

Deductive Verification in First Order Logic

Mooly Sagiv



Contributors

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



James R. Wilcox,

Doug Woos



And Also

Anindya Benerjee



Yotam Feldman



Neil Immerman



Aurojit Panda



Shachar Itzhaky



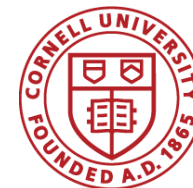
Aleks Nanevsky



Orr Tamir



Robbert van Renesse



The Ivy system

- Motivated by distributed systems
- Proof automation
 - Automatic checking of inductiveness
- Transparency
 - visible failures
 - The user never get stuck
- Modularity
 - The user can divide the verification problem into decidable parts



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

The mypyvy system

James R. Wilcox



- A low level engine for first order reasoning
- Proof automation
 - Automatic checking of inductiveness
 - Automatic invariant inference
- Transparency
 - visible failures
 - The user never get stuck



<https://github.com/wilcoxjay/mypyvy/>

Mypyvy instructions (james@certora.com)

VM: <https://tinyurl.com/mypyvy-vm>

Username: mypyvy

Password: mypyvy

Sources: <https://github.com/wilcoxjay/mypyvy>

verify - Check that a safety property holds using a given inductive invariant

updr - Infer a sufficiently strong invariant which implies a safety property

Deductive Verification in First Order Logic

[CAV'13] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, MS: [Effectively-Propositional Reasoning about Reachability in Linked Data Structures](#)

[JACM] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, Sharon Shoham: Property-Directed Inference of Universal Invariants or Proving Their Absence. J. ACM 64(1): 7:1-7:33 (2017)

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

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

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

[POPL'18] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, MS, Sharon Shoham: Reducing liveness to safety in first-order logic. PACMPL 2(POPL): 26:1-26:33 (2018)

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

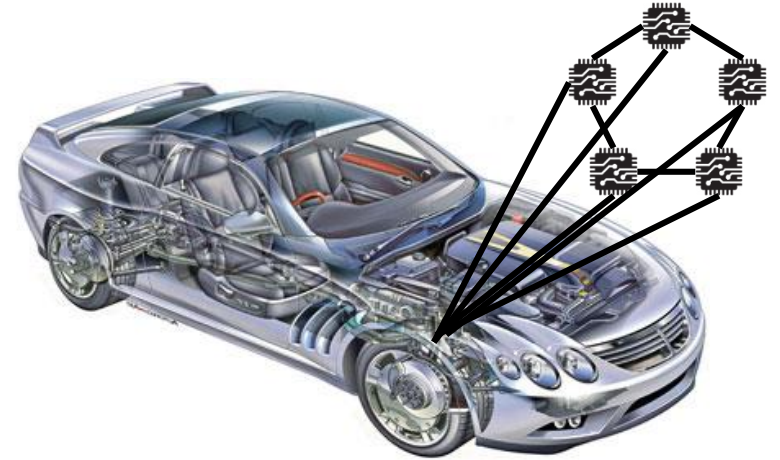
[CAV'19] Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, MS: Inferring Inductive Invariants from Phase Structures. CAV (2) 2019: 405-425

Agenda

- Wednesday
 - Motivation
 - **Inductive invariants**
 - A bird eye of deductive Verification in Ivy
- Thursday
 - **Decidable logics Effectively Propositional Logic EPR**
 - Case study
 - Reasoning about linked list
 - Modularity and decidability
- Friday
 - Mypyvy James Wilcox
- Saturday
 - Certora Verifying Smart Contracts

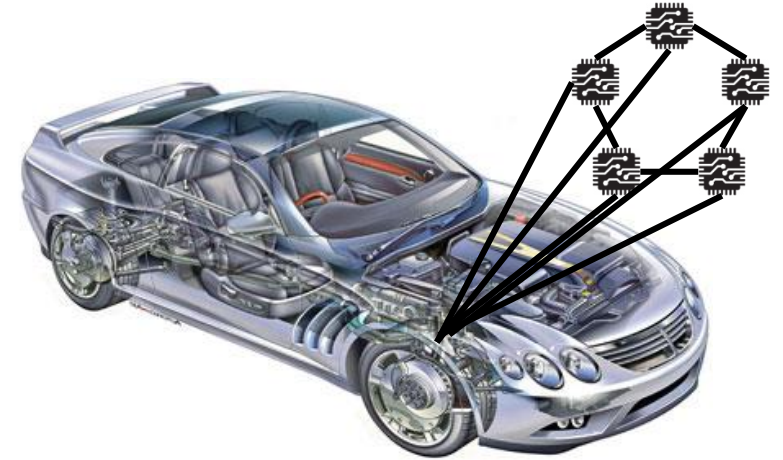
Why verify distributed protocols?

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



Why verify distributed protocols?

- Distributed systems are everywhere
 - Safety-critical systems
 - Cloud infrastructure
 - Blockchain
- Distributed systems are notoriously hard to get right



SIGCOMM'01

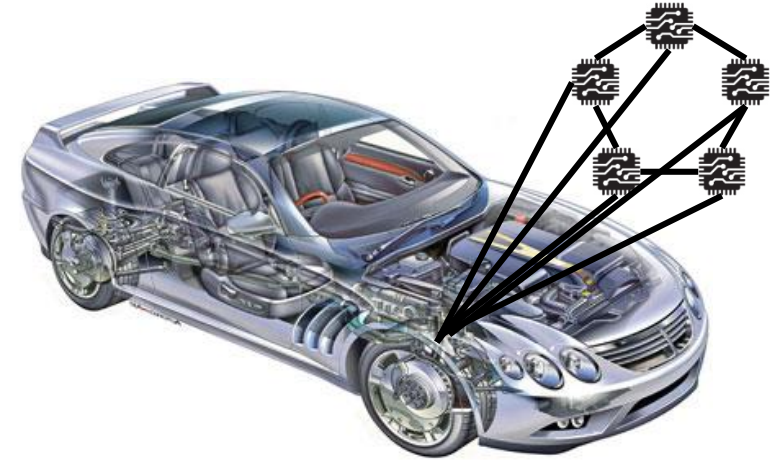
Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications

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

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

Why verify distributed protocols?

- Distributed systems are everywhere
 - Safety-critical systems
 - Cloud infrastructure
 - Blockchain
- Distributed systems are not



SIGCOMM'01

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

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M
Hari Balakrishnan, Member, IEEE

Attractive features of Chord include
correctness, and provable performance
concurrent node arrivals and departures. It can

Using Lightweight Modeling To Understand Chord

CCP'12
Pamela Zave
AT&T Laboratories—Research
Florham Park, New Jersey USA
pamela@research.att.com

Under the same assumptions made in the Chord papers,
the [SIGCOMM] version of the protocol is not correct, and
not one of the properties claimed invariant in [PODC] is
actually invariantly true of it. The [PODC] version satis-
fies one invariant, but is still not correct. The results are
presented by means of counterexamples to the invariants in
Section 4. In preparation for the results, Section 2 gives a



SOSP'07

Best Paper Award

Zyzyva: Speculative Byzantine Fault Tolerance

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

Zyzyva is a state machine replication protocol based on
protocols: (1) agreement, (2) view change, and (3)
agreement protocol orders requests for exe-
cution. The view change protocol coordinates

CACM'08

Zyzyva: Speculative Byzantine Fault Tolerance

ACM Transactions on Computer Systems '09

Zyzyva: Speculative Byzantine Fault Tolerance

RAMAKRISHNA KOTLA
Microsoft Research, Silicon Valley
and

LORENZO ALVISI, MIKE DAHLIN, ALLEN CLEMENT, and EDMUND WONG
The University of Texas at Austin

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

Revisiting Fast Practical Byzantine Fault Tolerance

Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

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

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

Proving distributed systems is hard

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

Our goal: reduce human effort while maintaining flexibility

Our approach: decompose verification into decidable problems

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

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

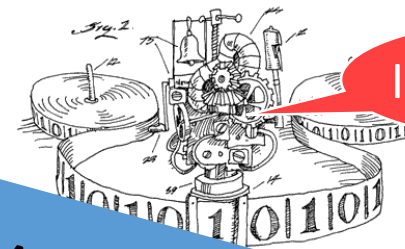
[PLDI'15] Wilcox et al. Verdi: a framework for implementing and formally verifying distributed systems

Automatic verification of infinite-state systems



Verification
behavior

Rice's Theorem



I can't decide!

*“Formal methods are the future of computer science. Always have been, **always will** be.” William E. Aitken*

Counterexample



Unknown / Diverge



Proof



Challenges in program verification

- Specifying program behavior
- Asymptotic complexity of program verification
 - The halting problem
 - Rice theorem
 - The ability of simple programs to represent complex behaviors
- The complexity of realistic systems
 - Huge code
 - Heterogeneous code
 - Missing code

Successful verification for systems



- [OSDI'16] H. Sigurbjarnarson, J. Bornholt, E. Torlak, Xi Wang: Push-Button Verification of File Systems via Crash Refinement **Best paper award**
- [SOSP'15a] H. Chen, D. Ziegler, T. Chajed, A. Chlipala, F. Kaashoek, N. Zeldovich: Using Crash Hoare logic for certifying the FSCQ file system. **Best paper award**
- [SOSP'15b] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S.T. V. Setty, B. Zill: IronFleet: proving practical distributed systems correct
- [PLDI'15a] J. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X.Wang, M. Ernst, T. Anderson: Verdi: a framework for implementing and formally verifying distributed systems
- [PLDI'15b] Nuno P. Lopes, David Menendez, Santosh Nagarakatte, John Regehr: Provably correct peephole optimizations with alive **Best paper award**
- [POPL'04] T. A. Henzinger, R. Jhala, R. Majumdar, K. L. McMillan: Abstractions from proofs **Test of time award**
- [PLDI'01] T. Ball, R. Majumdar, T.D. Millstein, S.K. Rajamani: Automatic Predicate Abstraction of C Programs **Test of time award**

Semi-automatic deductive verification



Deductive verification



Deductive Verification
Does there exist a proof for S that proves φ ?
→ Are the conditions valid?

