Ivy: Safety Verification by Interactive Generalization

Oded Padon

Verification Day

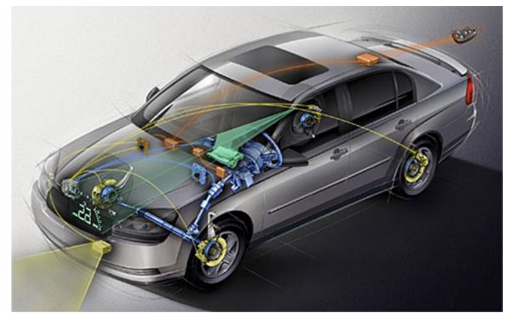
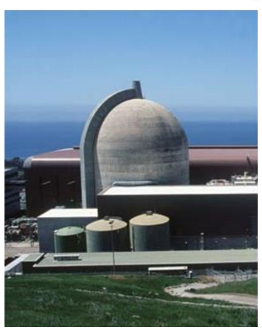
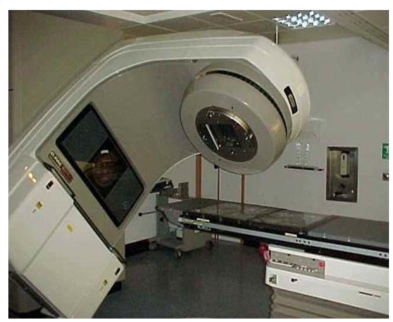
1-June-2016

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

Motivation

- Software is everywhere
- Distributed systems are everywhere
- Verification is needed to ensure safety of critical systems



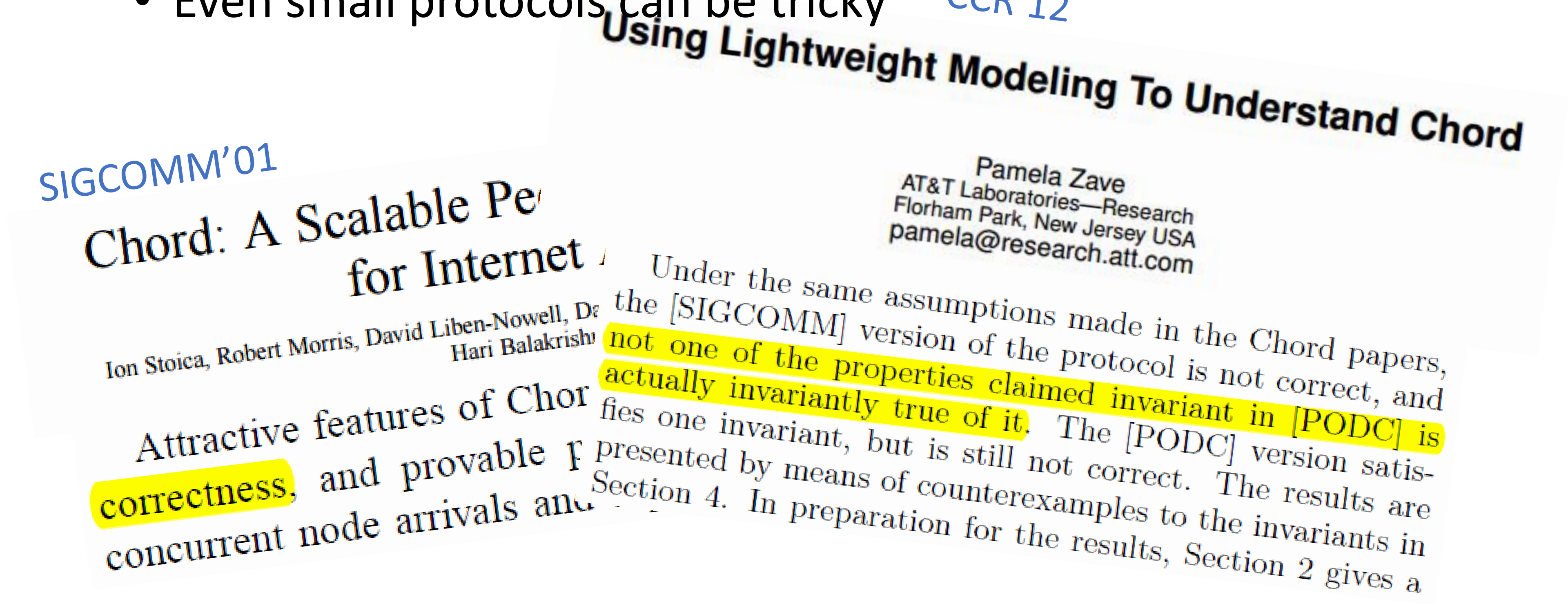





Why Verify Distributed Systems?

- Distributed systems are notoriously hard to get right
- Bugs occur on rare scenarios
  - Hard to test/reproduce
    - Testing covers tiny fraction of behaviors
    - Leaves most bugs for production
- Even small protocols can be tricky

CCR'12

SIGCOMM'01


Using Lightweight Modeling To Understand Chord

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

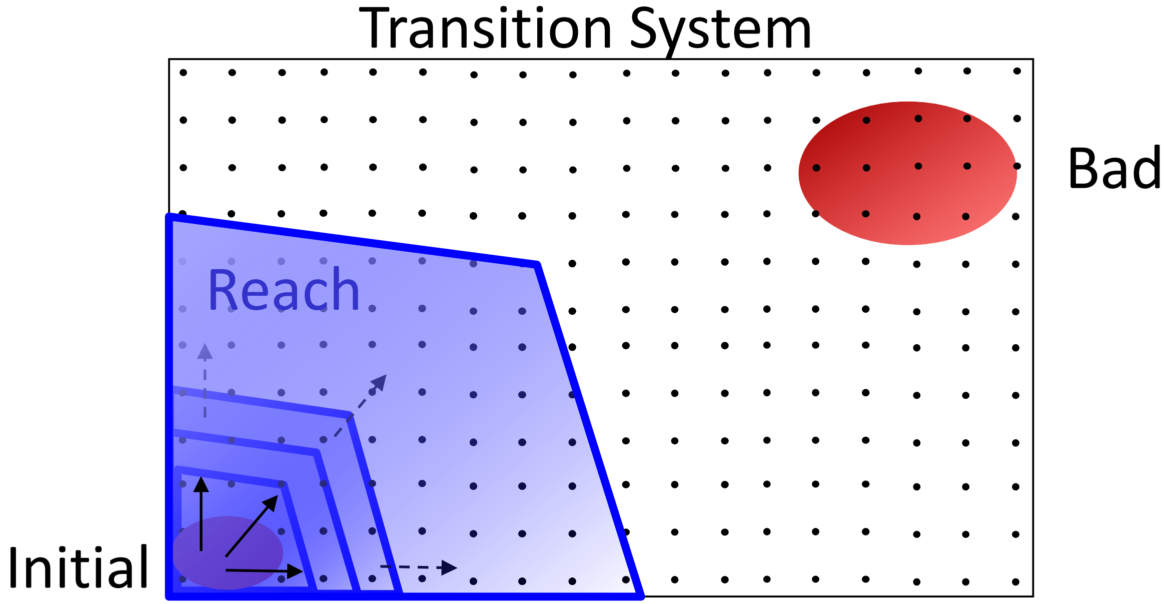
Under the same assumptions made in the Chord papers, the [SIGCOMM] version of the protocol is not correct, and not one of the properties claimed invariant in [PODC] is actually invariantly true of it. The [PODC] version satisfies one invariant, but is still not correct. The results are presented by means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

