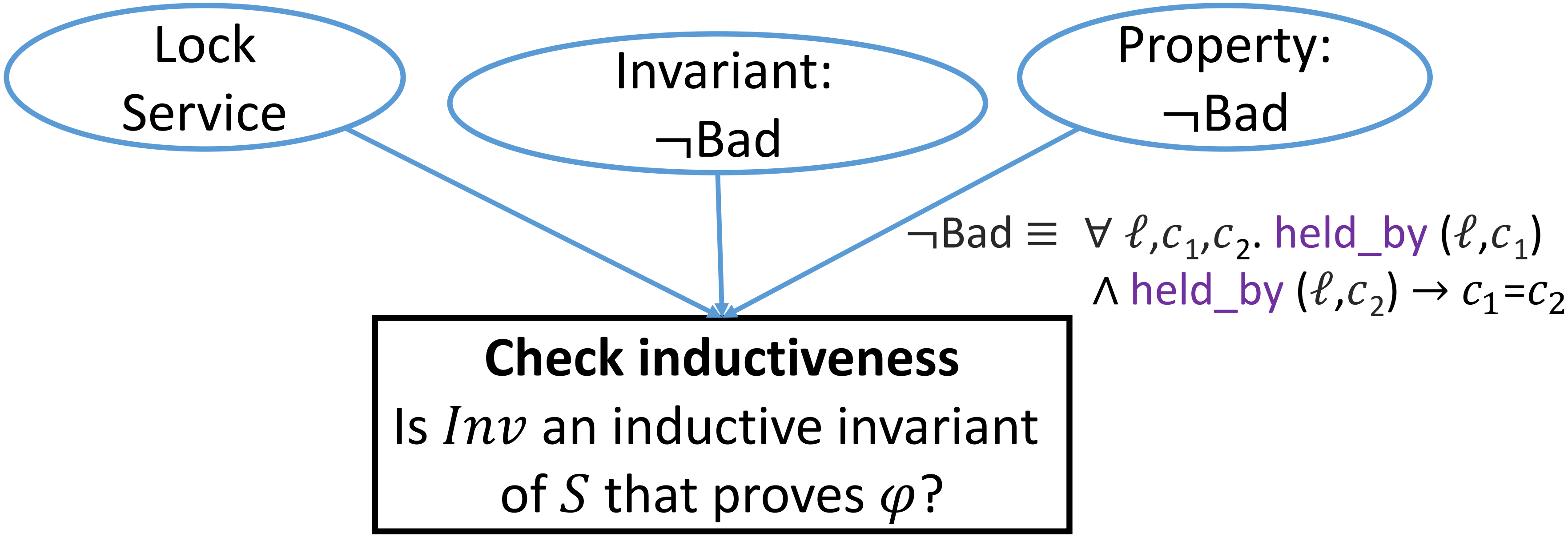
Specify and verify the protocol for any number of clients and locks

Interactive Invariant Inference

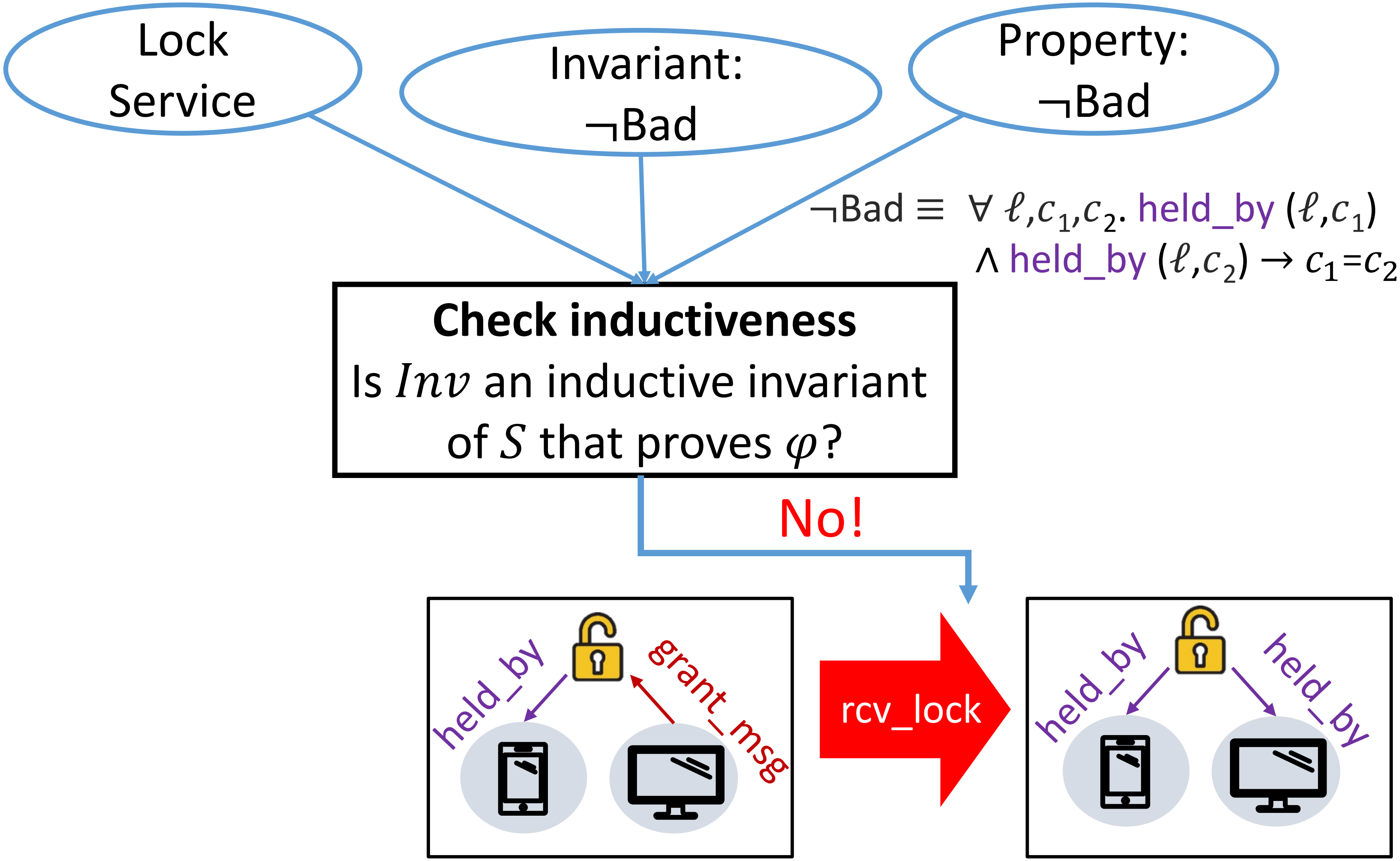
(1) Interaction based on CTIs



Lock Server Example



Lock Server Example



Inductive Invariant for Lock Server

server
clients
network

¬Bad ⫹

$I_0 = \forall\, \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \wedge \text{held\_by}(\ell, c_2) \rightarrow c_1 = c_2$

$I_1 = \forall\, \ell, c_1, c_2.\ \neg(\text{grant\_msg}(c_1, \ell) \wedge \text{held\_by}(\ell, c_2))$

$I_2 = \forall\, \ell, c_1, c_2.\ \neg(\text{unlock\_msg}(c_1, \ell) \wedge \text{held\_by}(\ell, c_2))$

$I_3 = \forall\, \ell, c_1, c_2.\ \neg(\text{unlock\_msg}(c_1, \ell) \wedge \text{grant\_msg}(c_2, \ell))$

$I_4 = \forall\, \ell, c_1, c_2.\ \text{grant\_msg}(c_1, \ell) \wedge \text{grant\_msg}(c_2, \ell) \rightarrow c_1 = c_2$

$I_5 = \forall\, \ell, c_1, c_2.\ \text{unlock\_msg}(c_1, \ell) \wedge \text{unlock\_msg}(c_2, \ell) \rightarrow c_1 = c_2$

$I_6 = \forall\, \ell, c.\ \neg(\text{grant\_msg}(c, \ell) \wedge \text{free}(\ell))$

$I_7 = \forall\, \ell, c.\ \neg(\text{held\_by}(\ell, c) \wedge \text{free}(\ell))$

$I_8 = \forall\, \ell, c.\ \neg(\text{unlock\_msg}(c, \ell) \wedge \text{free}(\ell))$

[PLDI15] Verdi: a framework for implementing and formally verifying distributed systems. J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. Ernst, T. Anderson

Inductive Invariant for Lock Server

server
clients
network

$\neg$Bad $\equiv$

$I_0 = \forall\, \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \land \text{held\_by}(\ell, c_2) \rightarrow c_1 = c_2$

$I_1 = \forall\, \ell, c_1, c_2.\ \neg(\text{grant\_msg}(c_1, \ell) \land \text{held\_by}(\ell, c_2))$

$I_2 = \forall\, \ell, c_1, c_2.\ \neg(\text{unlock\_msg}(c_1, \ell) \land \text{held\_by}(\ell, c_2))$

$I_3 = \forall\, \ell, c_1, c_2.\ \neg(\text{unlock\_msg}(c_1, \ell) \land \text{grant\_msg}(c_2, \ell))$

$I_4 = \forall\, \ell, c_1, c_2.\ \text{grant\_msg}(c_1, \ell) \land \text{grant\_msg}(c_2, \ell) \rightarrow c_1 = c_2$

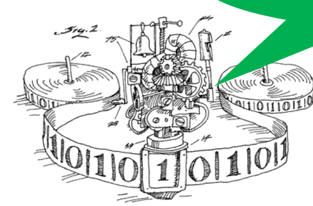
$I_5 = \forall\, \ell, c_1, c_2.\ \text{unlock\_msg}(c_1, \ell) \land \text{unlock\_msg}(c_2, \ell) \rightarrow c_1 = c_2$

$I_6 = \forall\, \ell, c.\ \neg(\text{grant\_msg}(c, \ell) \land \text{free}(\ell))$

$I_7 = \forall\, \ell, c.\ \neg(\text{held\_by}(\ell, c) \land \text{free}(\ell))$

$I_8 = \forall\, \ell, c.\ \neg(\text{unlock\_msg}(c, \ell) \land \text{free}(\ell))$

I *can* decide EPR!