Counter-model

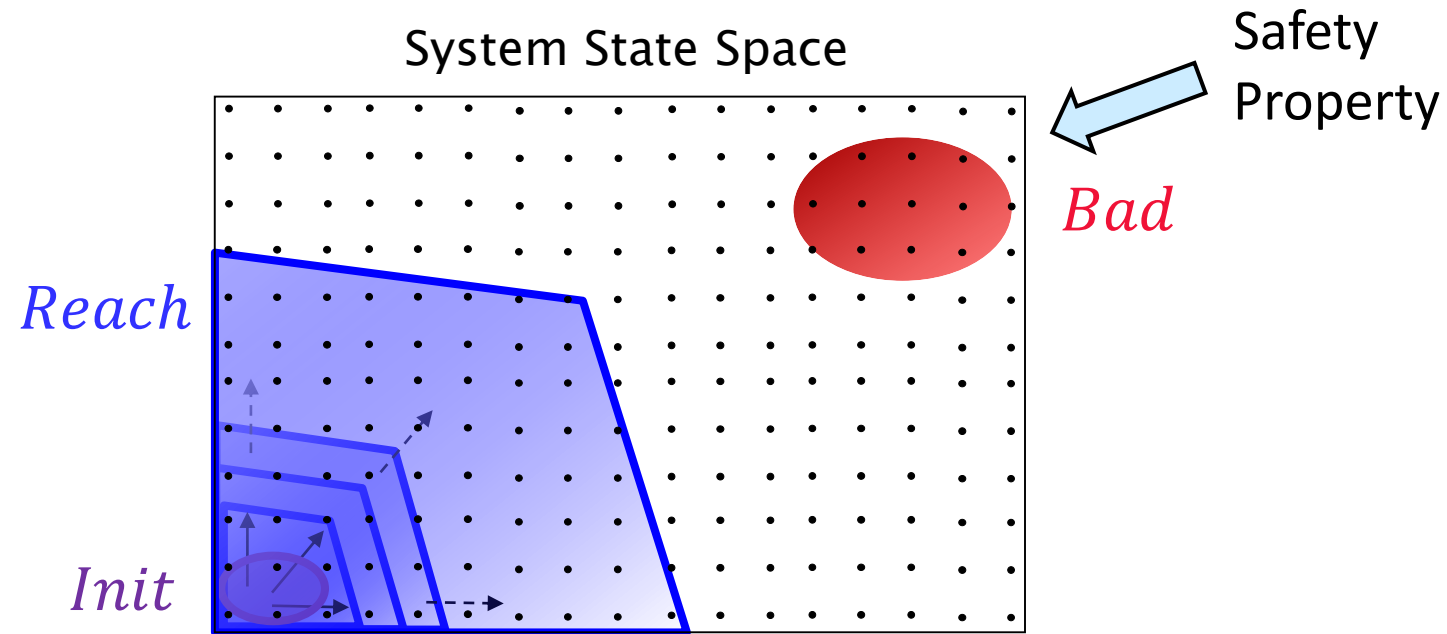


Proof



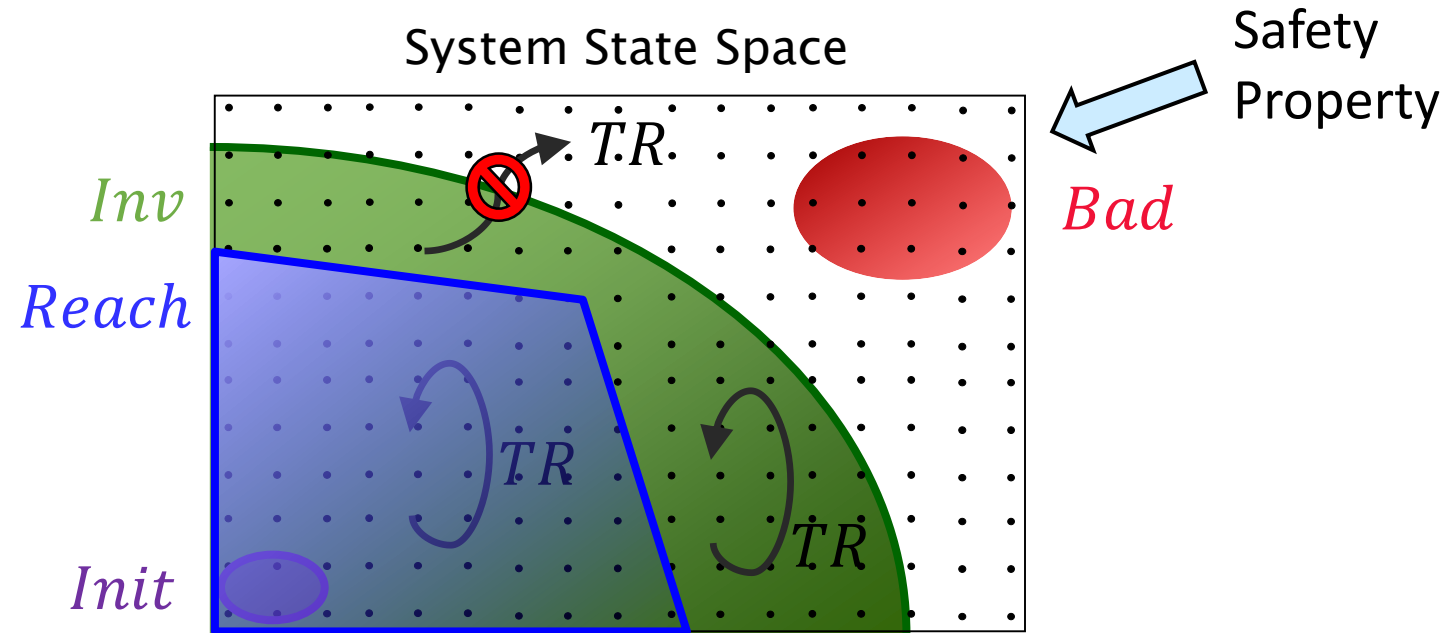
“Deduction is forever” Amir Pnueli

Inductive invariants



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

Inductive invariants



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

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

$\mathit{Init} \subseteq \mathit{Inv}$ (**Initiation**)

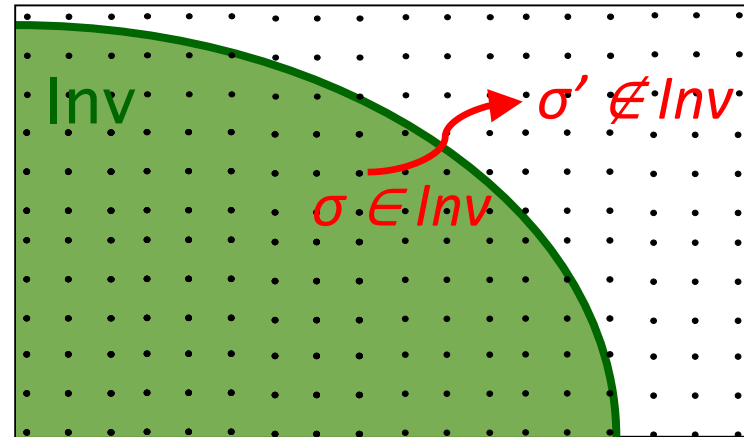
if $\sigma \in \mathit{Inv}$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in \mathit{Inv}$ (**Consecution**)

$\mathit{Inv} \cap \mathit{Bad} = \emptyset$ (**Safety**)

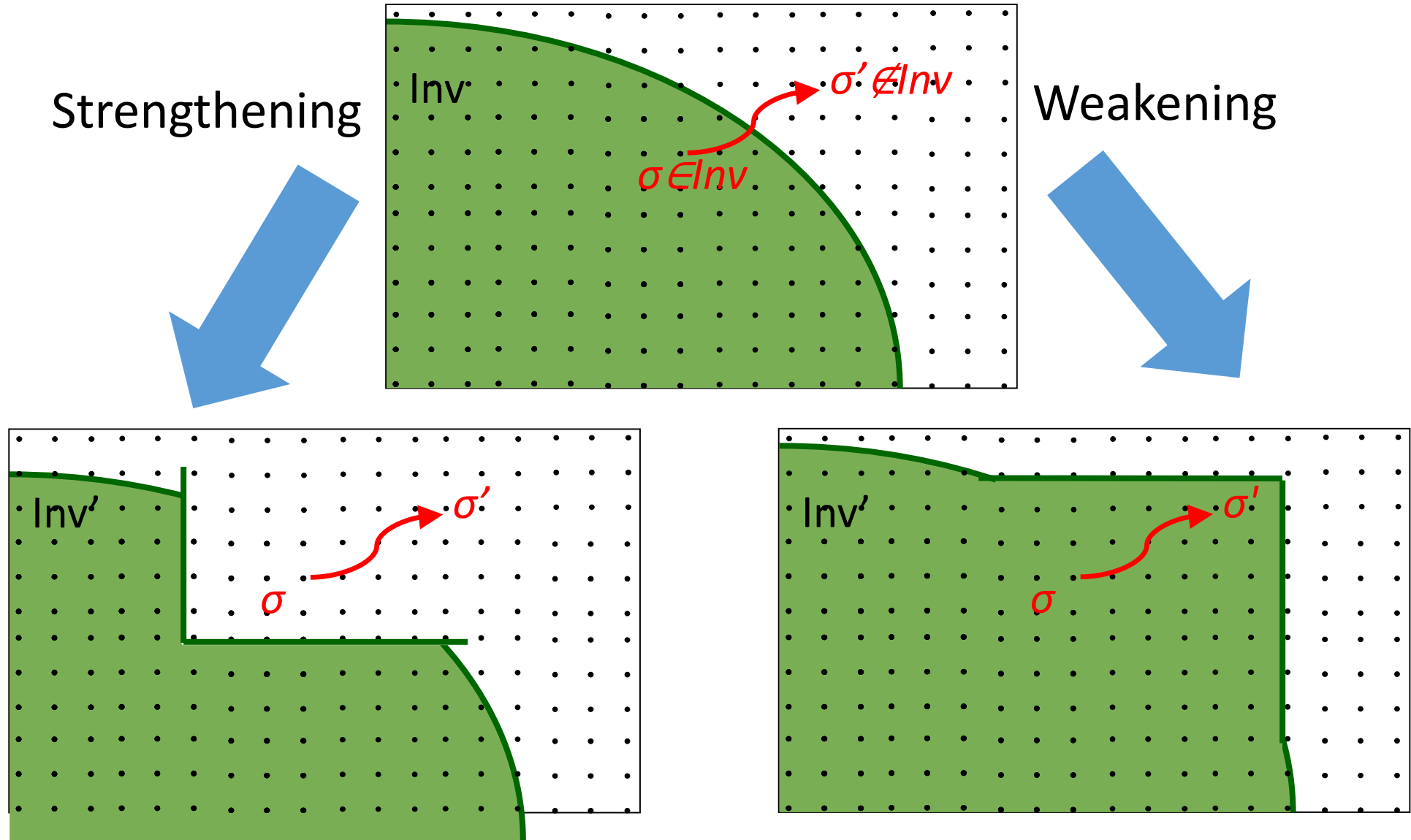
} translated to VC's

Counterexample To Induction (CTI)

- States σ, σ' are a CTI of Inv if:
 - $\sigma \in Inv$
 - $\sigma' \notin Inv$
 - $\sigma \rightarrow \sigma'$
- A CTI may indicate:
 - A bug in the system
 - A bug in the safety property
 - A bug in the inductive invariant
 - Too weak
 - Too strong

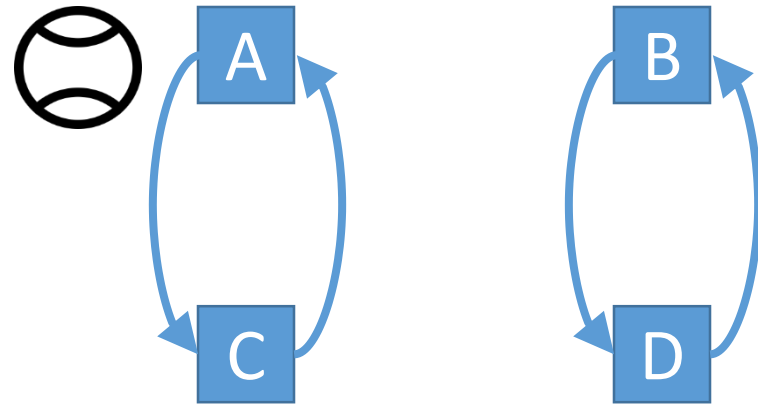


Strengthening & weakening from CTI



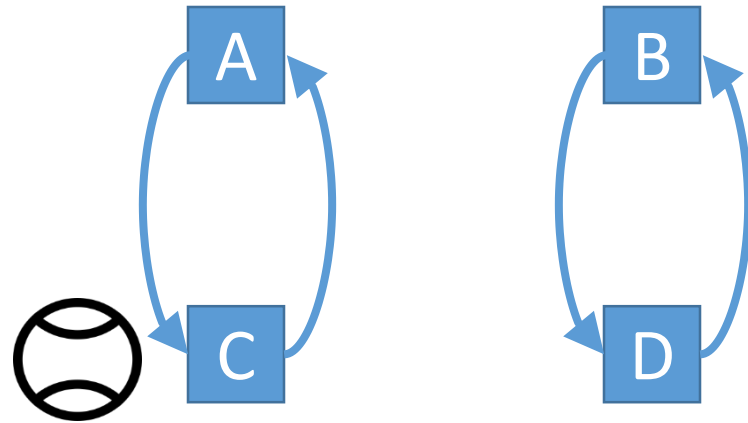
Induction on a ball game

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



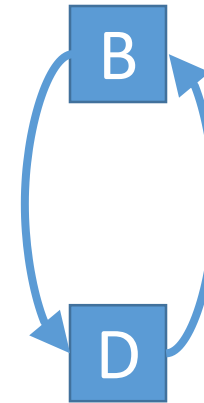
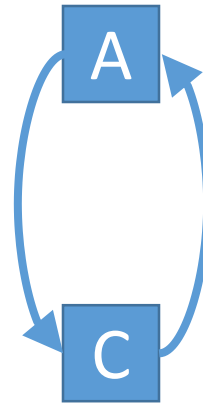
Induction on a ball game