Chord: A Scalable Pe
for Internet

Ion Stoica, Robert Morris, David Liben-Nowell, Da
Hari Balakrishn

Attractive features of Chor
correctness, and provable p
concurrent node arrivals an

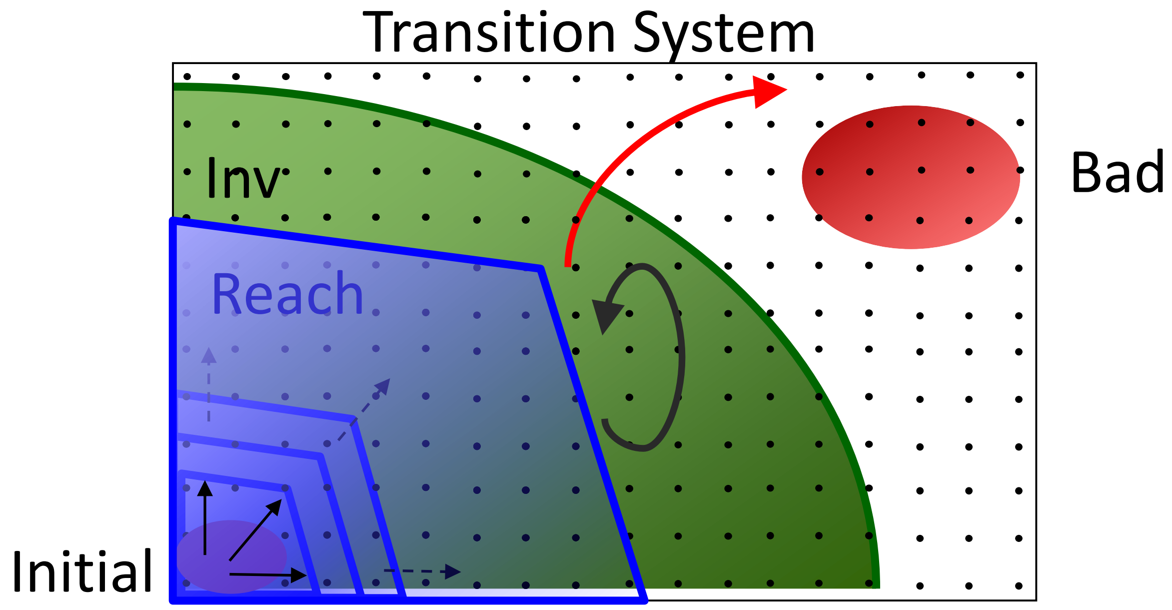
Safety of Transition Systems



System S is **safe** if no bad state is reachable

- $R_0 = Init$ – Initial states, reachable in 0 transitions
- $R_{i+1} = R_i \cup \{\sigma' \mid \sigma \rightarrow \sigma' \text{ and } \sigma \in R_i\}$
- $R = R_0 \cup R_1 \cup R_2 \cup \ldots$
- Safety: $R \cap Bad = \emptyset$
- K-Safety: $R_K \cap Bad = \emptyset$

Inductive Invariants



System S is safe if no bad state is reachable

System S is safe iff there exists an inductive invariant Inv s.t.:

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

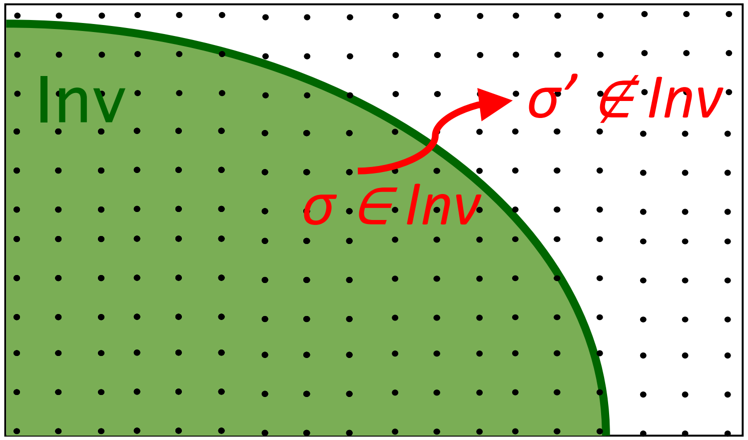
Init $\subseteq$ *Inv (Initiation)*

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

Counterexample To Induction (CTI)