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

EPR Solver

Proof

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)

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

* first mechanized proof

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	85	11	10.7
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)

Proof / code ratio:

IronFleet: ~4

Verdi: ~10

Ivy: ~0.2

Can we further assist the user in finding Inv?

IVy: Safety Verification by Interactive Generalization [PLDI'16]

Oded Padon
Kenneth McMillan
Aurojit Panda
Mooly Sagiv
Sharon Shoham



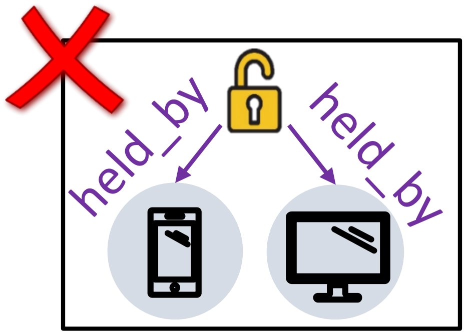

TEL AVIV UNIVERSITY אוניברסיטת תל אביב
Microsoft Research
Berkeley UNIVERSITY OF CALIFORNIA
TEL AVIV UNIVERSITY אוניברסיטת תל אביב

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

$\forall^*$ Inductive Invariant for Lock Server

$\neg$Bad = I_0 = $\forall\ \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \land \text{held\_by}(\ell, c_2) \rightarrow c_1 = c_2$

$I_0 \equiv \neg\exists \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \land \text{held\_by}(\ell, c_2) \land c_1 \neq c_2$



$\forall^*$ Inductive Invariant for Lock Server

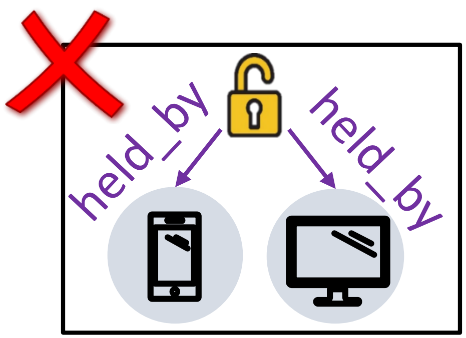
$\neg$Bad = $I_0 = \forall\, \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \wedge \text{held\_by}(\ell, c_2) \rightarrow c_1 = c_2$

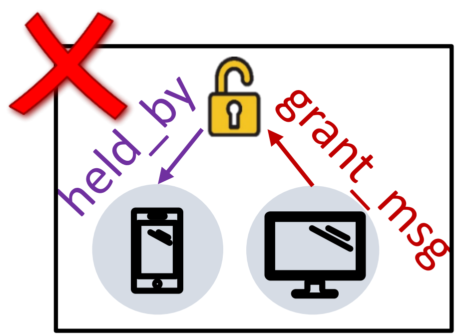
$I_0 \equiv \neg\exists \ell, c_1, c_2.\ \text{held\_by}(\ell, c_1) \wedge \text{held\_by}(\ell, c_2) \wedge c_1 \neq c_2$

$I_1 = \forall\, \ell, c_1, c_2.\ \neg(\text{grant\_msg}(c_1, \ell) \wedge \text{held\_by}(\ell, c_2))$

$I_1 \equiv \neg\exists \ell, c_1, c_2.\ \text{grant\_msg}(c_1, \ell) \wedge \text{held\_by}(\ell, c_2)$

⋮





…

Universally quantified invariant = excluded (partial) states

=> Find invariant by excluding (partial) states

From states to conjectures

Diagram generalizes states

- state σ is a finite first-ord

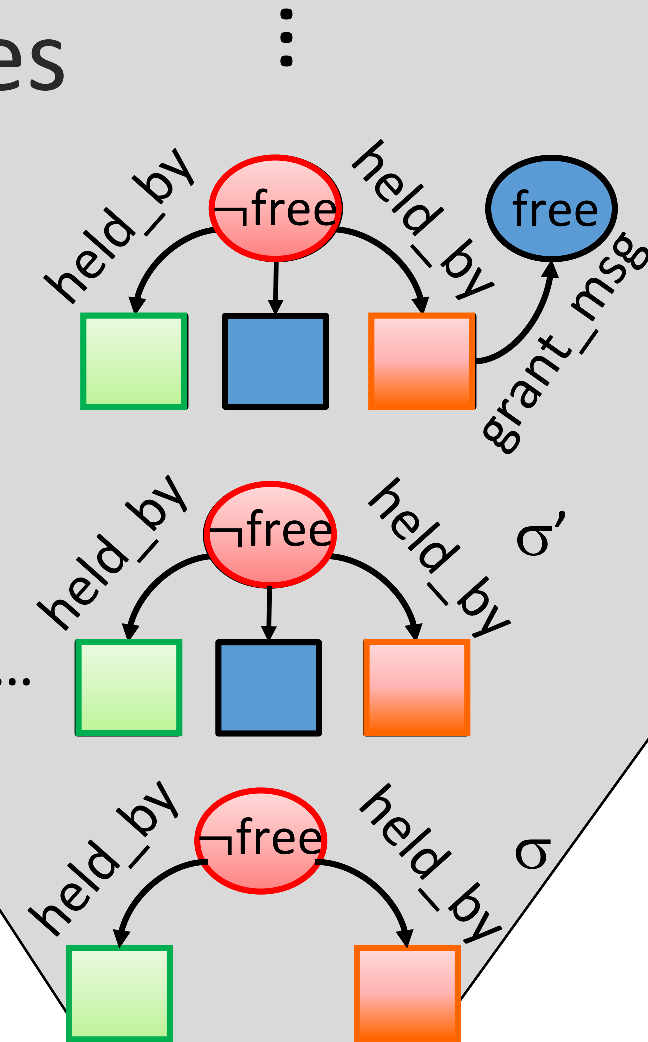
Diag(σ) =

$\exists\, x : L,\ y : C,\ z : C\ .\ y \neq z \wedge \neg free(x)$
$\quad \wedge\ held\_by(x, y) \wedge held\_by(x, z)$
$\quad \wedge\ \neg grnt\_msg(y, x) \wedge \neg grnt\_msg(z, x)\ \ldots$

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)



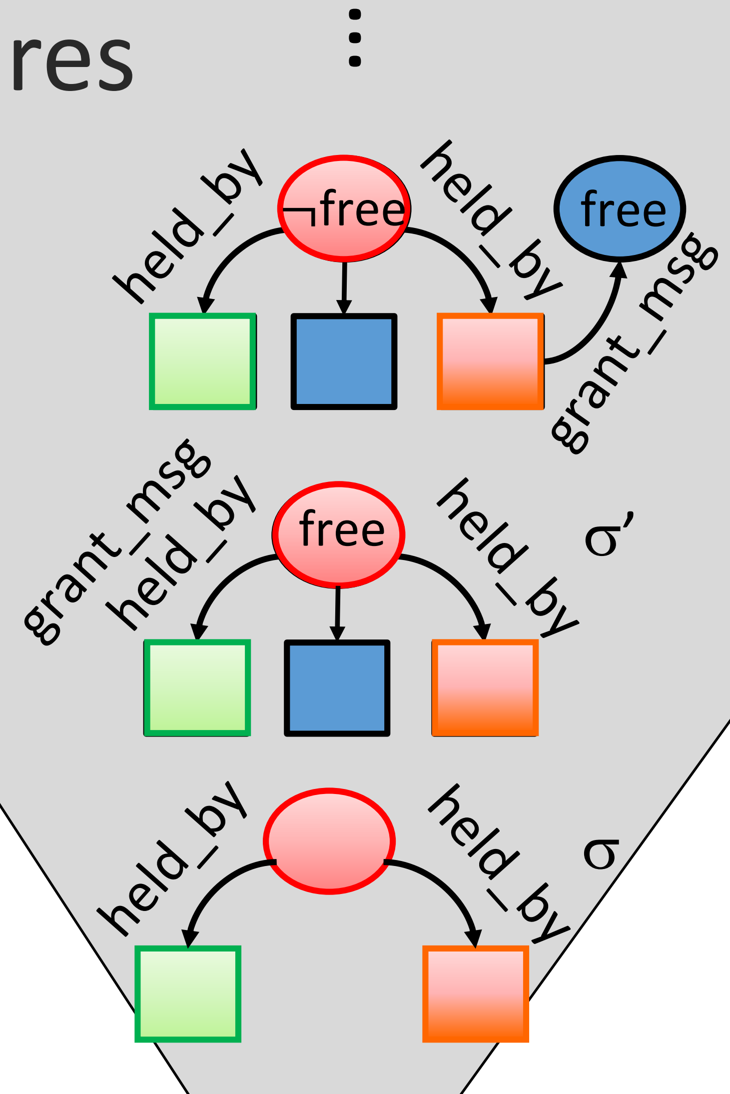
[CAV'15, JACM'17] Property-Directed Inference of Universal Invariants or Pr ir
Absence, A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky and S. Shoham.