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



Formalizing with induction

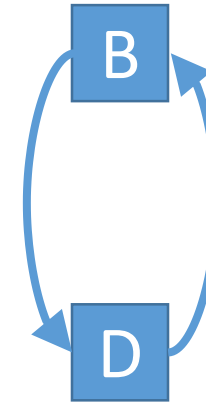
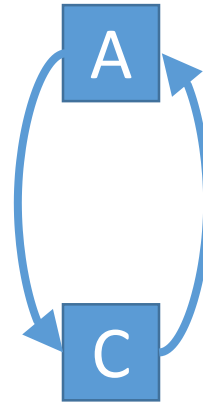
- $x_0 = A$
- $x_{n+1} = \begin{cases} C & \text{if } x_n = A \\ D & \text{if } x_n = B \\ A & \text{if } x_n = C \\ B & \text{if } x_n = D \end{cases}$
- Prove by induction $\forall n. x_n \neq D$
 - $x_0 \neq D$?
 - $x_m \neq D \Rightarrow x_{m+1} \neq D$?



What is a CTI here?

Formalizing with induction

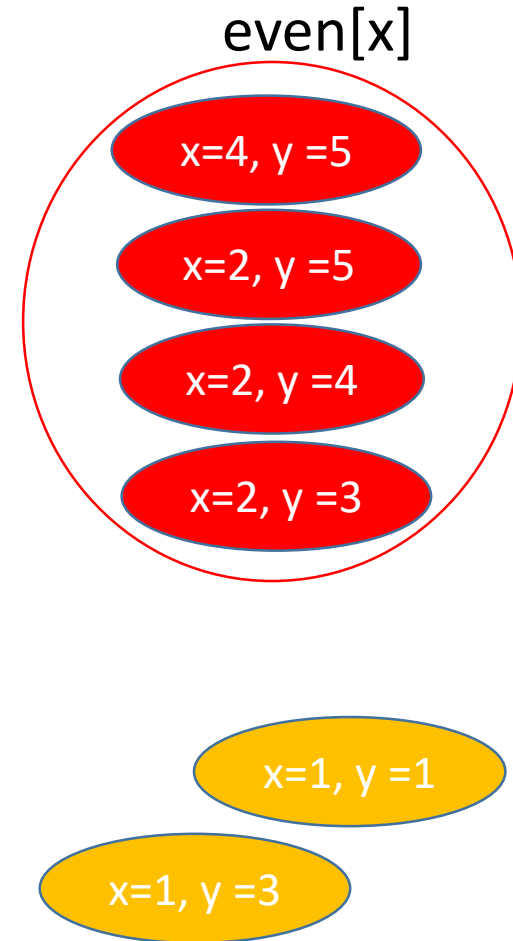
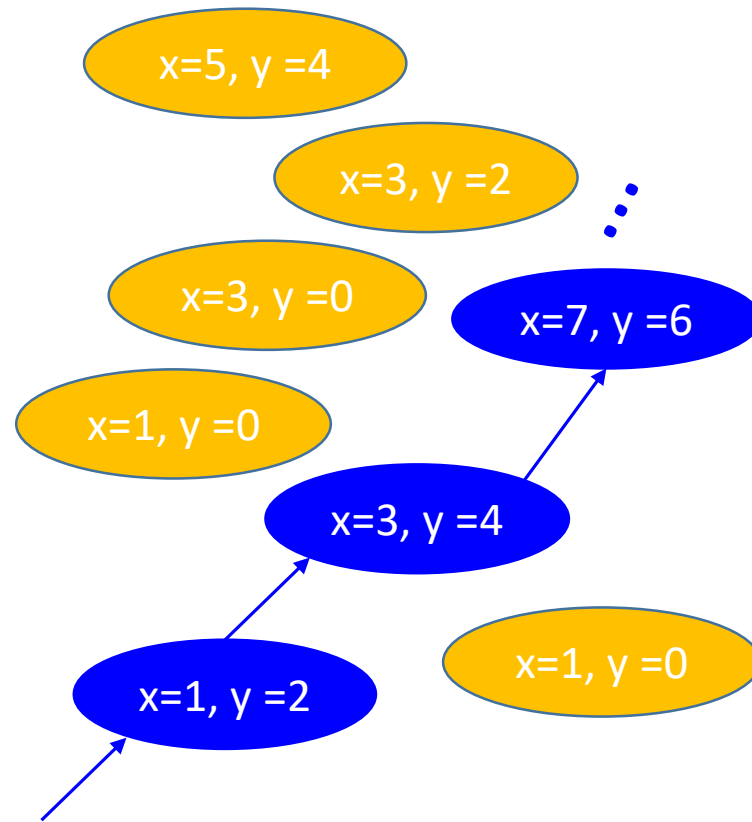
- $x_0 = A$
- $x_{n+1} = \begin{cases} C & \text{if } x_n = A \\ D & \text{if } x_n = B \\ A & \text{if } x_n = C \\ B & \text{if } x_n = D \end{cases}$