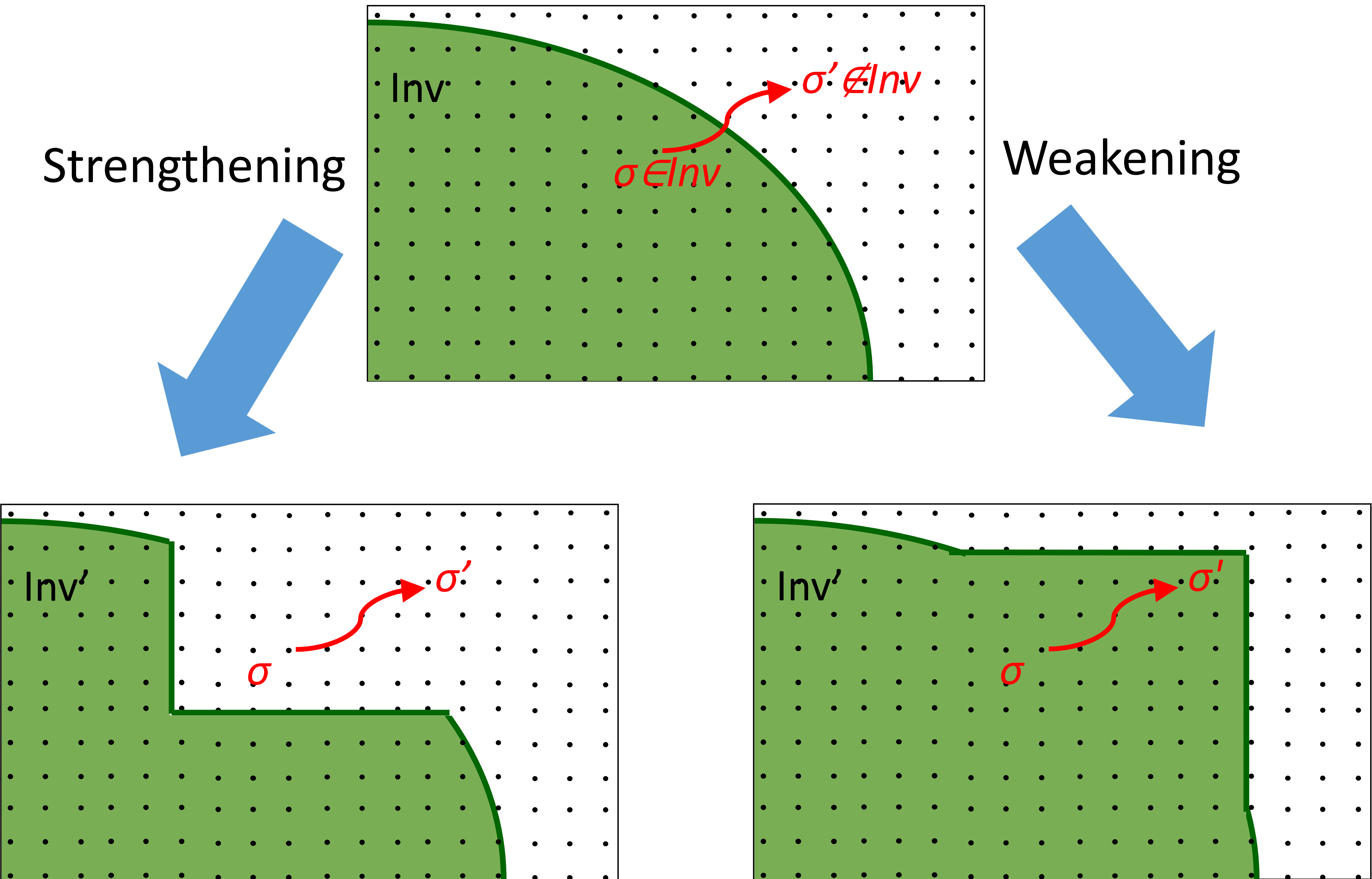
States σ,σ’ are a CTI of Inv if:

- σ ∈ Inv
- σ’ ∉ Inv
- σ → σ’



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

Strengthening & Weakening from CTI



Modeling with Logic

- SAT/SMT has made huge progress in the last decade
- Great impact on verification:
  Z3, Dafny, IronClad/IronFleet, and more

- State: finite first-order structure over vocabulary V
- Initial states and safety property (first-order formulas):
  - Init(V) – initial states
  - Bad(V) – bad states
- Transition relation:
  first-order formula TR(V, V')
  V' is a copy of V describing the next state

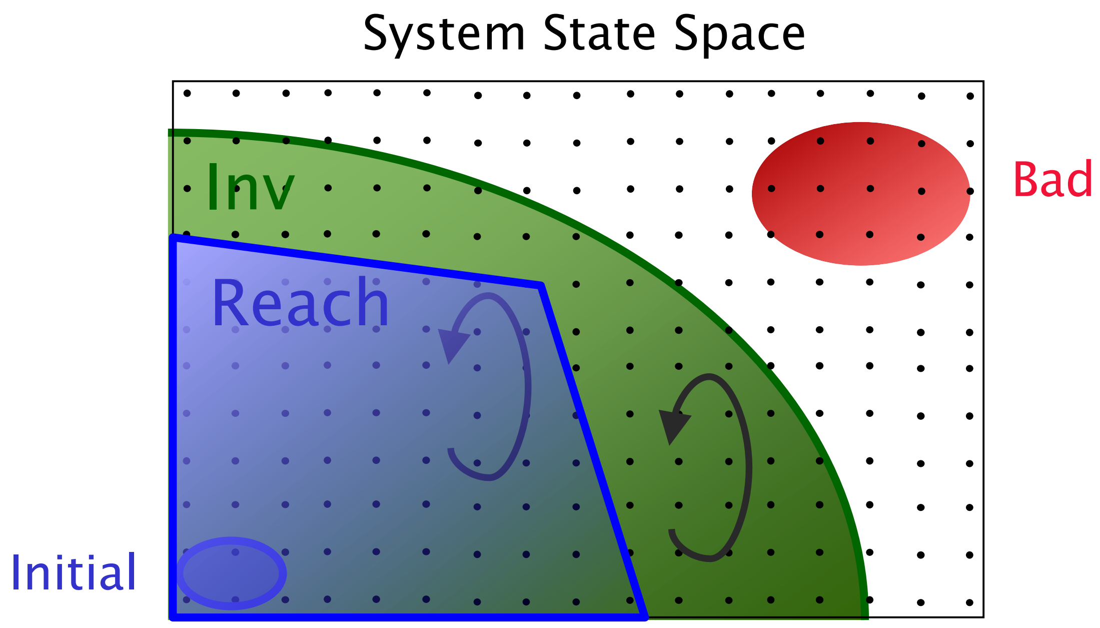
[LPAR'10] K.R.M. Leino: Dafny: An Automatic Program Verifier for Functional Correctness

[SOSP'15] C. Hawblitzel, J. Howell, M. Kapritsos, J.R. Lorch, B. Parno, M. Roberts, S. Setty, B. Zill: IronFleet: proving practical distributed systems correct

Inductive Invariant

Inv is an **inductive invariant** if:

- Initiation: $Init \Rightarrow Inv$ $\quad Init \wedge \neg Inv$ unsat
- Safety: $Inv \Rightarrow \neg Bad$ $\quad Inv \wedge Bad$ unsat
- Consecution: $Inv \wedge TR \Rightarrow Inv'$ $\quad Inv \wedge TR \wedge \neg Inv'$ unsat