From states to conjectures

Generalizes even more
if σ is a partial structure

Diag(σ) =
$\exists$ x : L, y : C, z : C . y ≠ z ∧
∧ held_by(x, y) ∧ held_by(x, z)

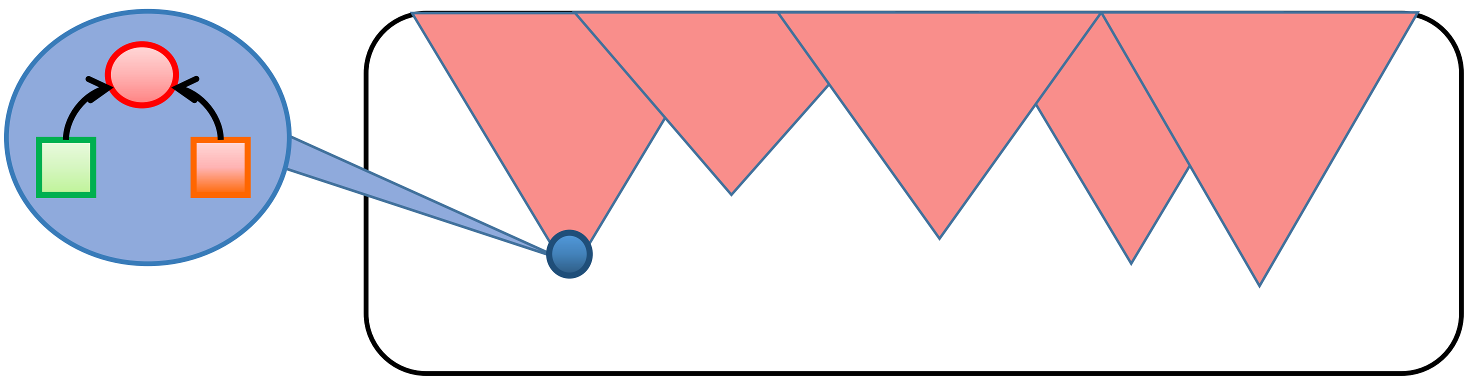
exclude(σ) = ¬Diag(σ)



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

$\forall^*$ Invariant - excluded substructures

$$\text{Inv} \equiv \underbrace{\forall \bar{x}.\ (l_{1,1}(\bar{x}) \vee \ldots \vee l_{1,m}(\bar{x}))}_{\text{clause / conjecture}} \wedge \ldots \wedge \forall \bar{x}.\ (l_{n,1}(\bar{x}) \vee \ldots \vee l_{n,m}(\bar{x}))$$

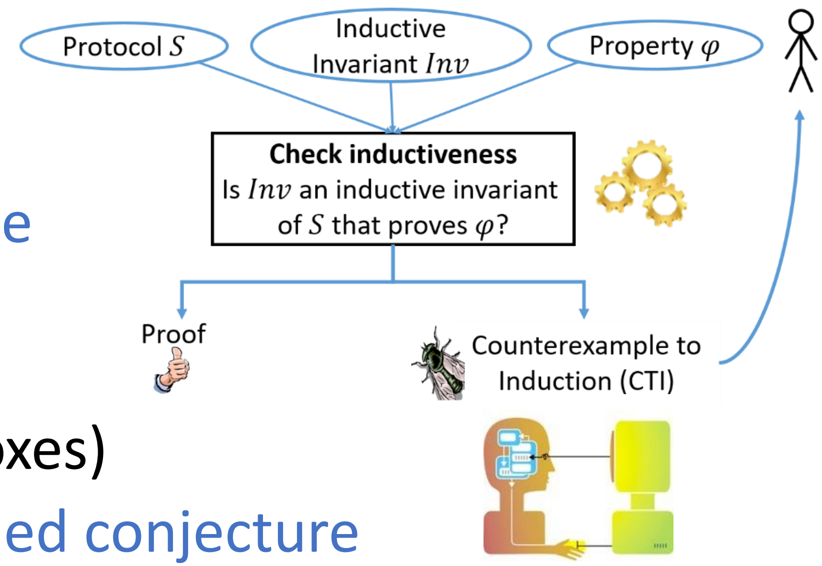


$$\text{Inv} \equiv \neg \underbrace{\exists \bar{x}.\ (\neg l_{1,1}(\bar{x}) \wedge \ldots \wedge \neg l_{1,m}(\bar{x}))}_{\text{cube}} \wedge \ldots \wedge \neg \exists \bar{x}.\ (\neg l_{n,1}(\bar{x}) \wedge \ldots \wedge \neg l_{n,m}(\bar{x}))$$

[PLDI16] Find the partial states to exclude ***interactively***

(2) Fine-Grained Interaction for $\forall^*$ Inv

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



Displays "minimal" CTI to exclude

Generalizes to a partial state

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

Translates to universally quantified conjecture

- uses 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}

Verified protocols [PLDI16]

Protocol	Model (# LOC)	Property (# Literals)	Invariant (# Literals)	Iterations
Leader in Ring	59	3	12	3
Learning Switch	50	11	18	3
DB Chain Replication	143	11	35	7
Chord (partial)	155	35	46	4
Lock Server (500 Coq lines [Verdi])	122	3	21	8
Distributed Lock (1 week [IronFleet])	41	3	26	12

User is involved in discovering each conjecture!

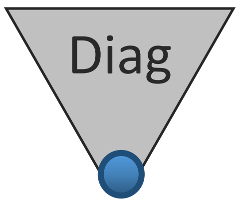
Can we automate this process?

UPDR: Automatic Invariant Inference

- Based on Bradley's IC3/PDR [VMCAI11,FMCAD11]
  - SAT-based verification of finite-state systems
  - Backward traversal to show absence of CEX of bounded length
  - Unreachable states generalized and blocked using lemmas

- UPDR abstracts concrete states using their diagram

=> Infers $\forall^*$ inductive invariants



- [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.

But…

- Automatic invariant inference is limited
  - Infinite search space
  - Undecidable to infer $\forall^*$ invariants [POPL'16]

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

How can the user convey their intuition to the inference procedure?

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

Inferring Phase Invariants from Phase Sketches

Yotam Feldman

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

James Wilcox

UNIVERSITY of WASHINGTON

Sharon Shoham

Mooly Sagiv

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

Phase Invariants

- Idea: add structure to the inductive invariant
- User provides the structure as “hints” to automatic inference

Reminder: Ind. Inv. for Lock Server

I_0 = $\forall\ \ell, c_1, c_2.$ held_by(ℓ, c_1) ∧ held_by(ℓ, c_2) → $c_1 = c_2$

I_1 = $\forall\ \ell, c_1, c_2.$ ¬(grant_msg(c_1,ℓ) ∧ held_by(ℓ, c_2))

I_2 = $\forall\ \ell, c_1, c_2.$ ¬(unlock_msg(c_1,ℓ) ∧ held_by(ℓ, c_2))

I_3 = $\forall\ \ell, c_1, c_2.$ ¬(unlock_msg(c_1,ℓ) ∧ grant_msg(c_2,ℓ))

I_4 = $\forall\ \ell, c_1, c_2.$ grant_msg(c_1,ℓ) ∧ grant_msg(c_2,ℓ) → $c_1 = c_2$