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

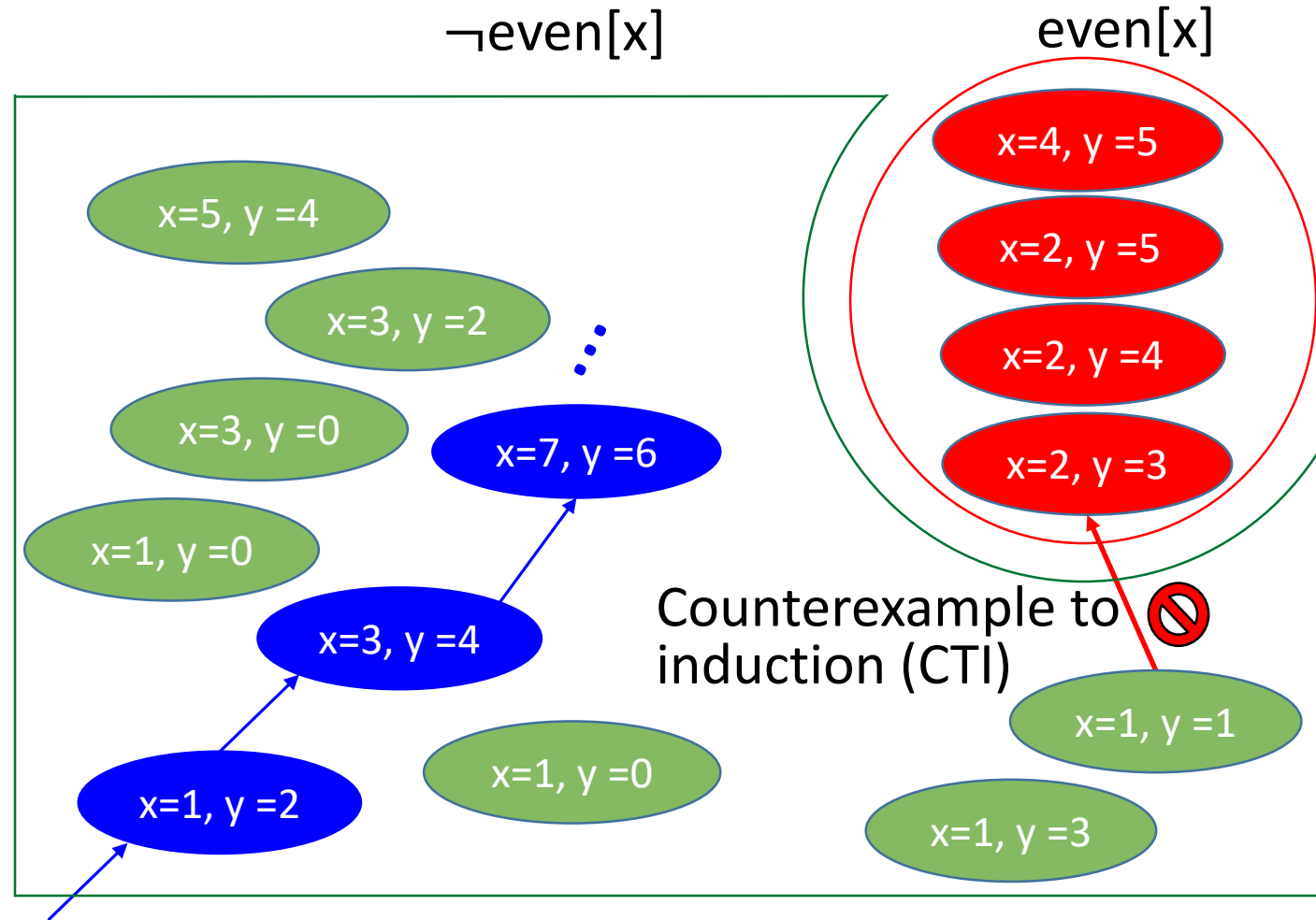
Simple example: loop invariants

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



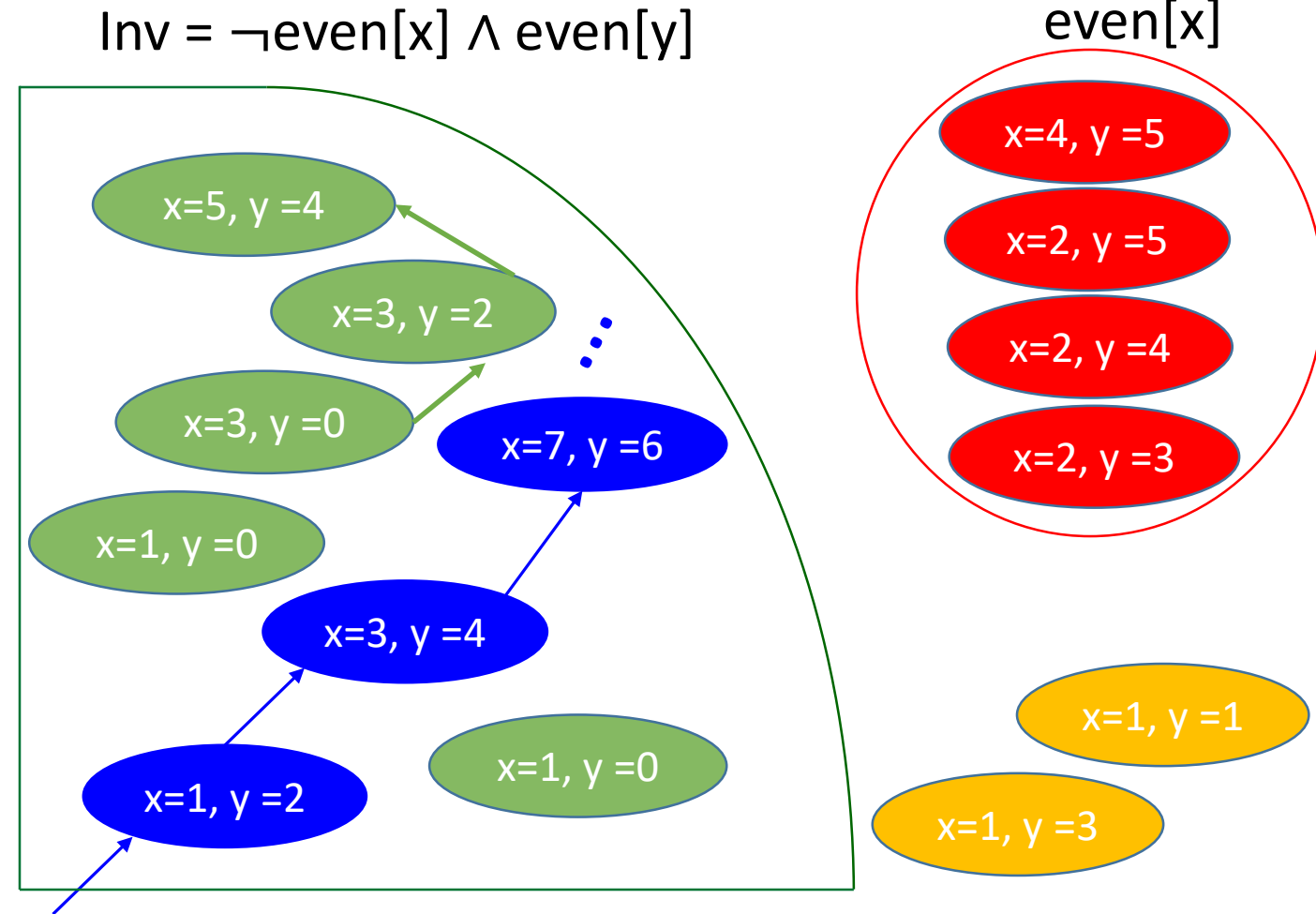
Simple example: loop invariants

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



Simple example: loop invariants

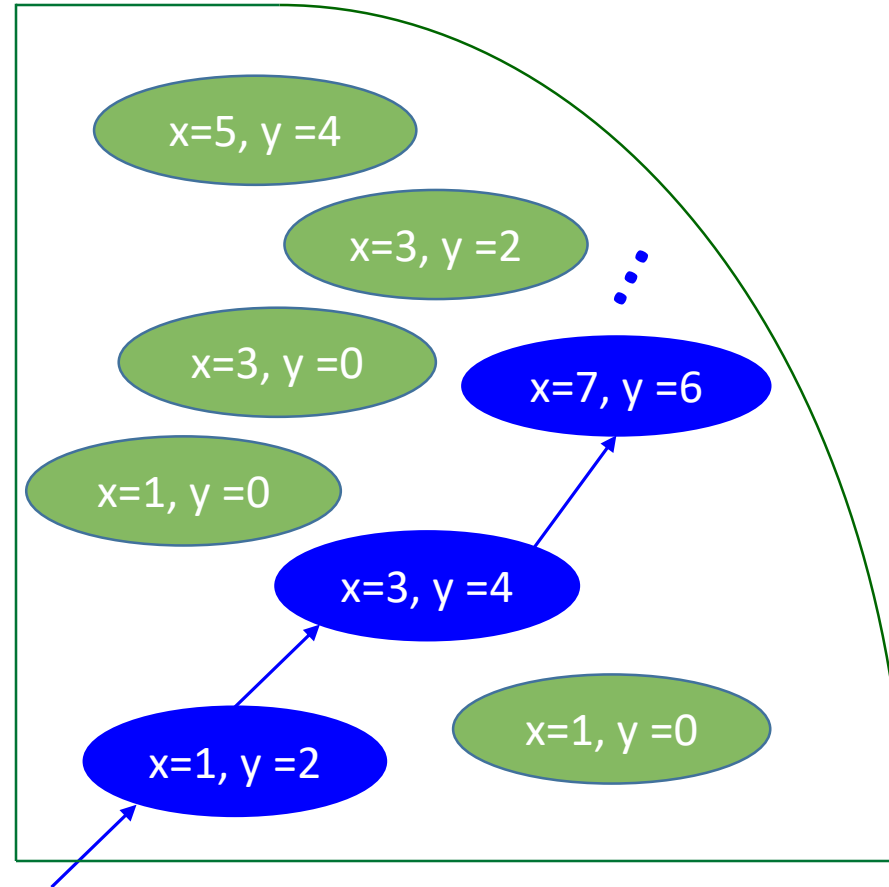
```
x := 1;  
y := 2;  
while * do {  
  assert  $\neg$ even[x];  
   $TR \left| \begin{array}{l} x := x + y; \\ y := y + 2; \end{array} \right. \}$ 
```



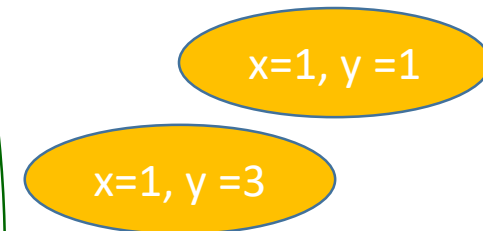
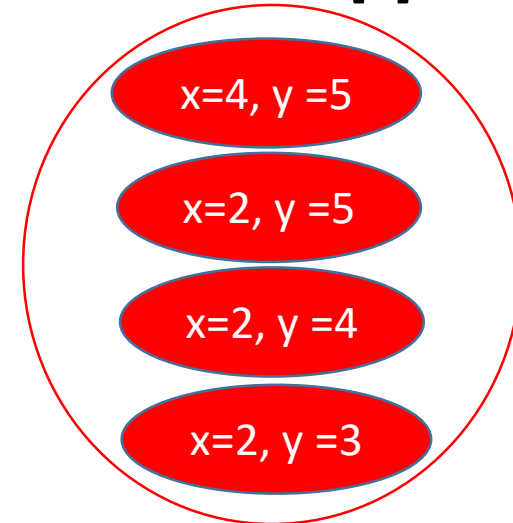
Simple example: loop invariants

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

Inv = \neg even[x] \wedge even[y]



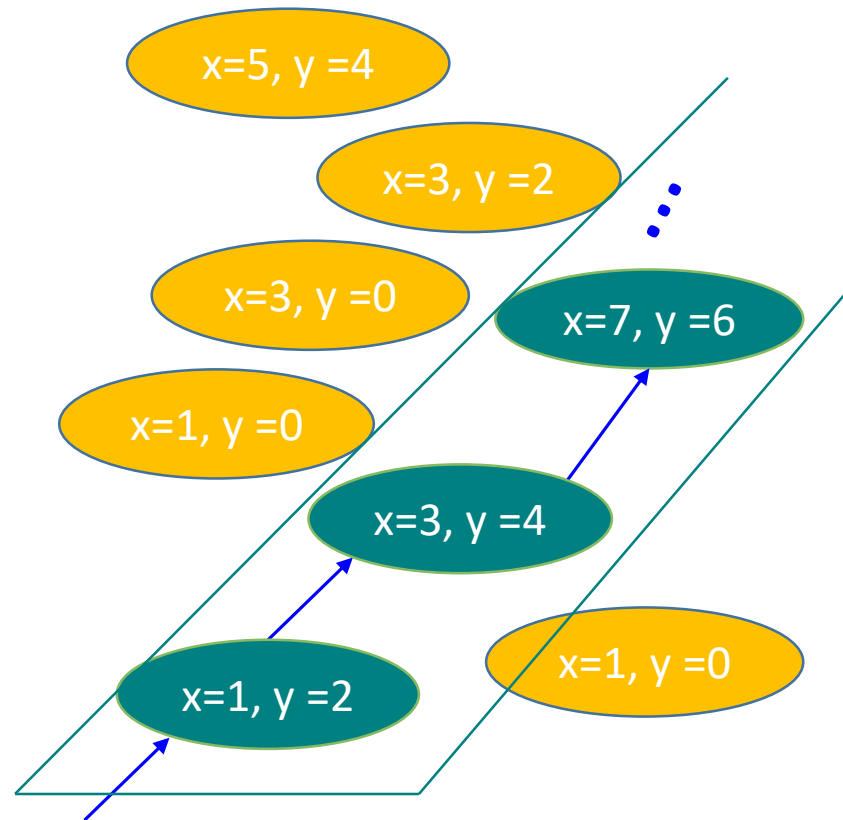
even[x]



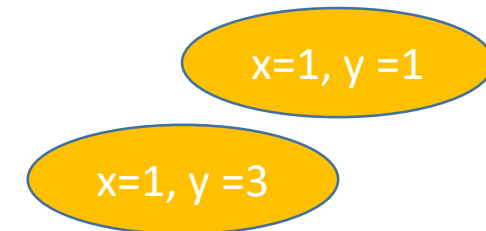
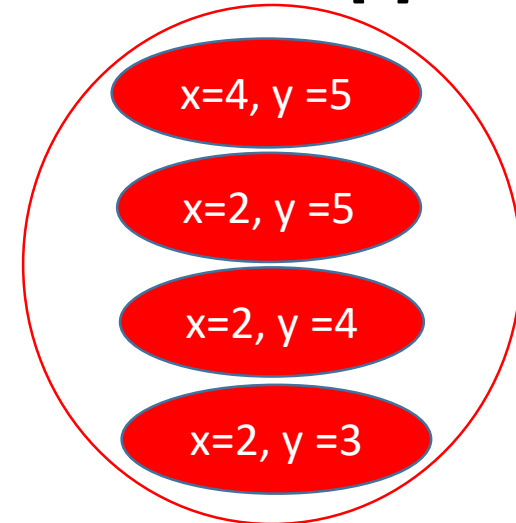
Simple example: loop invariants