Challenges

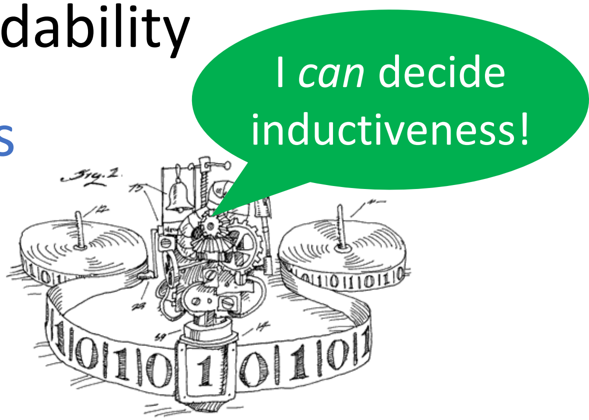
1. Formal specification:
   - Modeling the system (TR, Init)
   - Formalizing the safety property (Bad)
   - Specifying in logic
2. Deduction – Checking inductiveness
   - Undecidability of implication checking
     - Arithmetic, quantifier alternation, unbounded state
3. Inductive Invariants for Deductive Verification (Inv)
   - Hard to specify
   - Hard to infer
     - Undecidable even when deduction is decidable

Existing approaches

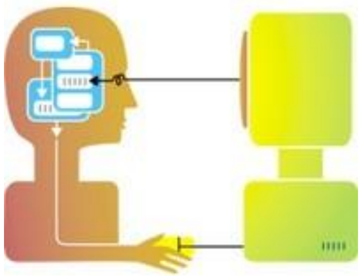
- Automated invariant inference
  - Model checking
    - Exploit finite state / finite abstraction
  - Abstract Interpretation
    - Sound abstraction
  - Limited for infinite state systems due to undecidability
- Use SMT for deduction with manual program annotations (e.g. Dafny)
  - Requires programmer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

Our Approach in Ivy

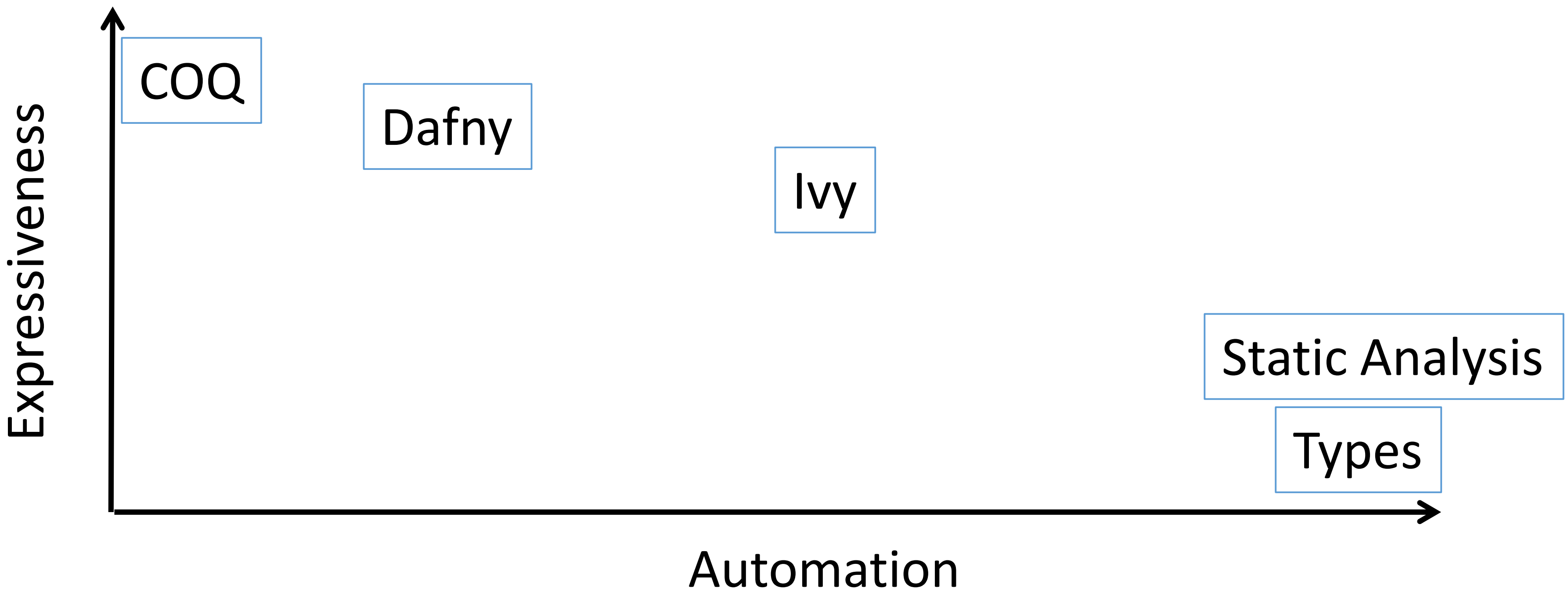
- Restrict the specification language for decidability
  - Deduction is decidable with SAT solvers
  - Challenge: model systems in a restricted language



- Finding inductive invariants:
  - Combine automated techniques with human guidance
  - Key: generalization from counterexamples to induction
  - Graphical user interaction
  - Decidability allows reliable automated checks

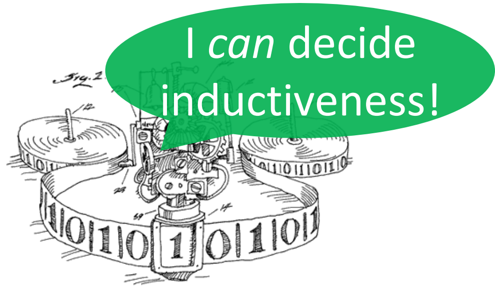


Expressiveness vs. Automation



	Coq	Dafny	Ivy	Static Analysis
Invariant	User	User	User + System	System
Deduction	User	System (Z3) + “User”	System (EPR Z3)	System

Relational Modeling Language (RML)



- Designed to make verification tasks decidable
  - Yet expressive enough to model systems
- Finite relations and stratified function symbols
  - Used to describe system state
    - E.g., pending packets, nodes' local data structures
  - Also used to record ghost information
  - Stratification: function f: A→B, so no function g: B→A
- Universally quantified axioms
  - Total orders, partial orders, lists, trees, rings, quorums, …
- No numerics
- Simple (quantifier-free) updates
- Imperative constructs with non-determinism
- Turing-Complete
- Universal inductive invariants are decidable to check

Effectively Propositional Logic – EPR
a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic
  - Restricted quantifier prefix: $\exists^* \forall^* \phi_{Q.F.}$
  - No $\forall^* \exists^*$
  - No function symbols
    - Possible to add stratified function symbols
  - No arithmetic
- Small model property
  - $\exists x_1, \ldots, x_n. \forall y_1, \ldots, y_m. \phi_{Q.F.}$ has a model iff it has a model of at most n+k elements (k - number of constant symbols)
- Satisfiability is decidable
  - NEXPTIME / Σ_2
- Supported by theorem provers (e.g., Z3, iProver, Vampire)