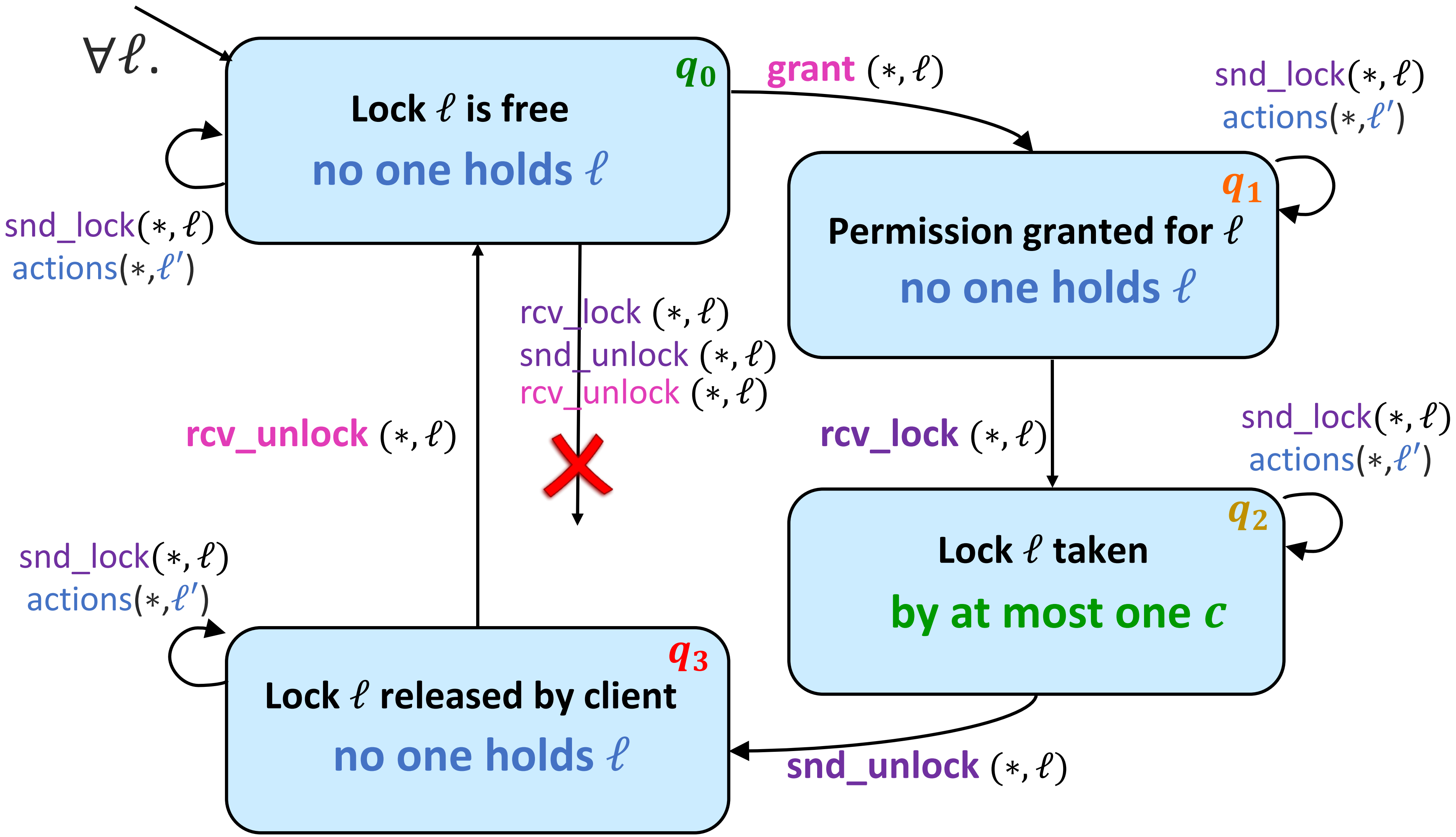
I_5 = $\forall\ \ell, c_1, c_2.$ unlock_msg(c_1,ℓ) ∧ unlock_msg(c_2,ℓ) → $c_1 = c_2$

I_6 = $\forall\ \ell, c.$ ¬(grant_msg(c,ℓ) ∧ free(ℓ))

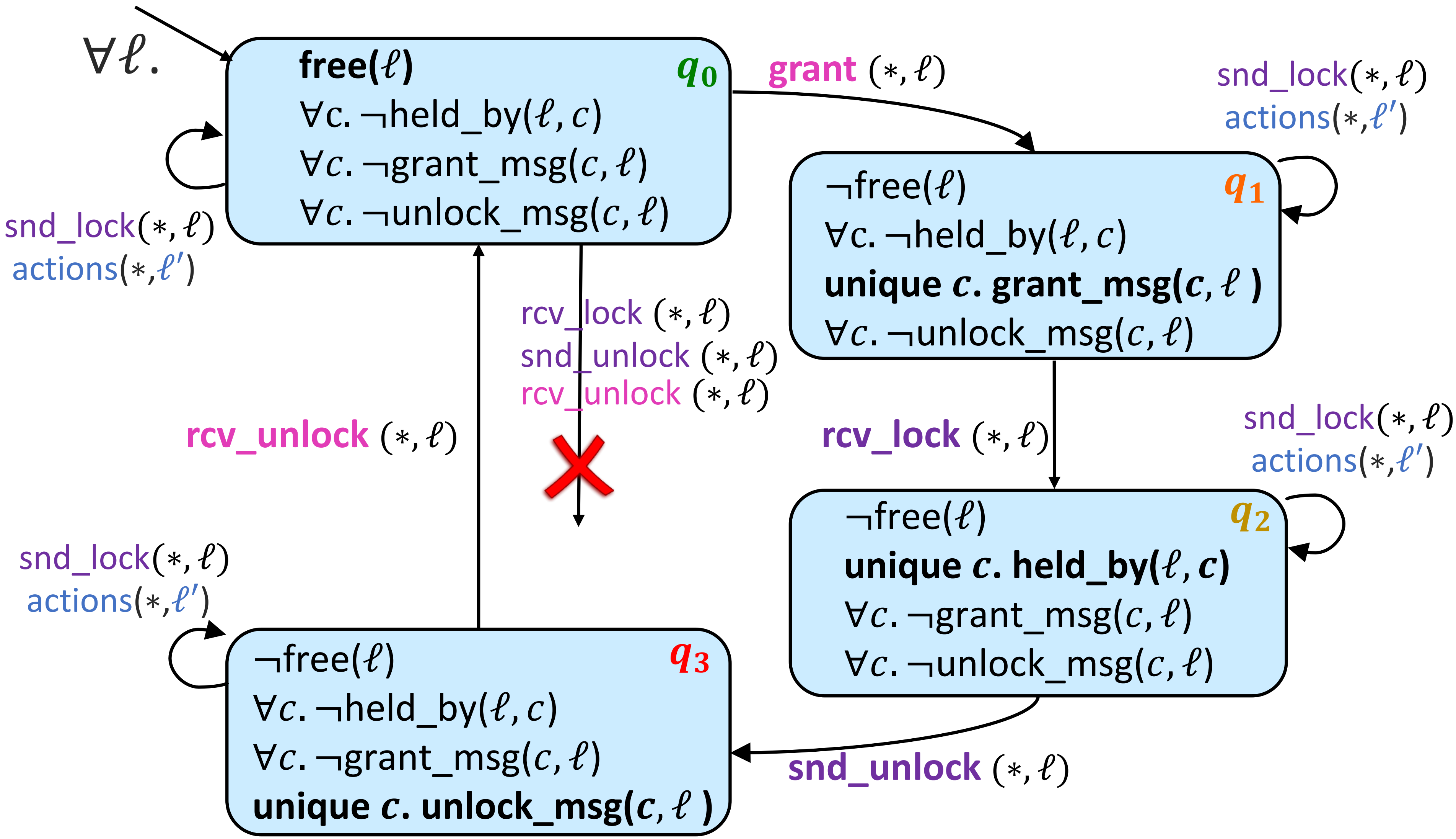
I_7 = $\forall\ \ell, c.$ ¬(held_by(ℓ,c) ∧ free(ℓ))

I_8 = $\forall\ \ell, c.$ ¬(unlock_msg(c,ℓ) ∧ free(ℓ))

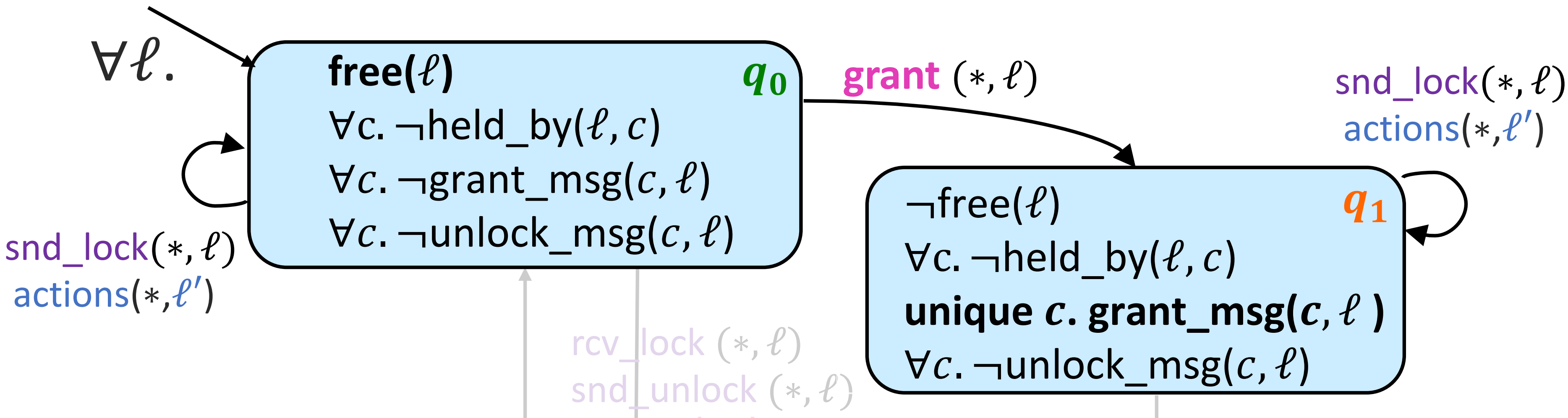
Phase Structure of Lock Server's Proof



Inductive Phase Invariant for Lock Server



Inductive Phase Invariant for Lock Server



Instead of monolithic consecution

Initiation: $Init \Longrightarrow \varphi_{q_0}$

Inductive: $\varphi_{q_0} \wedge TR_{\text{grant}(*,\ell)} \Longrightarrow {\varphi'}_{q_1}$