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

$$\text{Inv} = y^2 - 2y - 4x + 4 = 0$$

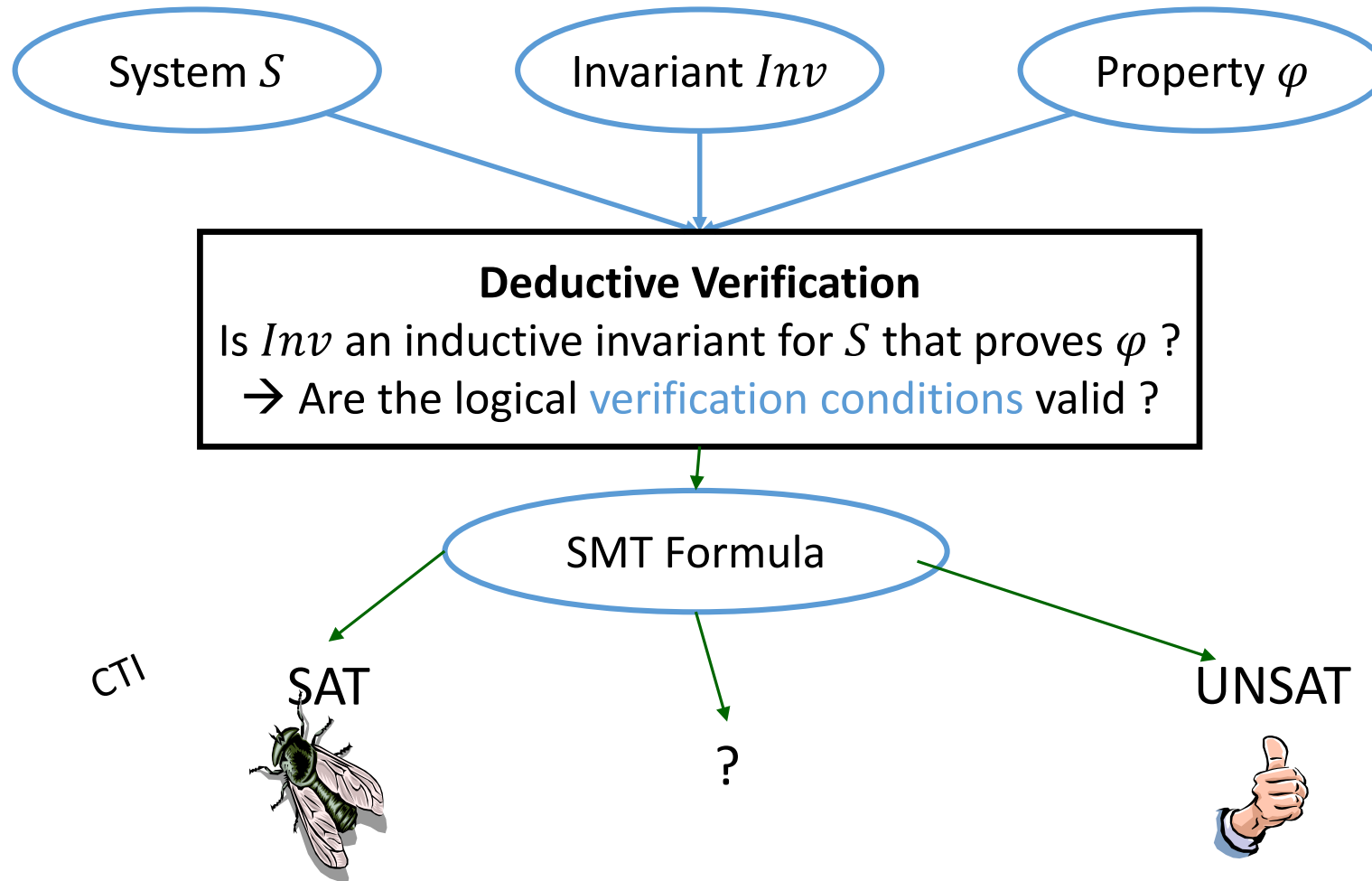


even[x]





Dafny [Leino'17]

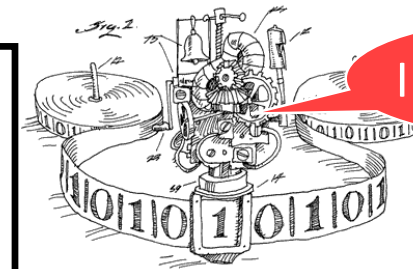


Deductive verification



Deductive Verification
Is Inv an inductive invariant for S that proves φ ?
→ Are the logical **verification conditions** valid ?

Church's Theorem



I can't decide!

Counter-model



Unknown / Diverge

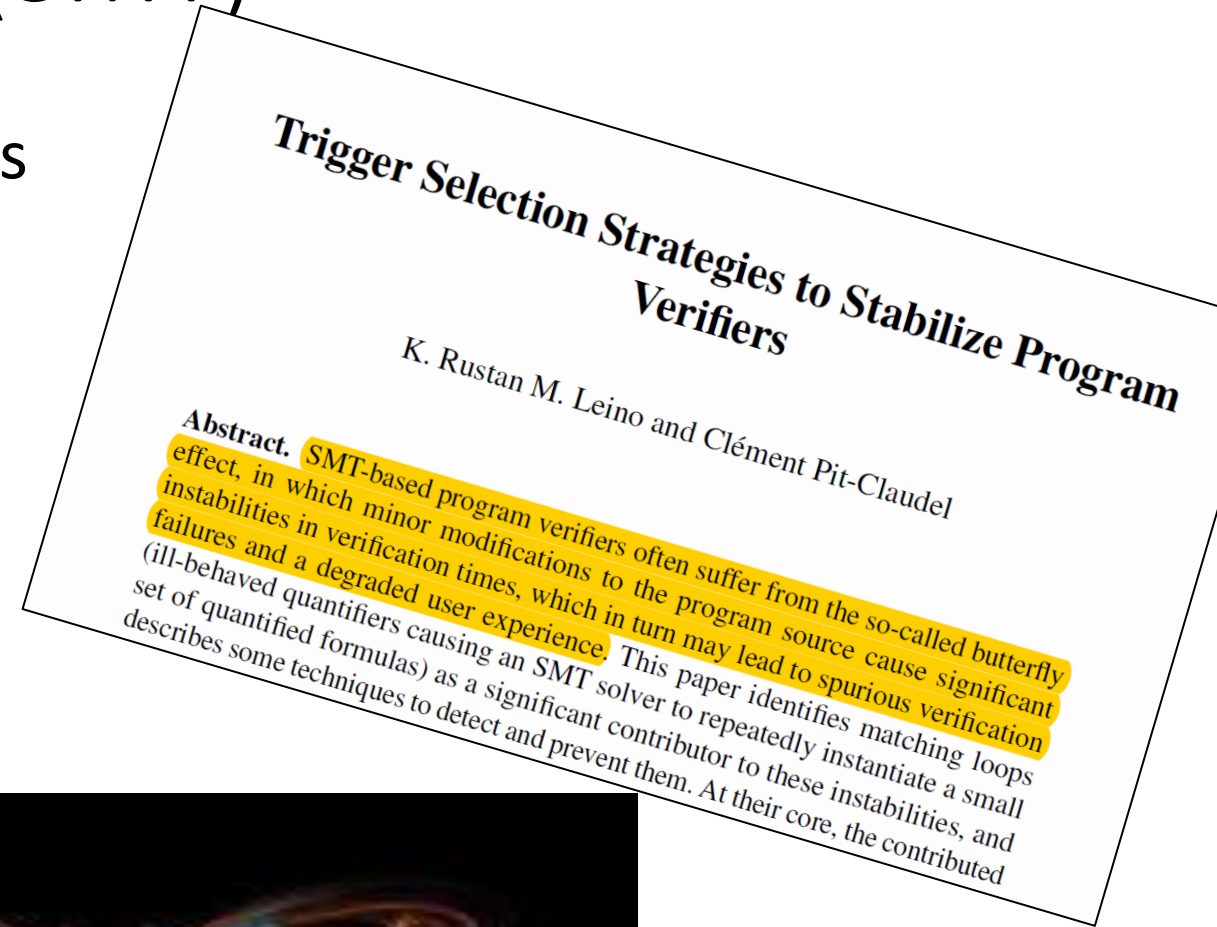


Proof



Effects of undecidability(SMT)

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



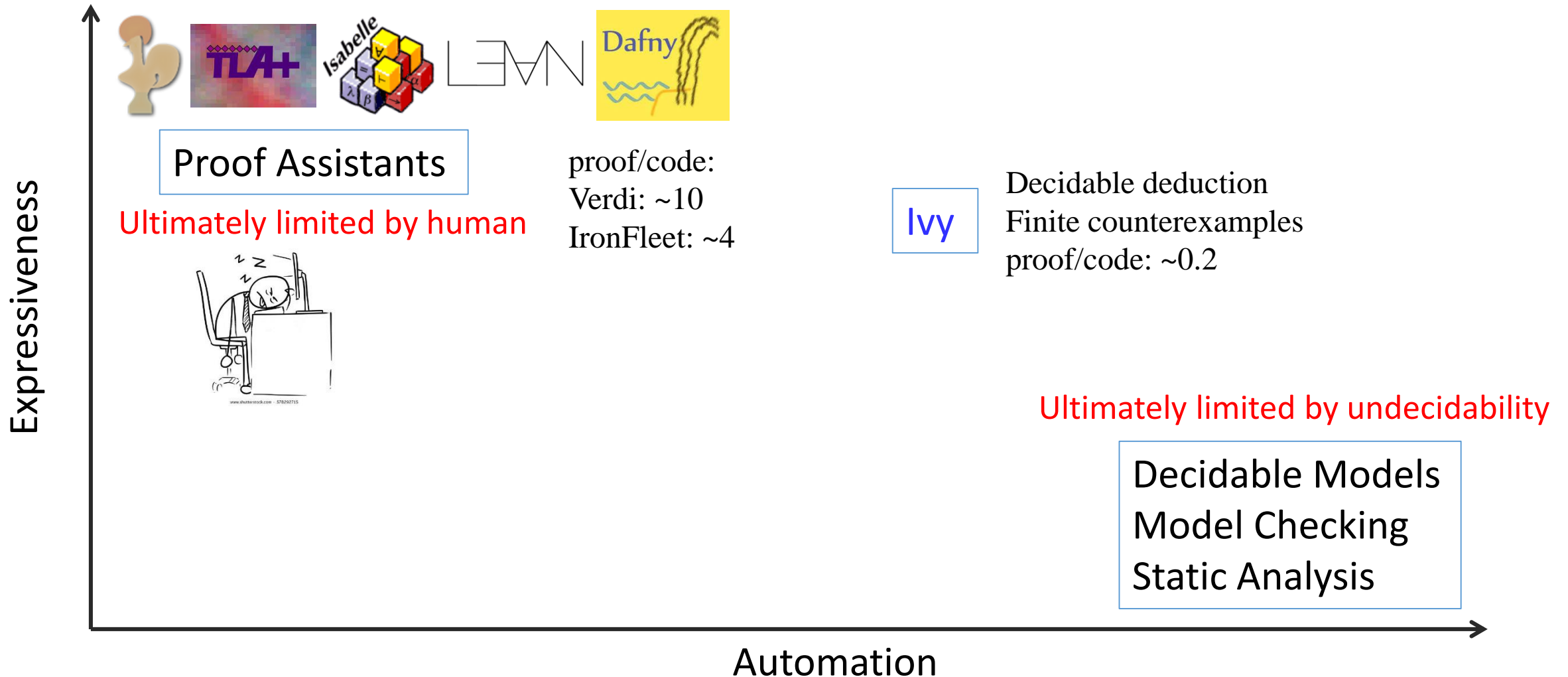
Copyright: Michael Hanke



Challenges in deductive verification

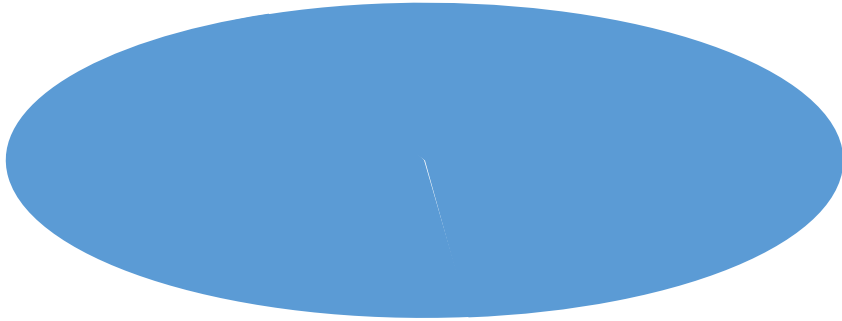
1. **Formal specification**: formalizing infinite-state systems and their **properties**
2. **Deduction**: checking inductiveness
 - Undecidability of implication checking
 - Unbounded state (threads, messages), arithmetic, quantifier alternation
3. **Inference**: finding **inductive invariants** (Inv)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