F. Ramsey. *On a problem in formal logic.* Proc. London Math. Soc. 1930

Using EPR for Verification

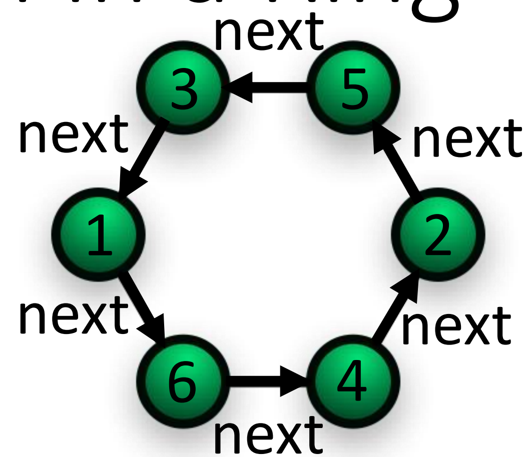
- System Model
  - V – vocabulary with relations and stratified function symbols
  - TR(V, V') – tra $\exists^*\forall^*$
  - Init(V) – initia $\exists^*\forall^*$
  - Bad(V) – bad $\exists^*\forall^*$

Alternation-Free:
Boolean combination of
closed $\exists^*$ and $\forall^*$ formulas

- Inv(V) is an inductive invariant if: A-F
  - $\text{Init}(V) \Rightarrow \text{Inv}(V)$ $\quad \text{Init} \wedge \neg \text{Inv}$ $\quad$ unsat
  - $\text{Inv}(V) \wedge \text{TR}(V,V') \Rightarrow \text{Inv}(V')$ $\quad \text{Inv} \wedge \text{TR} \wedge \neg \text{Inv}'$ $\quad$ unsat
  - $\text{Inv}(V) \Rightarrow \neg \text{Bad}(V)$ $\quad \text{Inv} \wedge \text{Bad}$ $\quad$ unsat

Decidable to check
SAT($\exists^*\forall^*$)

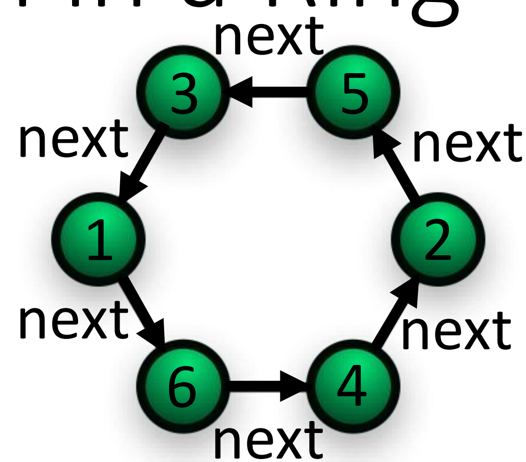
Example: Leader Election in a Ring



- Nodes are organized in a 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*

Example: Leader Election in a Ring



- Nodes are organized in a 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
- Theore
  - The

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.

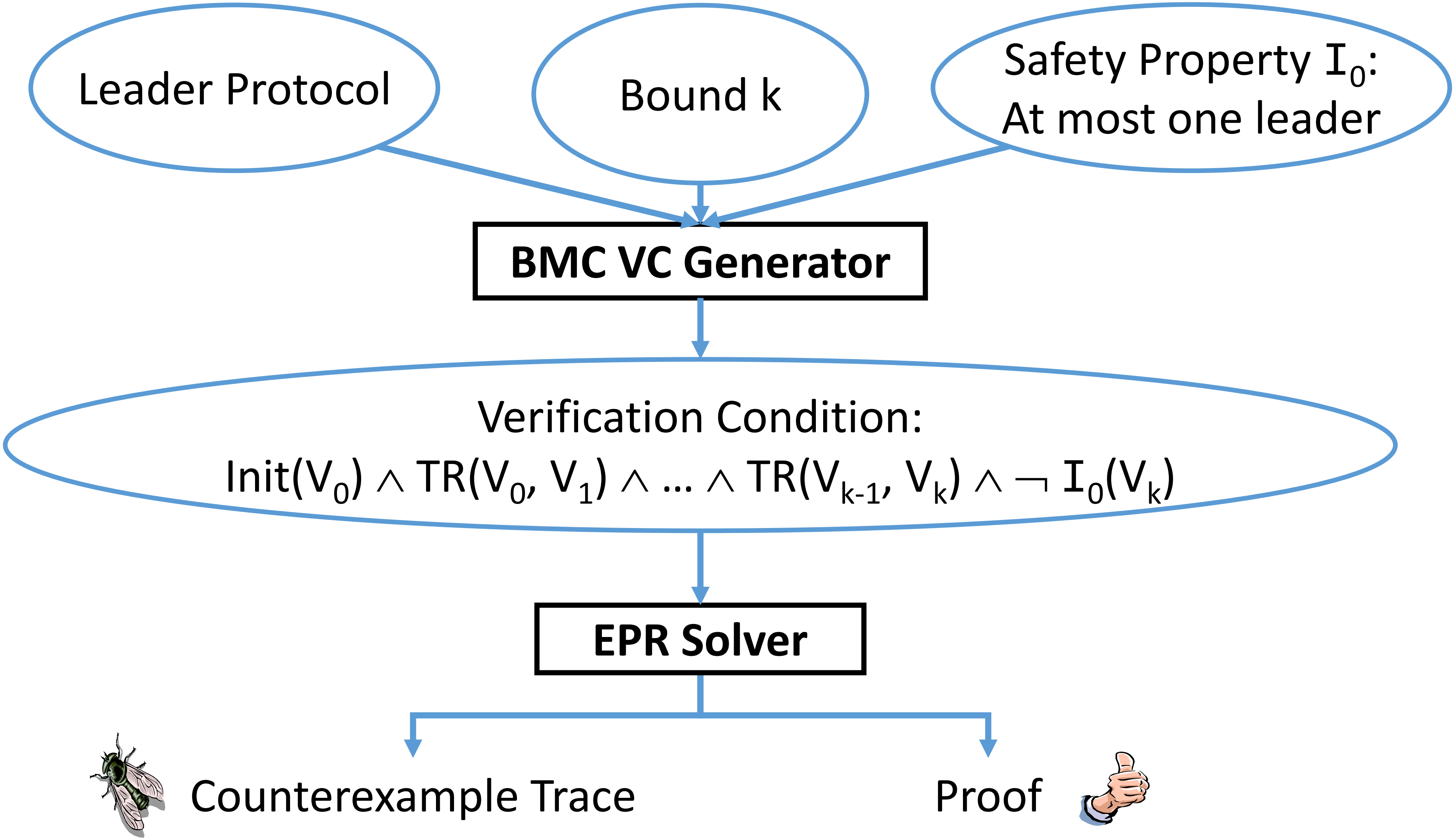
[CACM'79] ... *ralized extrema-fin*

Leader Election Protocol (RML)