$\varphi_{q_0} \wedge (TR_{\text{request}(*,\ell)} \vee TR_{\text{actions}(*,\ell')}) \Longrightarrow {\varphi'}_{q_0}$

Covers: $\varphi_{q_0} \wedge TR \Longrightarrow TR_{\text{grant}(*,\ell)} \vee TR_{\text{request}(*,\ell)} \vee TR_{\text{actions}(*,\ell')}$

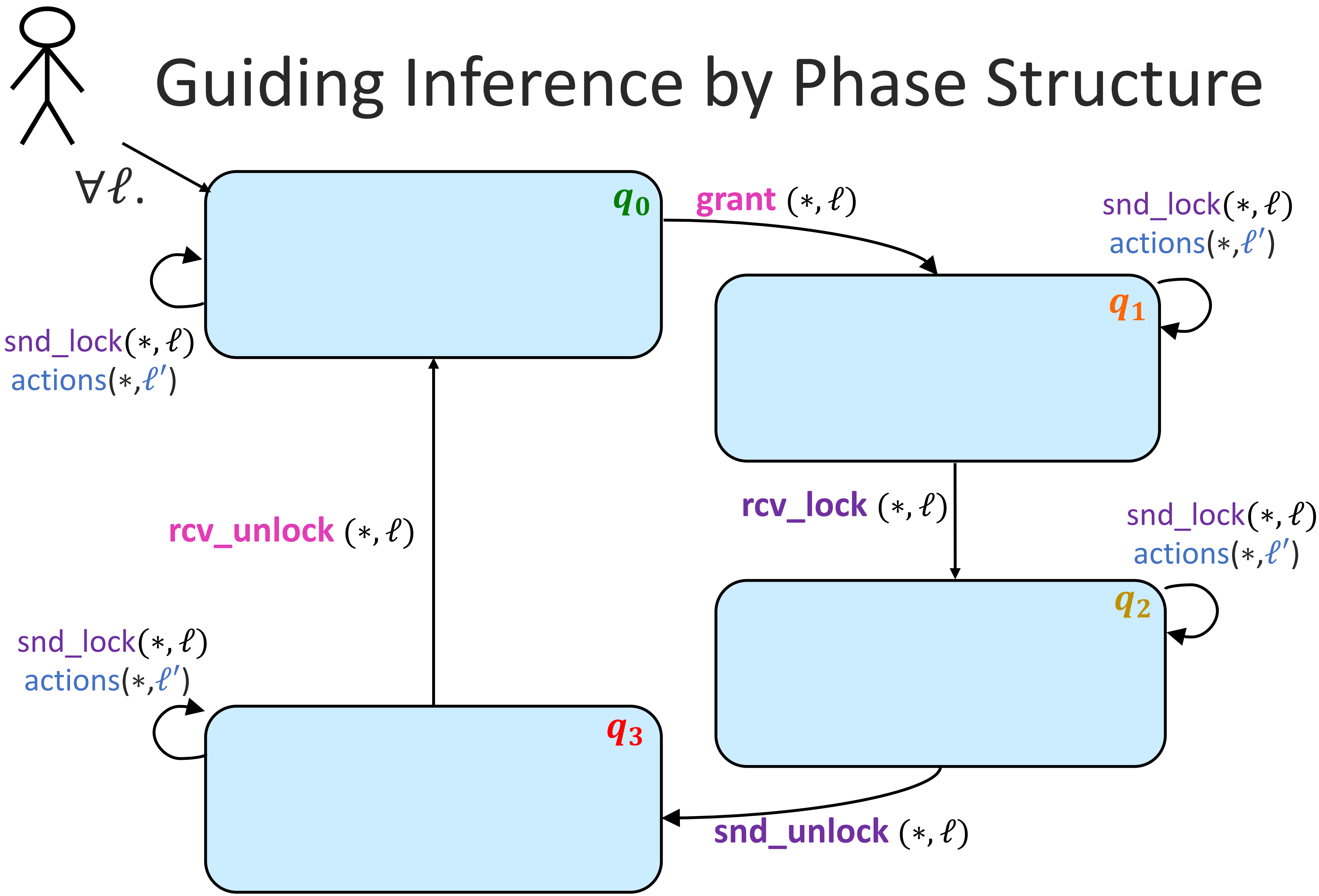
Safe: $\varphi_{q_0} \Longrightarrow \forall\, c_1, c_2.\ \text{held\_by}(\ell, c_1) \wedge \text{held\_by}(\ell, c_2) \rightarrow c_1 = c_2$

Guiding Inference by Phase Structure

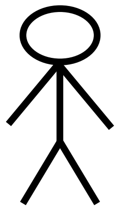
1. **User** provides the **phase structure** as the proof's **essence**

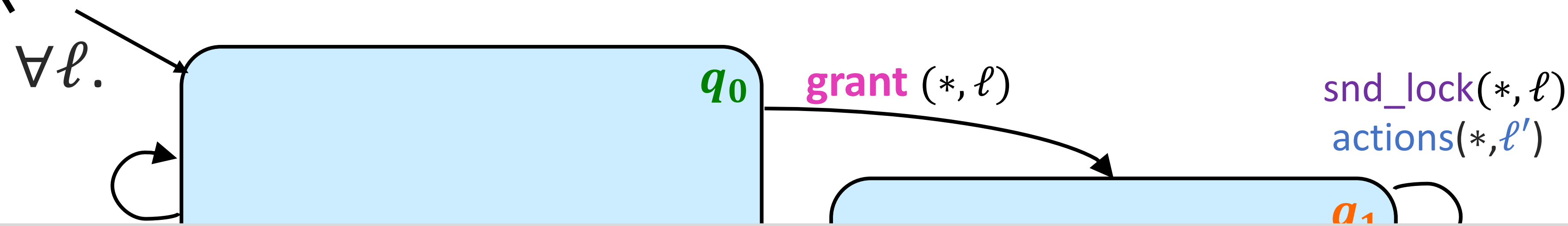
2. **Automatically** infer **phase characterizations** for a **full formal proof**

Guiding Inference by Phase Structure



Guiding Inference by Phase Structure





Infer **phase characterizations** $\varphi_{q_0}, \varphi_{q_1}, \varphi_{q_2}, \varphi_{q_3}$ s.t.

Initiation $\quad Init \implies \varphi_{q_0}$

Instead of monolithic consecution:

- Inductive $\quad \varphi_q \wedge TR_{(q,p)} \implies {\varphi'}_p$
- Cover $\quad \varphi_q \wedge TR \implies \bigvee_{(q,p) \in E} TR_{(q,p)}$