State of the art in formal verification

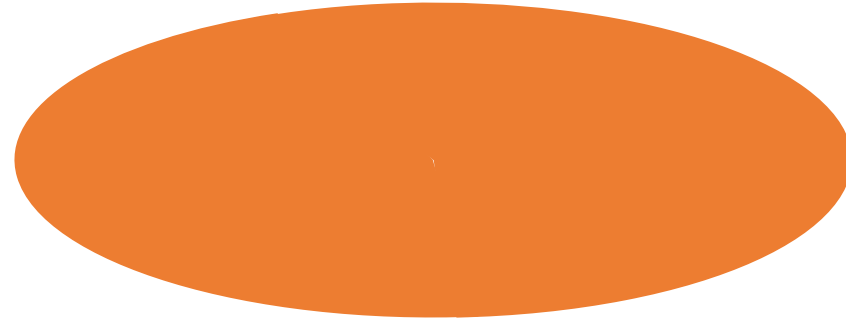


Modularity

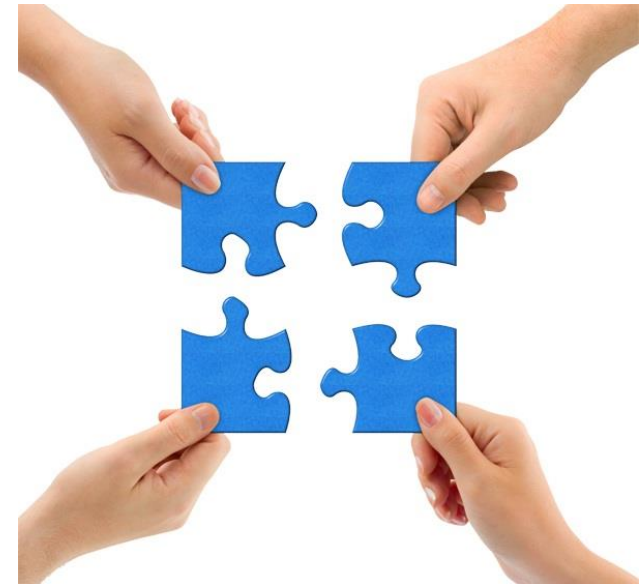
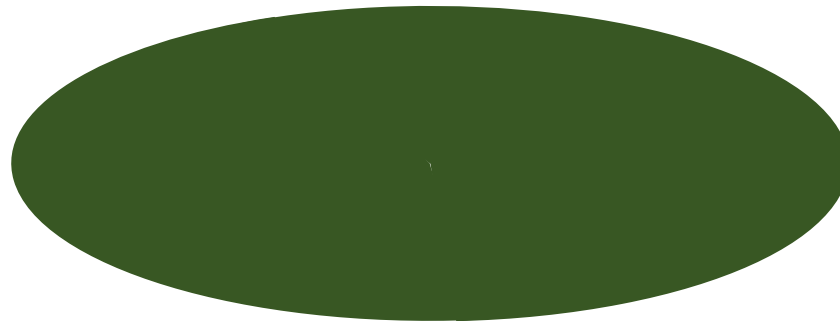
Original system



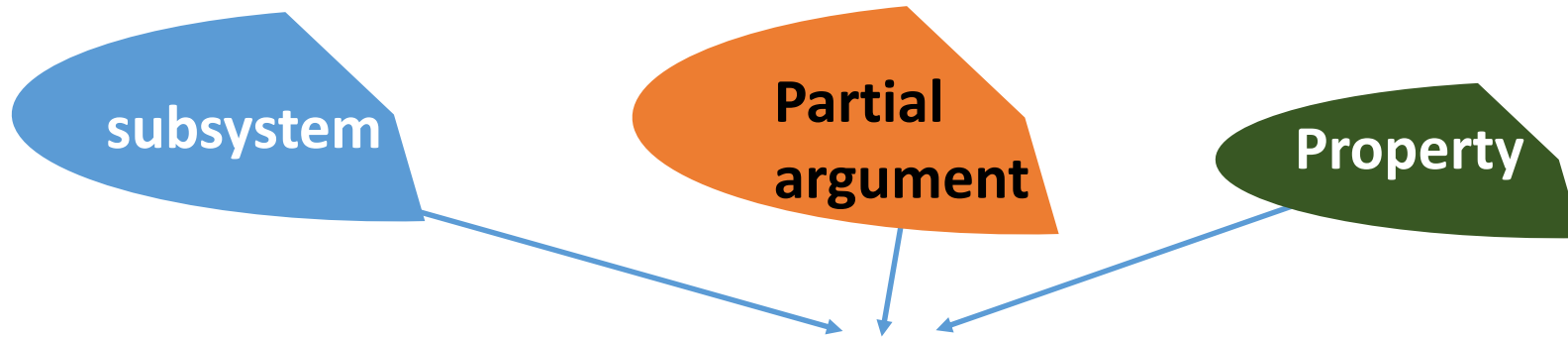
Original inductive argument



Original property



Verification of each module



Verification tool

Incorrect
Finds bug



Correct
Finds proof



**NO
UNDECIDABILITY**
😊

Ivy's principles

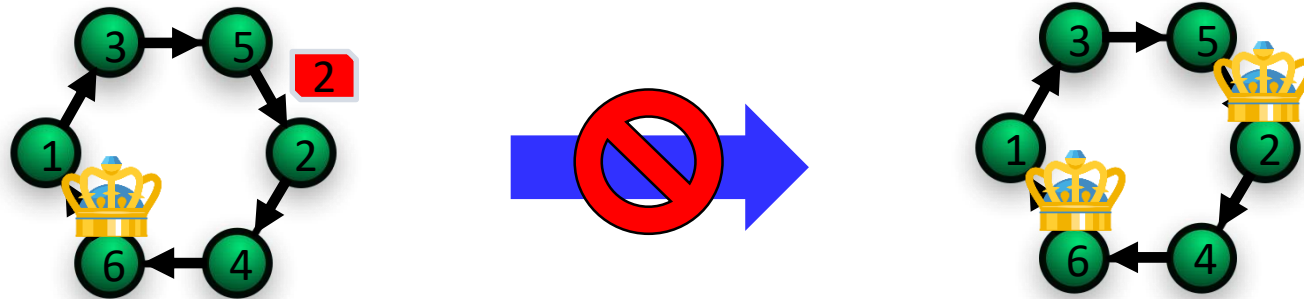
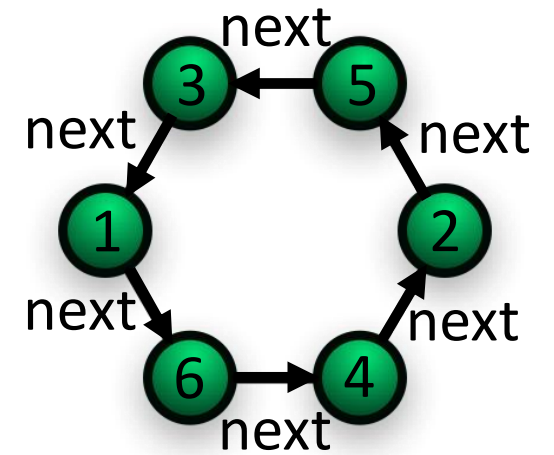
- **Modularity**
 - The user breaks the verification system into small problems expressed in decidable logics
 - The system explores circular assume/guarantee reasoning to prove correctness
- **Inductive invariants and transition systems are expressed in decidable logics**
 - Turing complete imperative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

Languages and verification

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

Example: Leader election in a ring

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



Example: Leader election in a ring

- Unidirectional ring of nodes, unique numeric ids

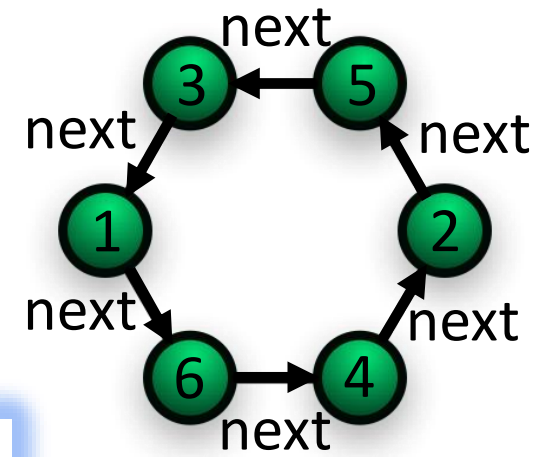
- Protocol:

- Each node sends its id to the next
- Upon receiving a message, a node passes it (to the next) if the id

Proposition: This algorithm detects one and only one highest number.

- Theorem

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.

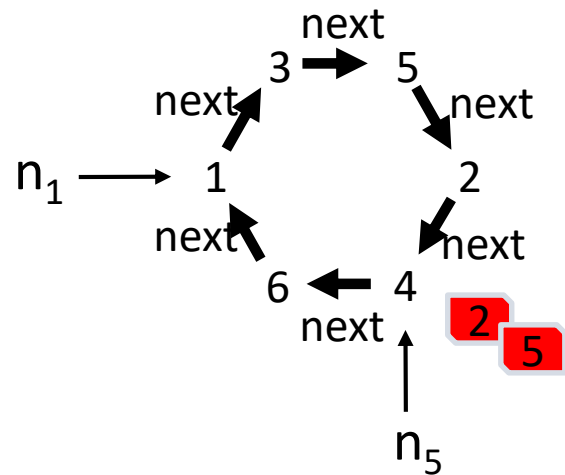


Leader election protocol – first-order logic

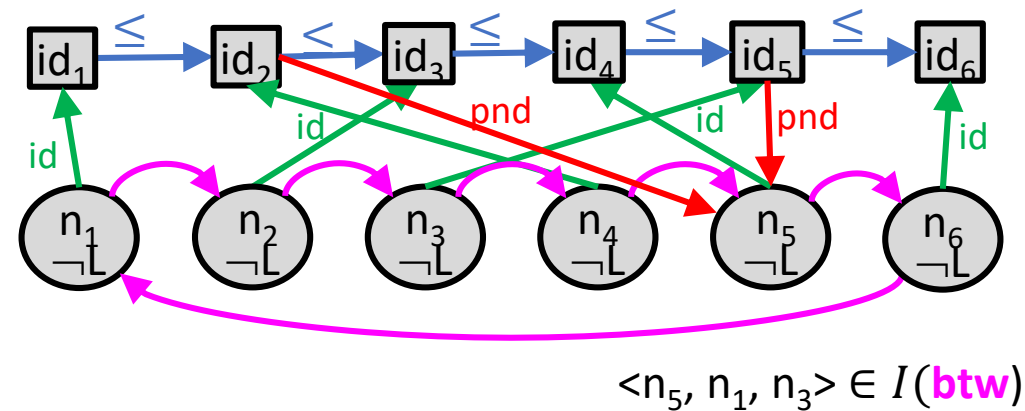
- \leq (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node \rightarrow ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state



first-order structure



Leader election protocol – first-order logic

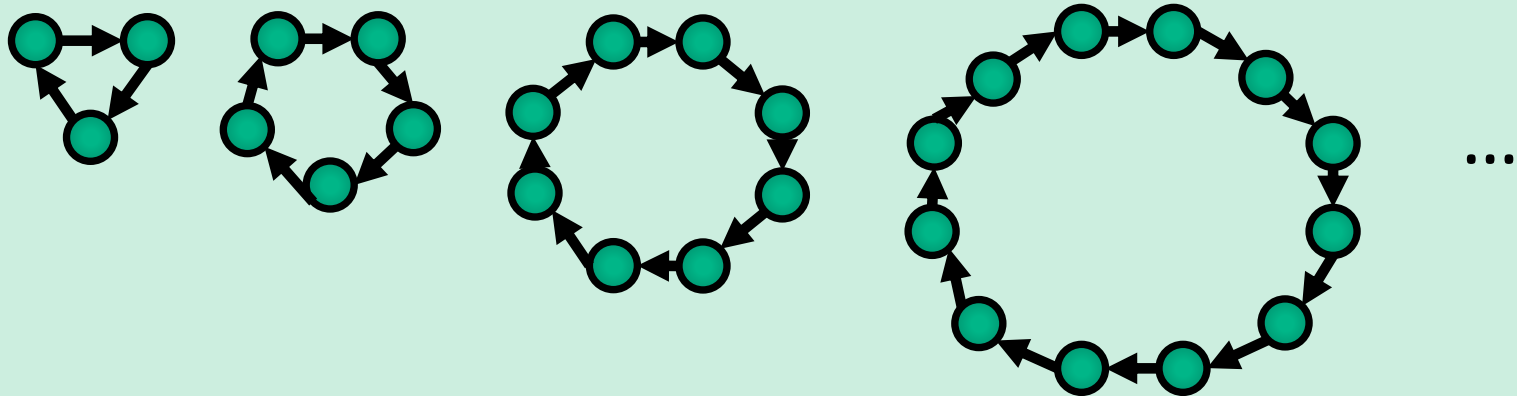
- \leq (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node \rightarrow ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