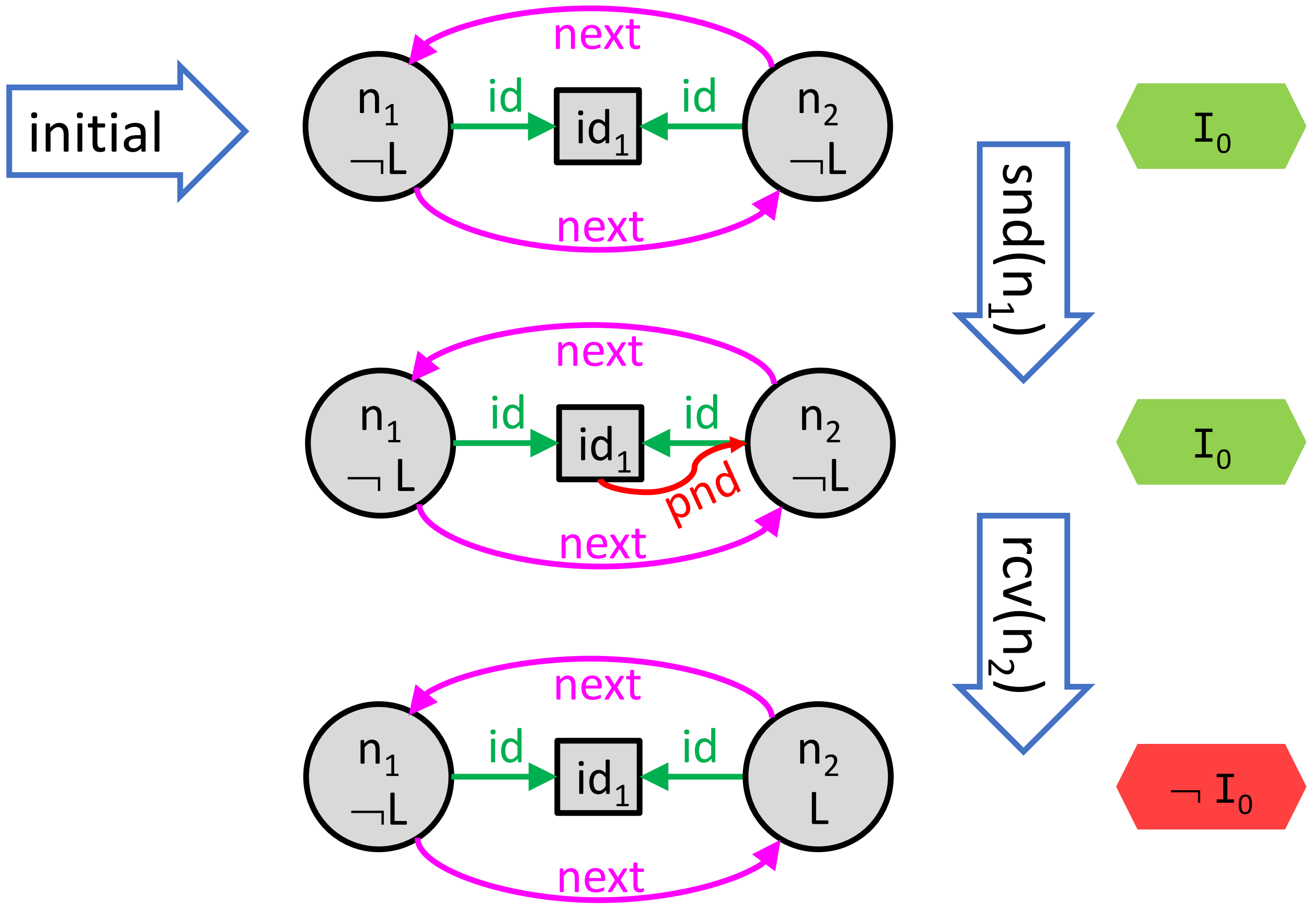
- ⩽ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

```
while true do {
  {
    // send(n1)
    n1 := *;
    n2 := *;
    assume next(n1, n2);
    pending.insert(id[n1], n2)
  }
  ||
  {
    // receive(n1)
    m, n1 := pending.remove();
    if id[n1] = m then:
      // found leader
      leader.insert(n1)
    else if id[n1] ⩽ m then:
      // pass message
      n2 := *;
      assume next(n1, n2);
      pending.insert(m, n2)
  }
}
conjecture I₀ = ¬∃x, y: Node. x ≠ y ∧ leader(x) ∧ id[x] ⩽ id[y]
```

Bounded Model Checking (BMC)



BMC(2)



initial
n1
¬L
id
id1
id
n2
¬L
next
next
I0
snd(n1)
n1
¬L
id
id1
id
n2
¬L
pnd
next
next
I0
rcv(n2)
n1
¬L
id
id1
id
n2
L
next
next
¬ I0