Safe $\quad \varphi_q \implies \text{Safety}$

Additional safety constraints

Phase-UPDR: Inference of $\forall^*$ characterization

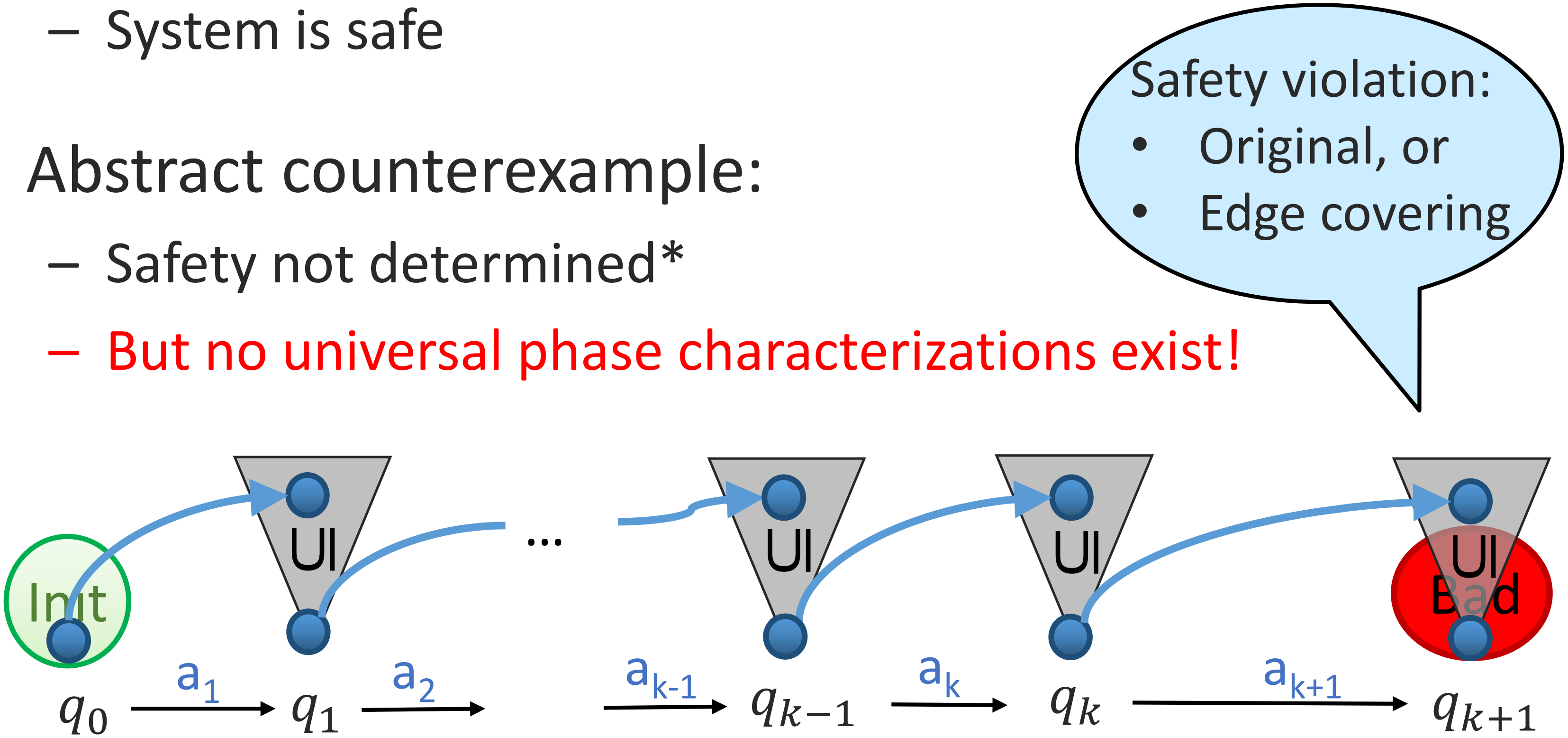
* System of *linear* second-order Constrained Horn Clauses (CHCs)

Phase-UPDR: Possible outcomes

- Universal phase characterizations found
  - System is safe

Phase-UPDR: Possible outcomes

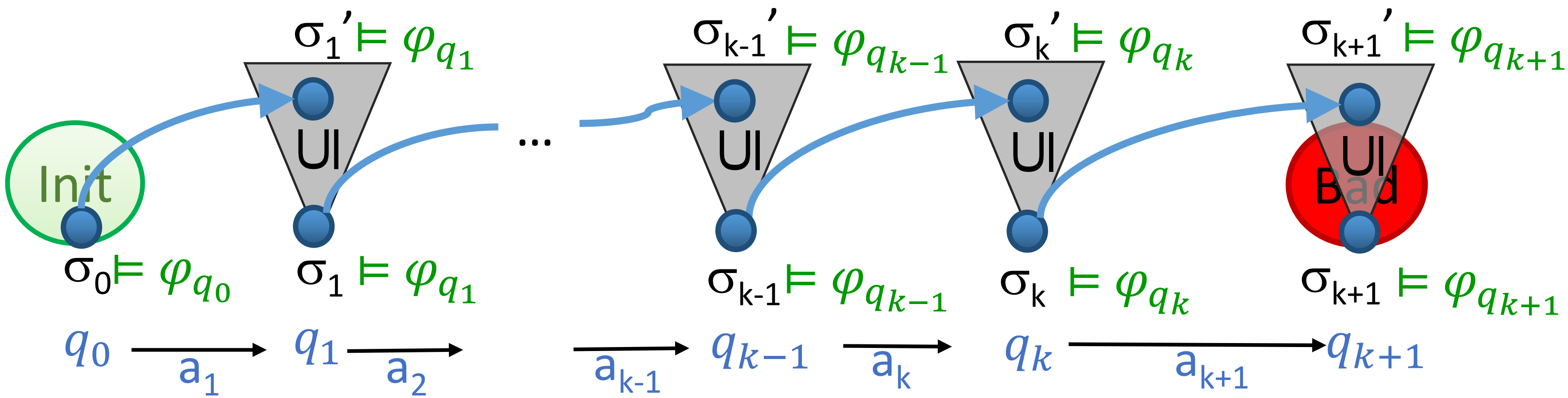
- Universal phase characterizations found
  - System is safe
- Abstract counterexample:
  - Safety not determined*
  - But no universal phase characterizations exist!



* can use Bounded Model Checking to find real counterexamples

Proving absence of universal phase characterizations

Suppose that universally quantified characterizations φ_{q_i} exist. Then:



φ_{q_0} satisfies initiation: $\sigma_0 \models \text{Init} \Rightarrow \sigma_0 \models \varphi_{s_0}$

$\varphi_{q_{i-1}}$ is inductive: $\sigma_{i-1} \models \varphi_{q_{i-1}} \wedge \mathrm{TR}_{a_{i-1}}(\sigma_{i-1}, \sigma_i') \Rightarrow \sigma_i' \models \varphi_{q_i}$

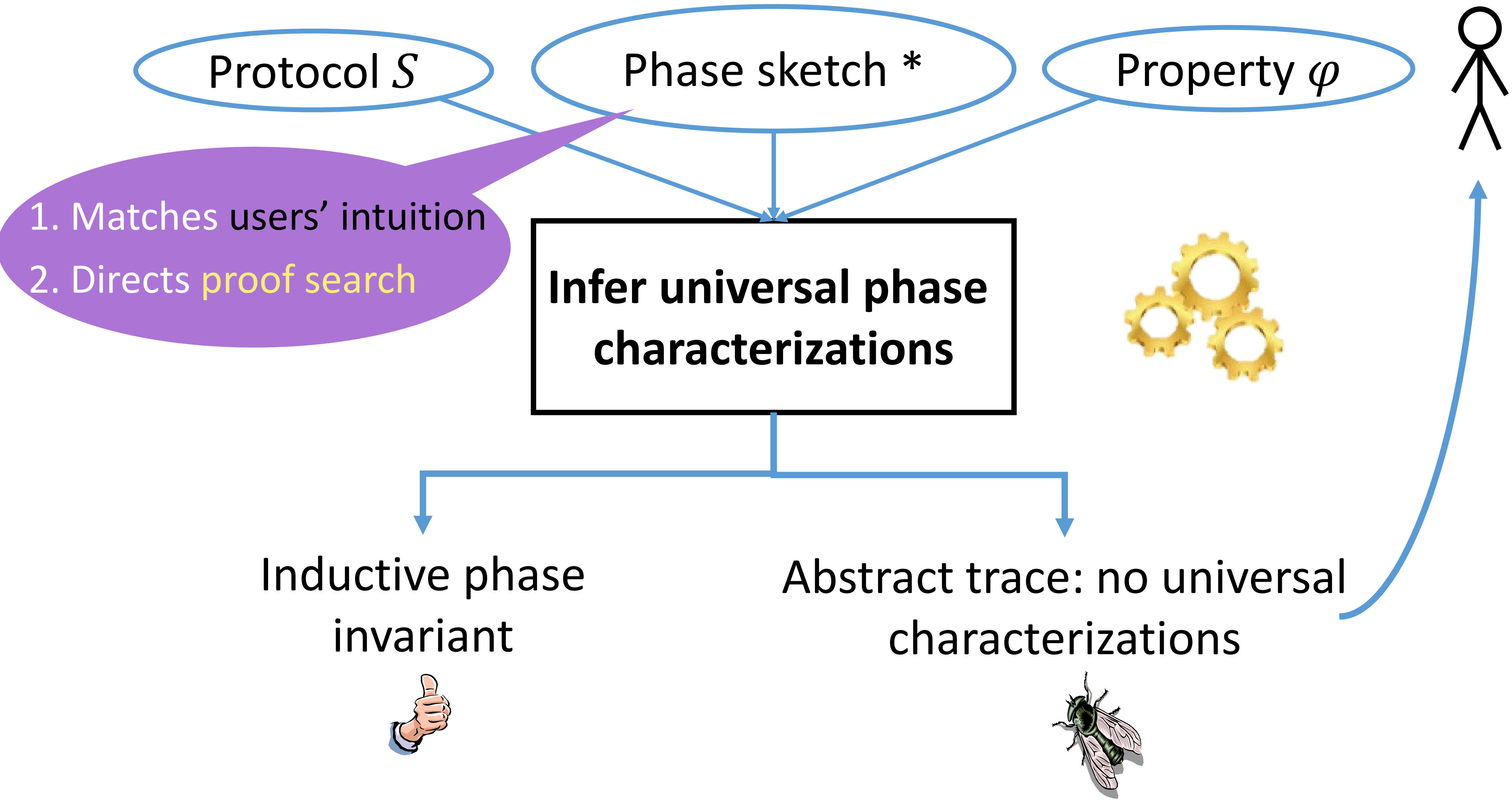
φ_{q_i} is universal: $\sigma_i' \models \text{Diag}(\sigma_i) \Rightarrow \sigma_i \models \varphi_{q_i}$

Contradicts safety!