} Axiomatized in first-order logic

protocol state

first-order structure

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



Leader election protocol – first-order logic

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

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

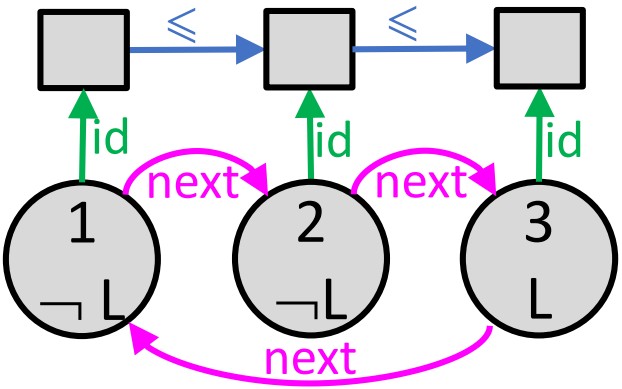
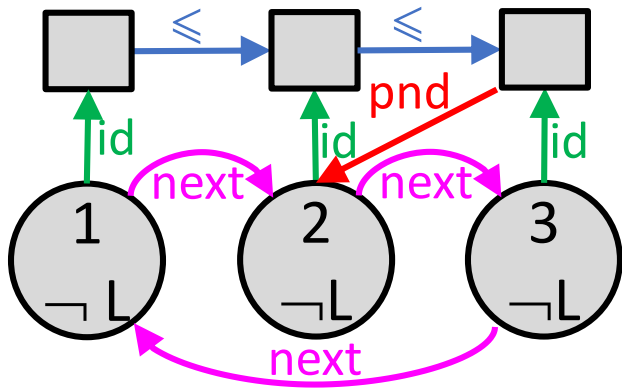
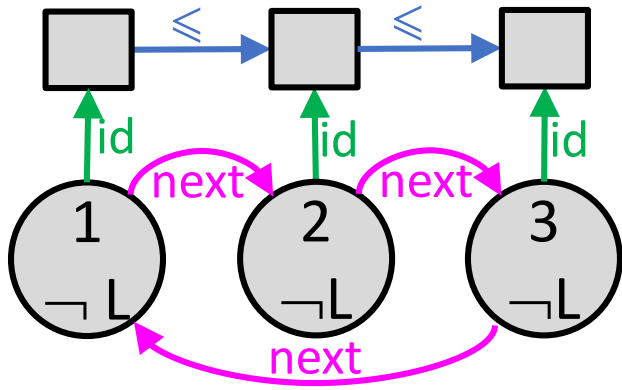
```
action receive(n: Node, m: ID) = {  
  requires pending(m, n);  
  if id(n) = m then  
    // found leader  
    leader(n) := true  
  else if id(n)  $\leq$  m then  
    // pass message  
    "s := next(n)";  
    pending(m, s) := true  
}
```

TR(send):

$\exists n, s: \text{Node}. "s = \text{next}(n)" \wedge \forall x: \text{ID}, y: \text{Node}. \text{pending}'(x, y) \leftrightarrow (\text{pending}(x, y) \vee (x = \text{id}(n) \wedge y = s))$

Bad:

$\text{assert } I0 = \forall x, y: \text{Node}. \text{leader}(x) \wedge \text{leader}(y) \rightarrow x = y$

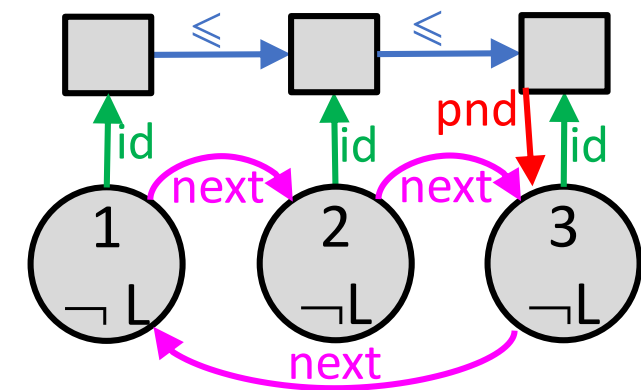
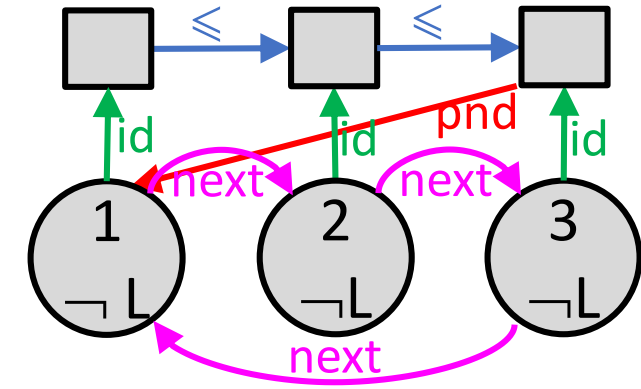


send(3)

rcv(1, id(3))

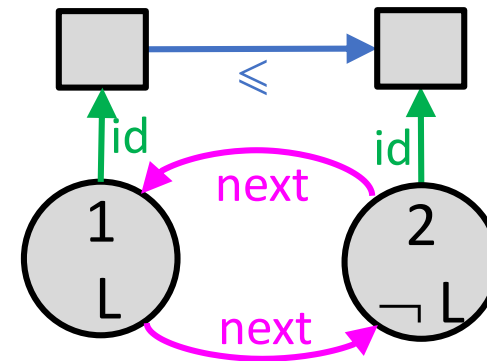
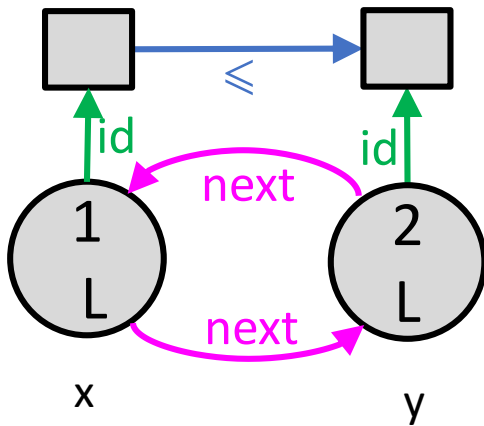
rcv(2, id(3))

rcv(3, id(3))



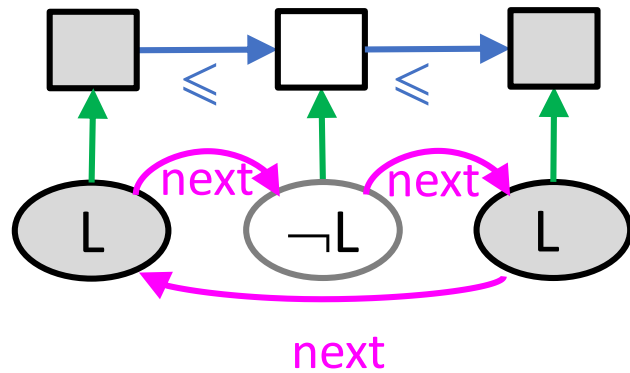
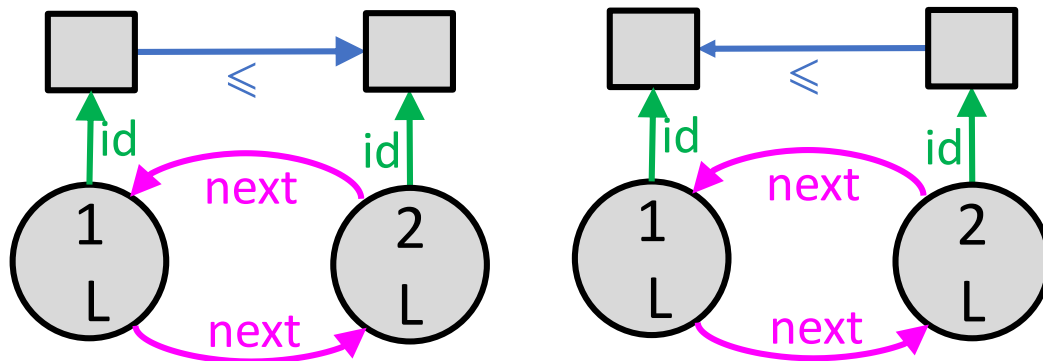
Representing Sets of States with First Order Formulas

- Configurations with at least two leaders
 - $\exists X, Y: \text{Node}. \mathbf{leader}(X) \wedge \mathbf{leader}(Y) \wedge X \neq Y$



Representing Sets of States with First Order Formulas

- Configurations with at least two leaders
 - $\exists X, Y: \text{Node}. \mathbf{leader}(X) \wedge \mathbf{leader}(Y) \wedge X \neq Y$



...