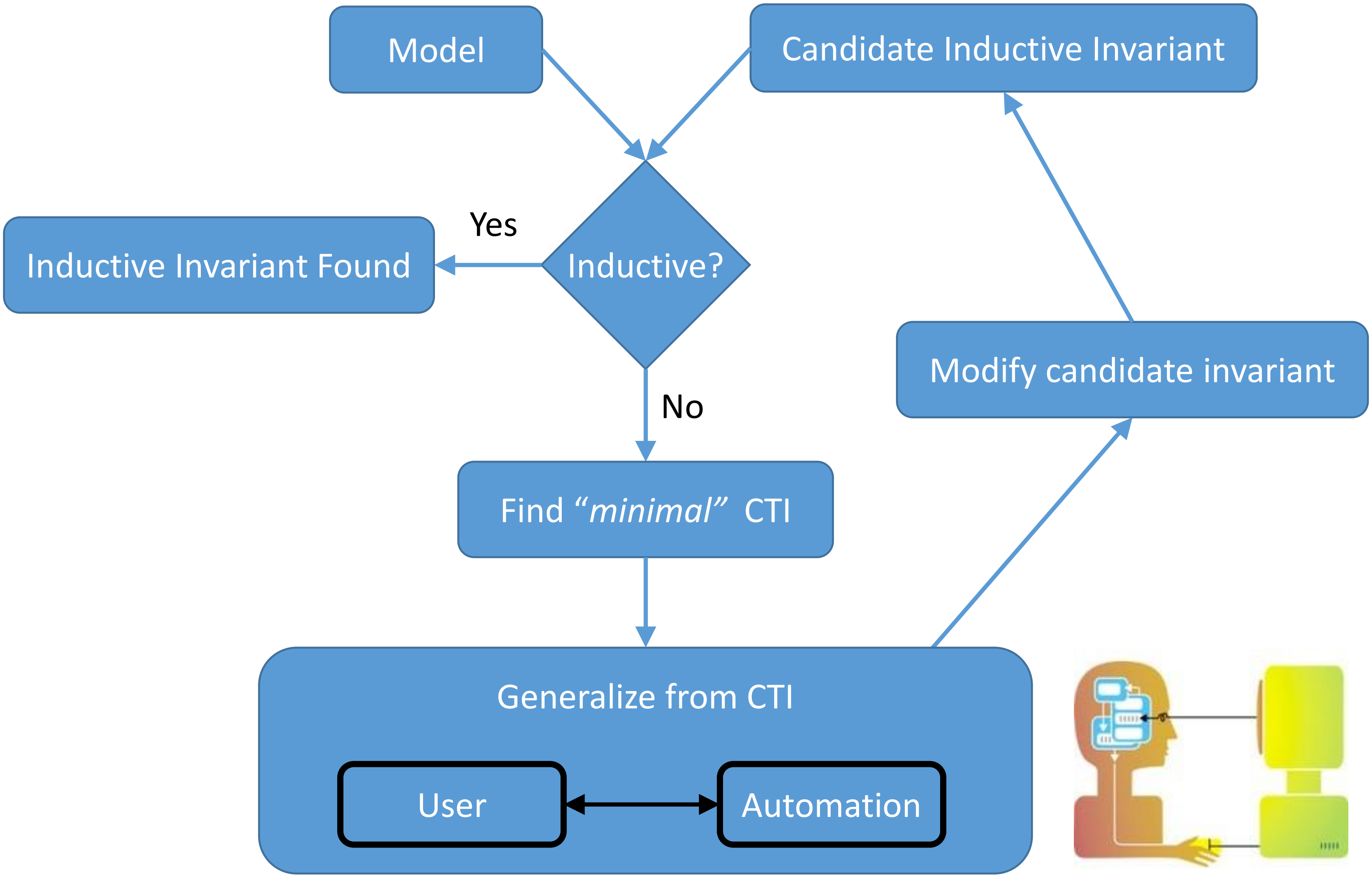
Leader Protocol – 2^{nd} attempt

Axiom $\forall x, y\colon \text{Node}.\ \mathit{id}[x] = \mathit{id}[y] \Rightarrow x = y$

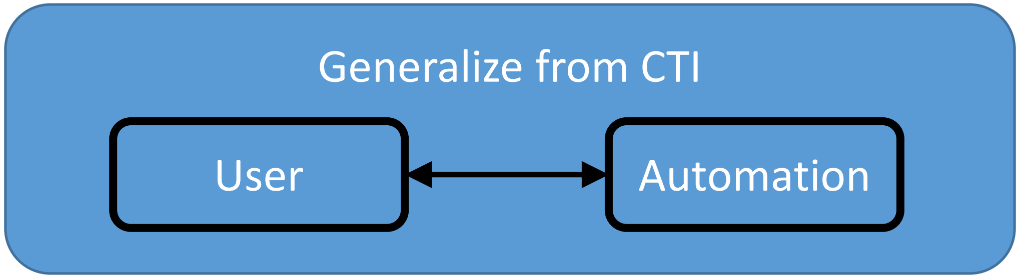
Looks good, let's find an inductive invariant!

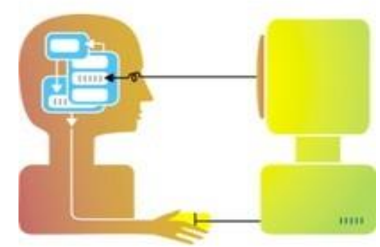
- BMC(1) – OK
- BMC(2) – OK
- BMC(3) – OK
- BMC(4) – OK
- BMC(5) – OK
- BMC(6) – OK
- BMC(7) – OK
- BMC(8) – OK