If there exist $\varphi_{q_i} \in \forall^*$, then any **abstract trace** does not reach Bad

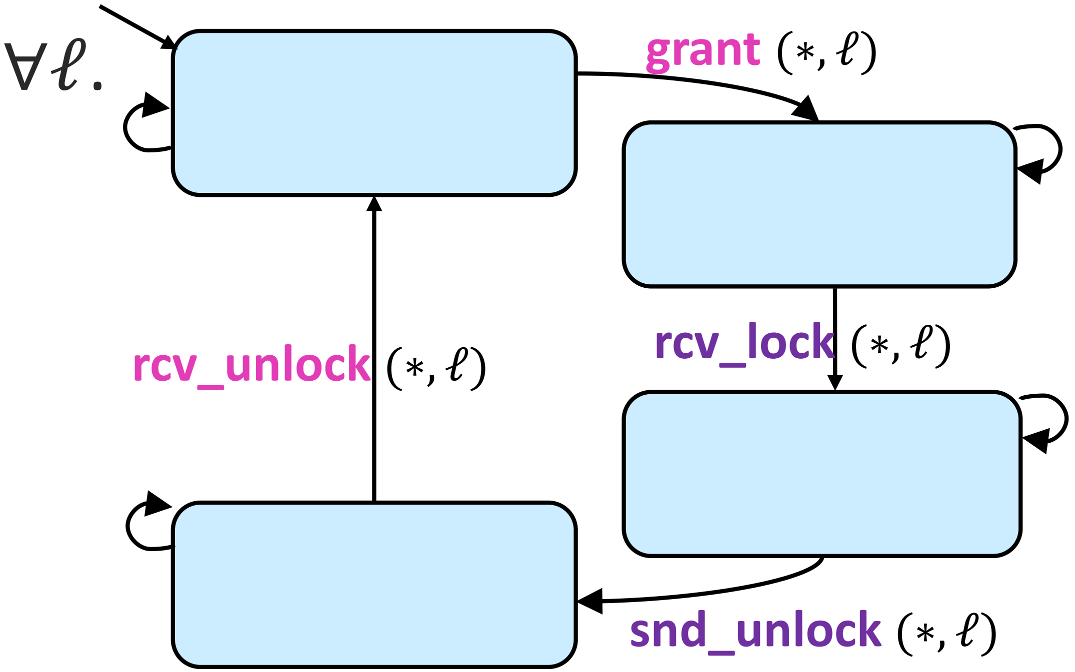
➔ An abstract trace to Bad implies no universal phase characterizations

(3) Interaction based on phase sketches



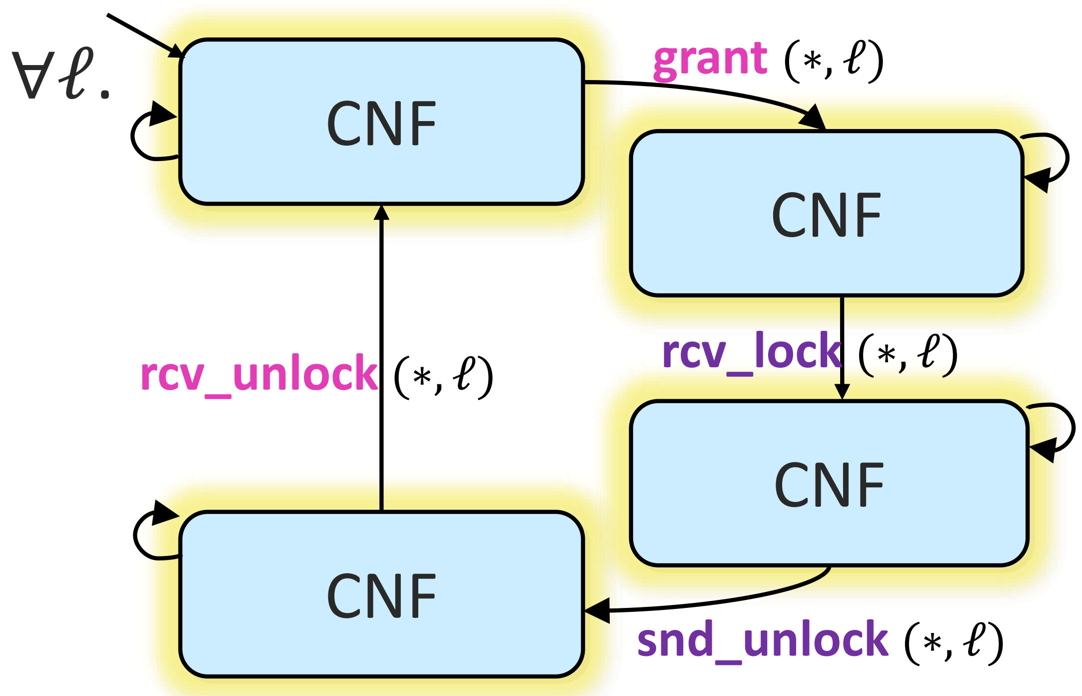
* Phase structure, possibly with partial phase characterizations