Leader election protocol – inductive invariant

Safety property: I_0

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

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

$$I_1 = \forall n_1$$

$$I_2 = \forall n_1$$

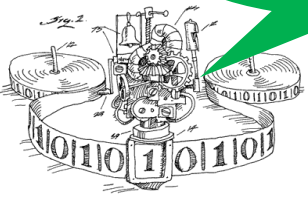
$$I_3 = \forall n_1, n_2, n_3: \text{Node}. \text{btw}(n_1, n_2, n_3) \wedge \text{pending}(\text{id}[n_2], n_1) \rightarrow \text{id}[n_1] < \text{id}[n_2]$$

How can we find an inductive invariant without knowing it?

can be self-pending

I can decide EPR!

cannot bypass higher nodes



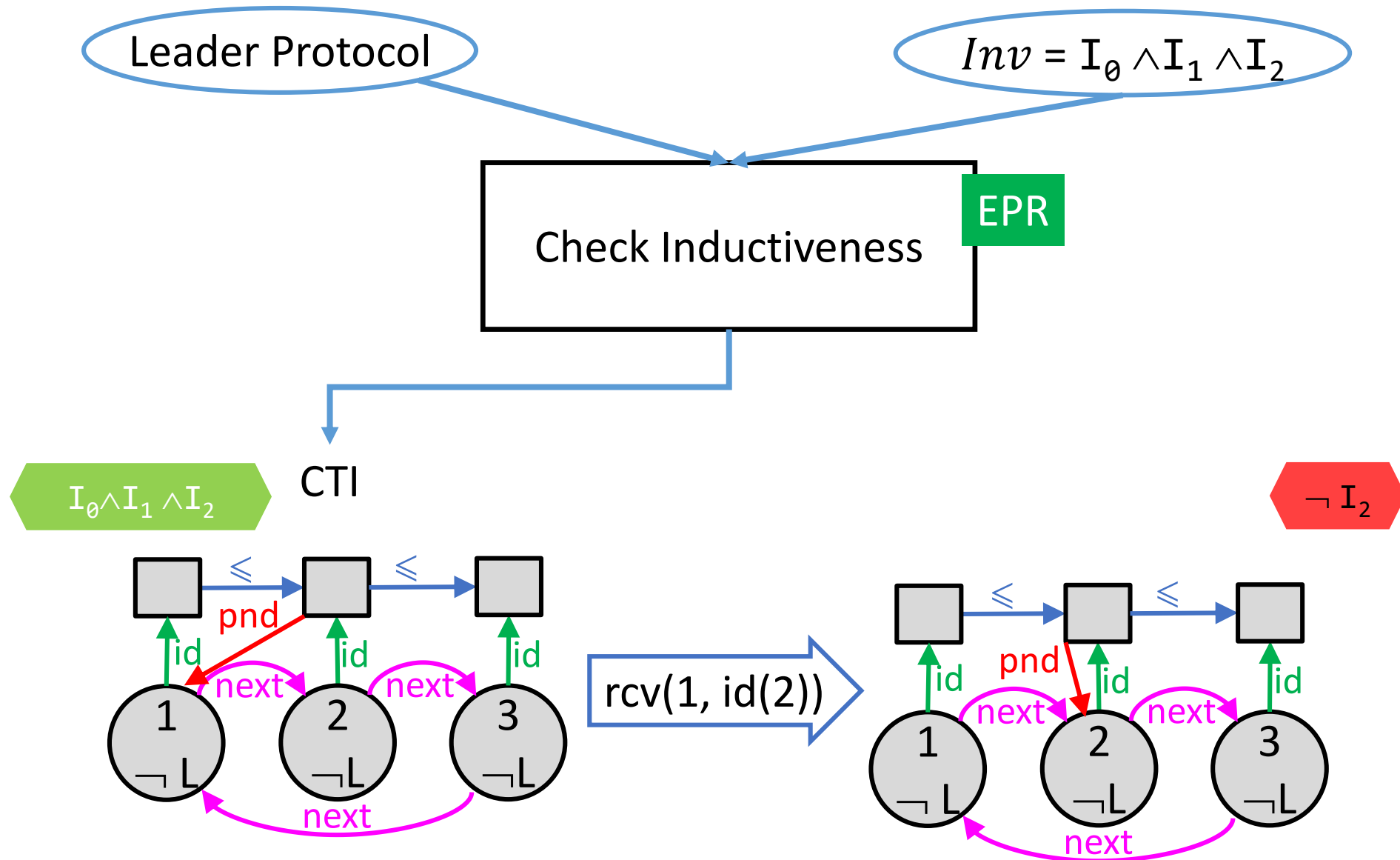
- $\leq (ID, ID)$ – total order on node id's
- **VC Generator** – $\text{Init}(V) \wedge \neg \text{Inv}(V) \rightarrow \text{Inv}(V')$
- **id**: Node \rightarrow ID – relate a node to its ID
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

EPR Solver

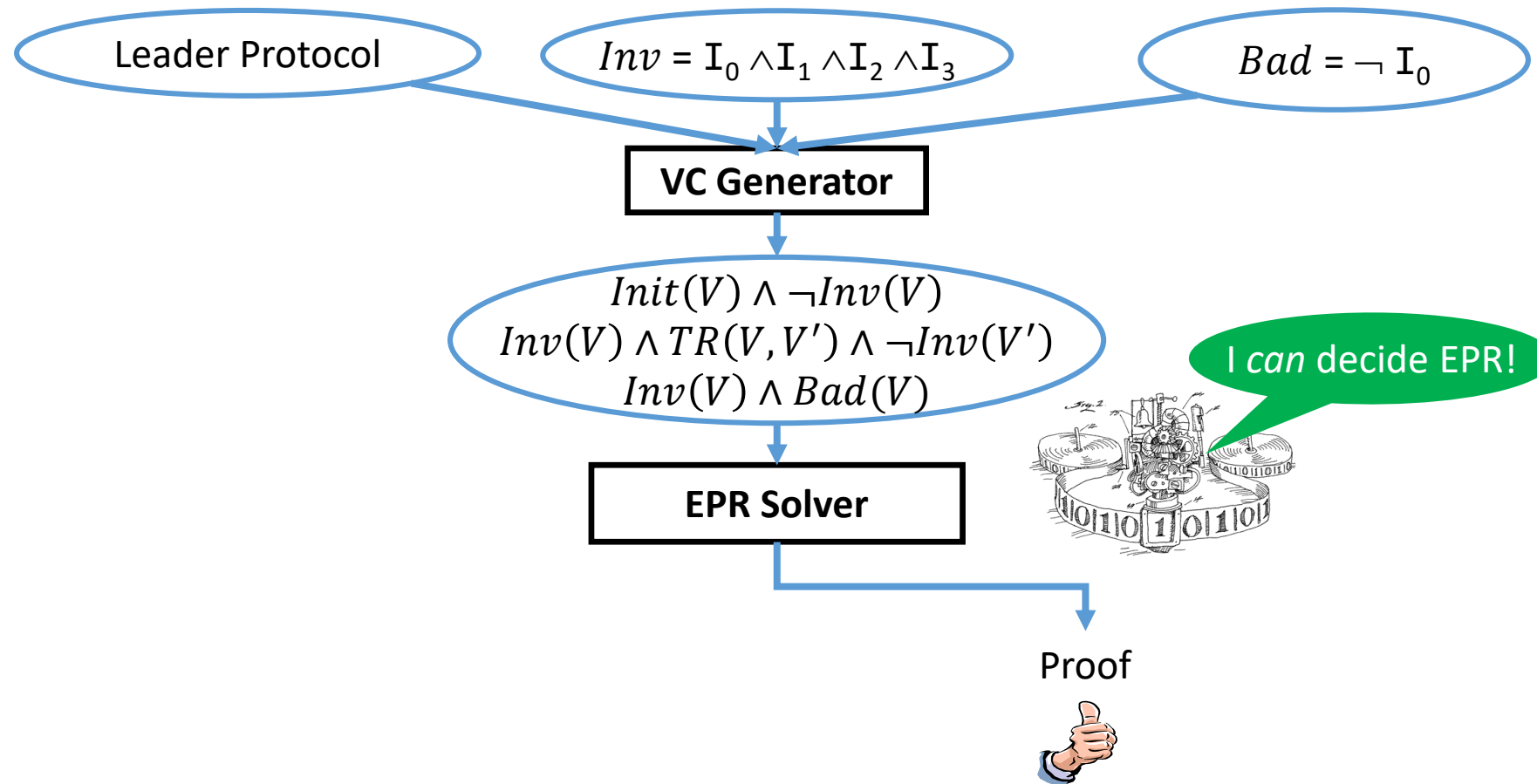


Proof

Ivy: check inductiveness

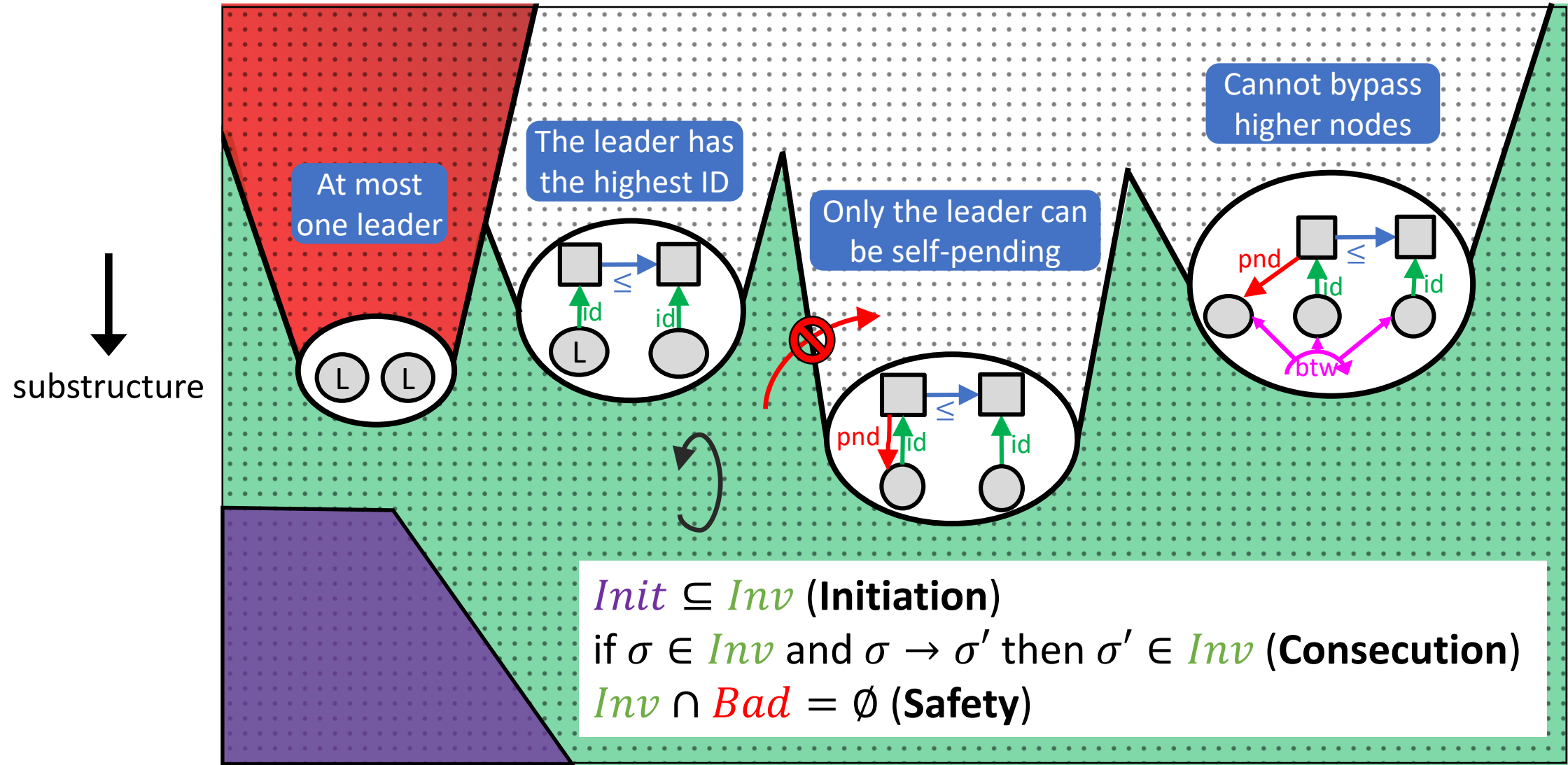


Ivy: check inductiveness



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

\forall^* invariant – excluded substructures



Summary

- Distributed protocols are interesting for verification
 - But real distributed systems are more complex
- Can be naturally modeled in pure first order logic
- Decidable logics can be used to reason about interesting systems
 - **No more butterfly effects**
 - But some jagged corners
 - Details on Wednesday