Benefits of Phases for UPDR



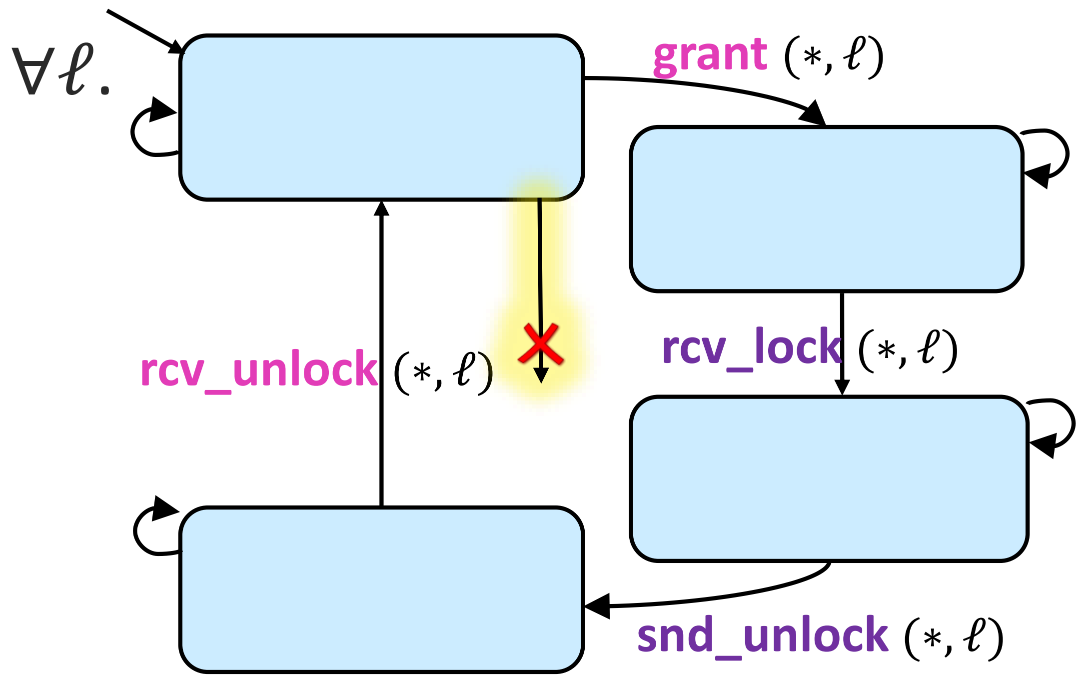
Benefits of Phases for UPDR

- **Disjunctive structure**



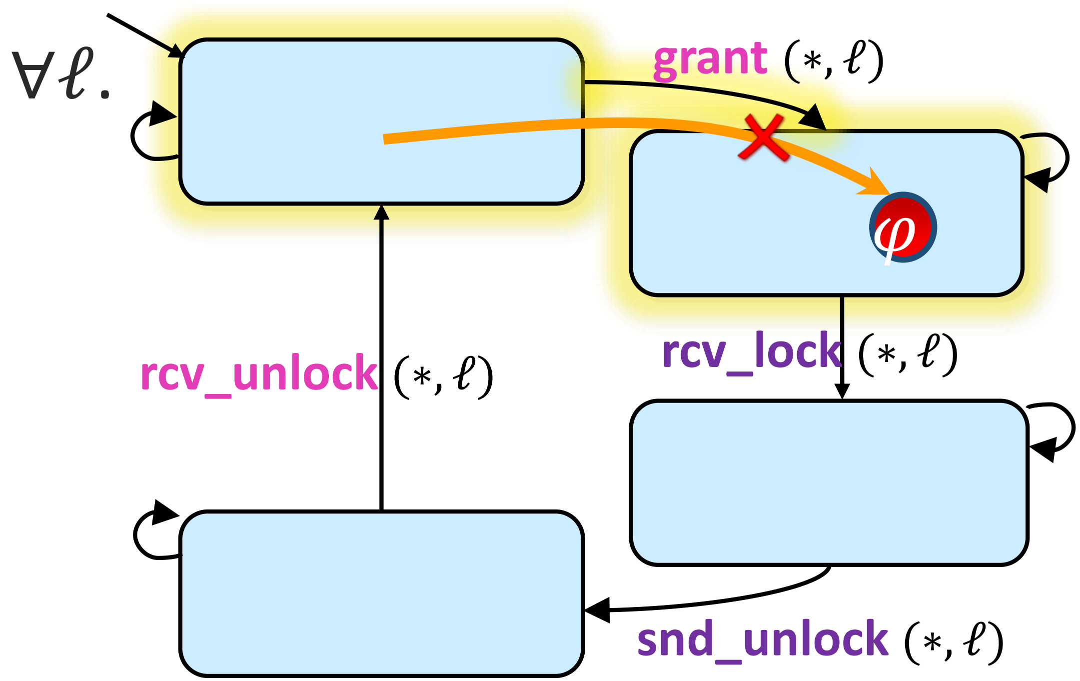
Benefits of Phases for UPDR

- Disjunctive structure
- **Impossible transitions**



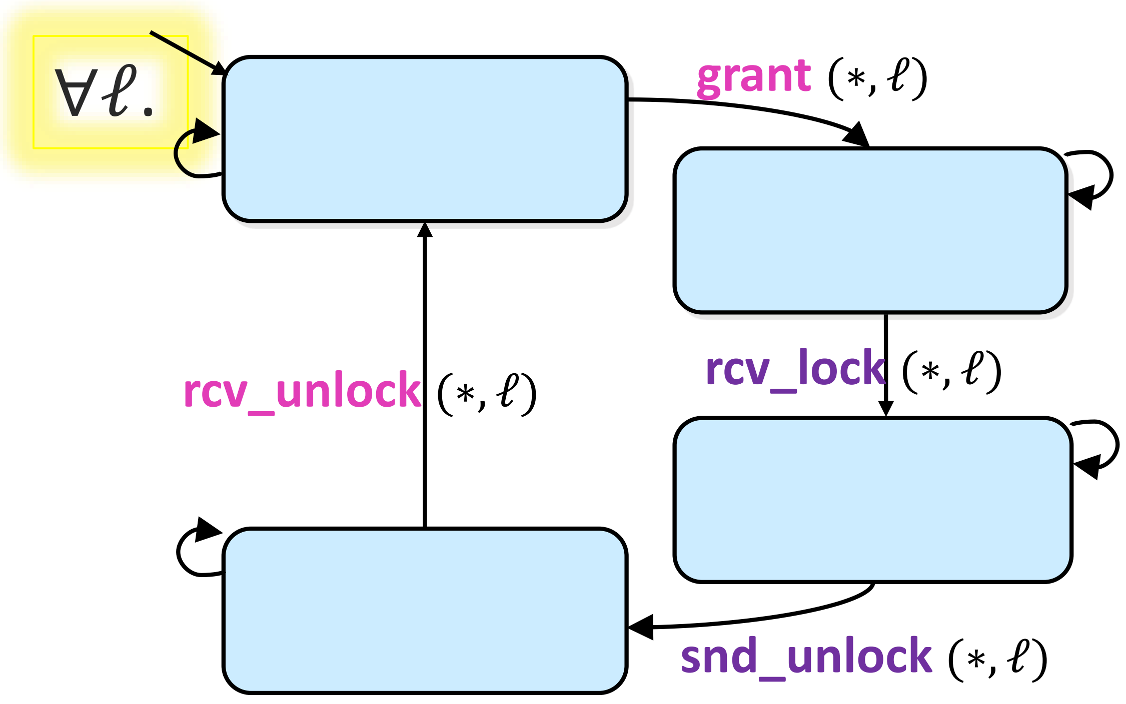
Benefits of Phases for UPDR

- Disjunctive structure
- Impossible transitions
- **Generalization w.r.t. subsystem**



Benefits of Phases for UPDR

- Disjunctive structure
- Impossible transitions
- Generalization w.r.t. subsystem
- **Arity reduction?**

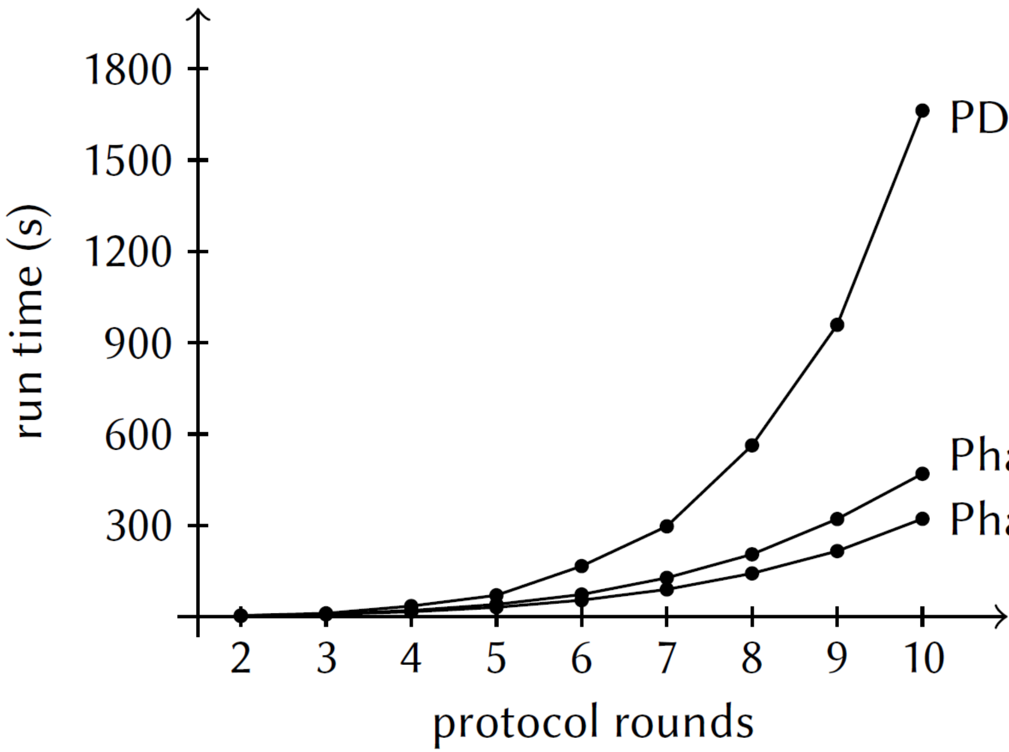


Evaluation

Protocol	Phase Sketch * [min]	Phase Structure [min]	Inductive Invariant [min]
Lock server (single lock)	00:05	**00:04**	00:21
Lock server (multiple locks)	**00:10**	00:11	00:22
Ring leader election	00:12	**00:03**	02:04
Simple consensus	03:04	02:07	**01:27**
Sharded KV (basic, one key)	**00:02**	00:03	00:08
Sharded KV (basic, multiple keys)	**00:05**	00:08	00:06
Sharded KV (w/ retransmissions)	**03:01**	38:17	> 3 hours

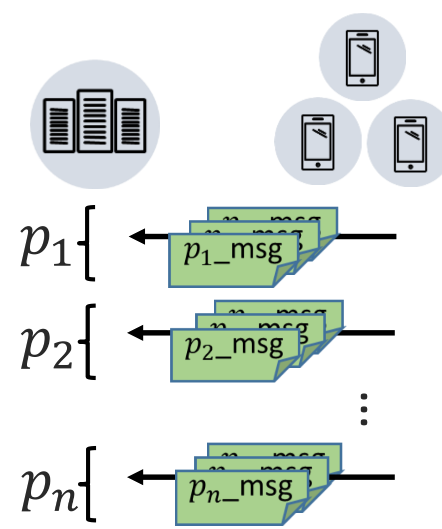
* With partial phase characterizations

Structure and Scaling

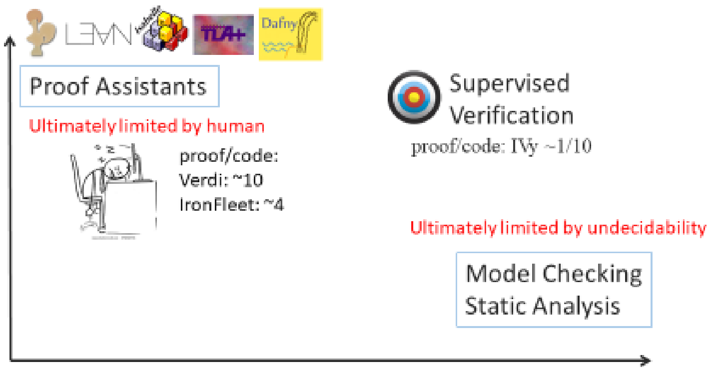


n-phase commit:

- start p_{i+1} when $\forall c.\, p_i\_\mathrm{msg}(c)$
- done when $\forall c.\, p_n\_\mathrm{msg}(c)$
- **Safety**: done $\rightarrow \forall c.\, p_1\_\mathrm{msg}(c)$



Summary



Interactive verification using decidable logic

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

- Interaction based on CTIs
- Fine-grained interaction based on diagrams
- Coarse-grained interaction based on phase sketches & relaxed traces

Find ways to guide verification tools!

- Dividing the problem between human and machine
- Other logics
- Inference schemes
- Forms of interaction
- Theoretical understanding of limitations and tradeoffs

Seeking postdocs and students

erc Supervised Verification of Infinite-State Systems