Invariant Inference in Ivy



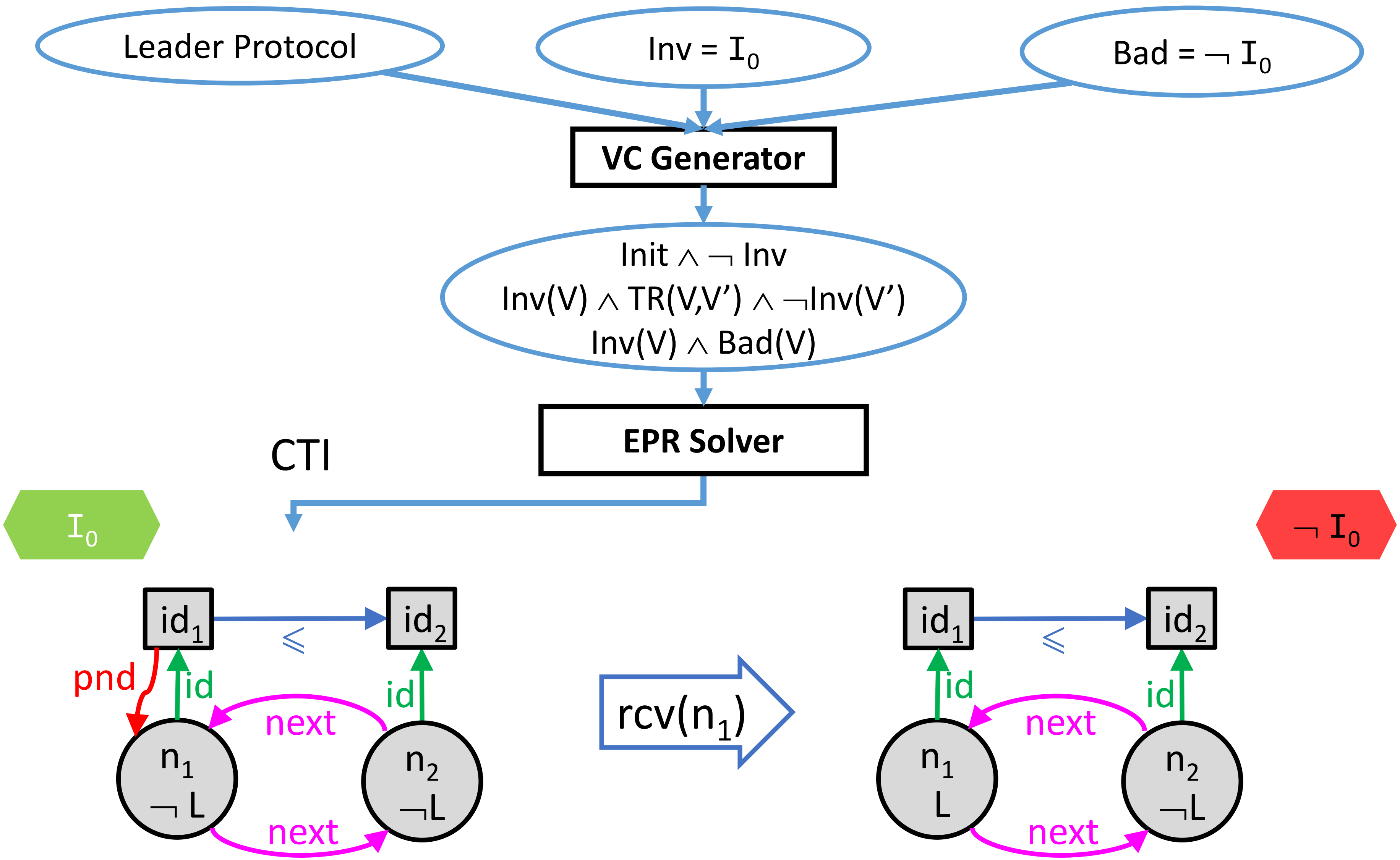
Interactive Generalization from CTI



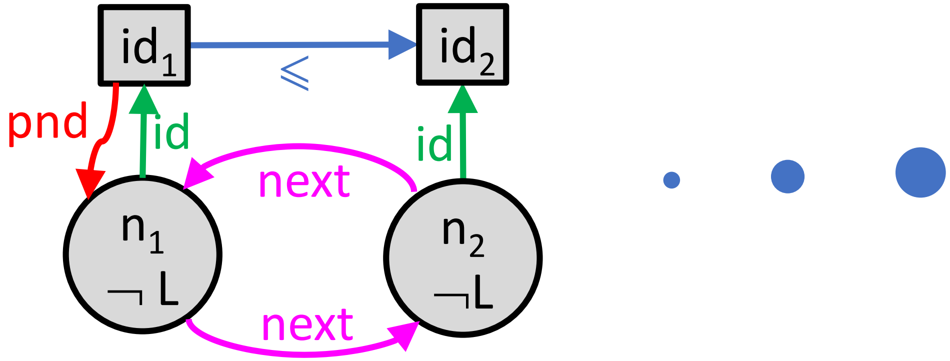


1. Generalize by removing facts to form a conjecture
   - User graphically selects which facts to remove
2. Check if the conjecture is true up to K: BMC(K)
   - User determines the right K to use
   - Ivy uses a SAT solver - sound & complete
3. Automatically remove more facts: Interpolate(K)
   - Ivy uses the SAT solver to discover more facts that can be removed
   - User examines the result – it could be wrong

Algorithmic Deductive Verification (1)



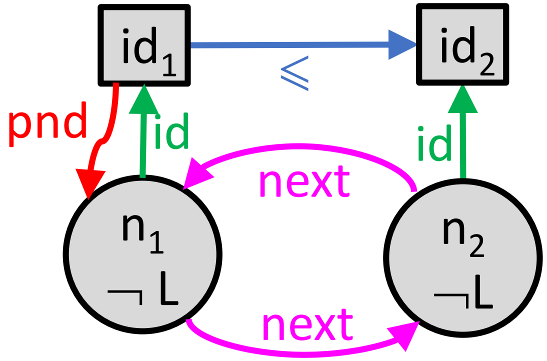
Generalize from CTI (1)



Only the highest id can be self pending

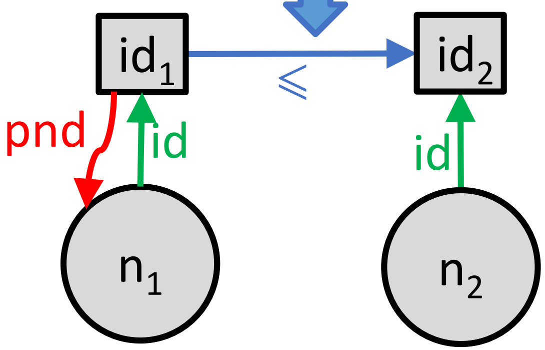
1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node's own id
3. A node that receives its own id becomes the leader

Generalize from CTI (1)



Only the highest id can be self pending

Project to {pnd,≤, id}



BMC(3)

$$C_1 = \neg\exists n_1, n_2\text{: Node. } n_1 \neq n_2 \wedge pnd(id[n_1], n_1) \wedge id[n]_1 \neq id[n_2] \wedge id[n_1] \leqslant id[n_2]$$

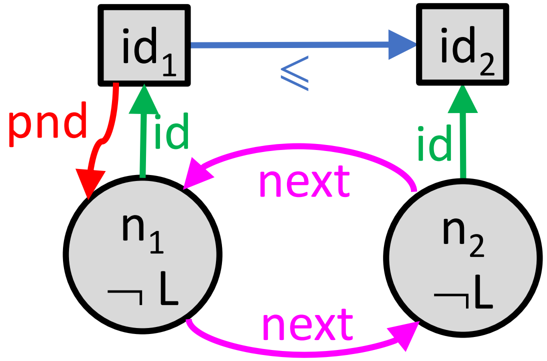
BMC VC Generator (K=3, C_1)

$Init(V_0) \wedge TR(V_0,V_1) \wedge TR(V_1,V_2) \wedge TR(V_1,V_3) \wedge \neg C_1(V_3)$

EPR Solver

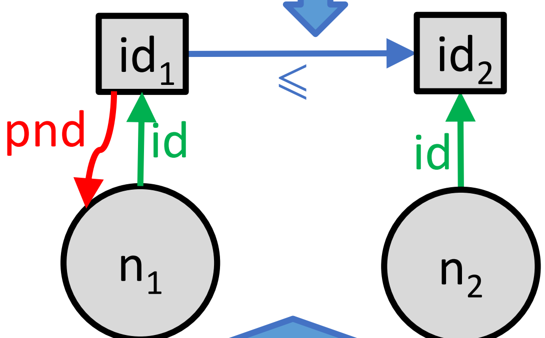
Proof

Generalize from CTI (1)



Only the highest id can be self pending

Project to {pnd,⩽, id}



Interp(3)

$C_1 = \neg\exists n_1, n_2\text{: Node. } n_1 \neq n_2 \wedge pnd(id[n_1], n_1) \wedge id[n]_1 \neq id[n_2] \wedge id[n_1] \leqslant id[n_2]$

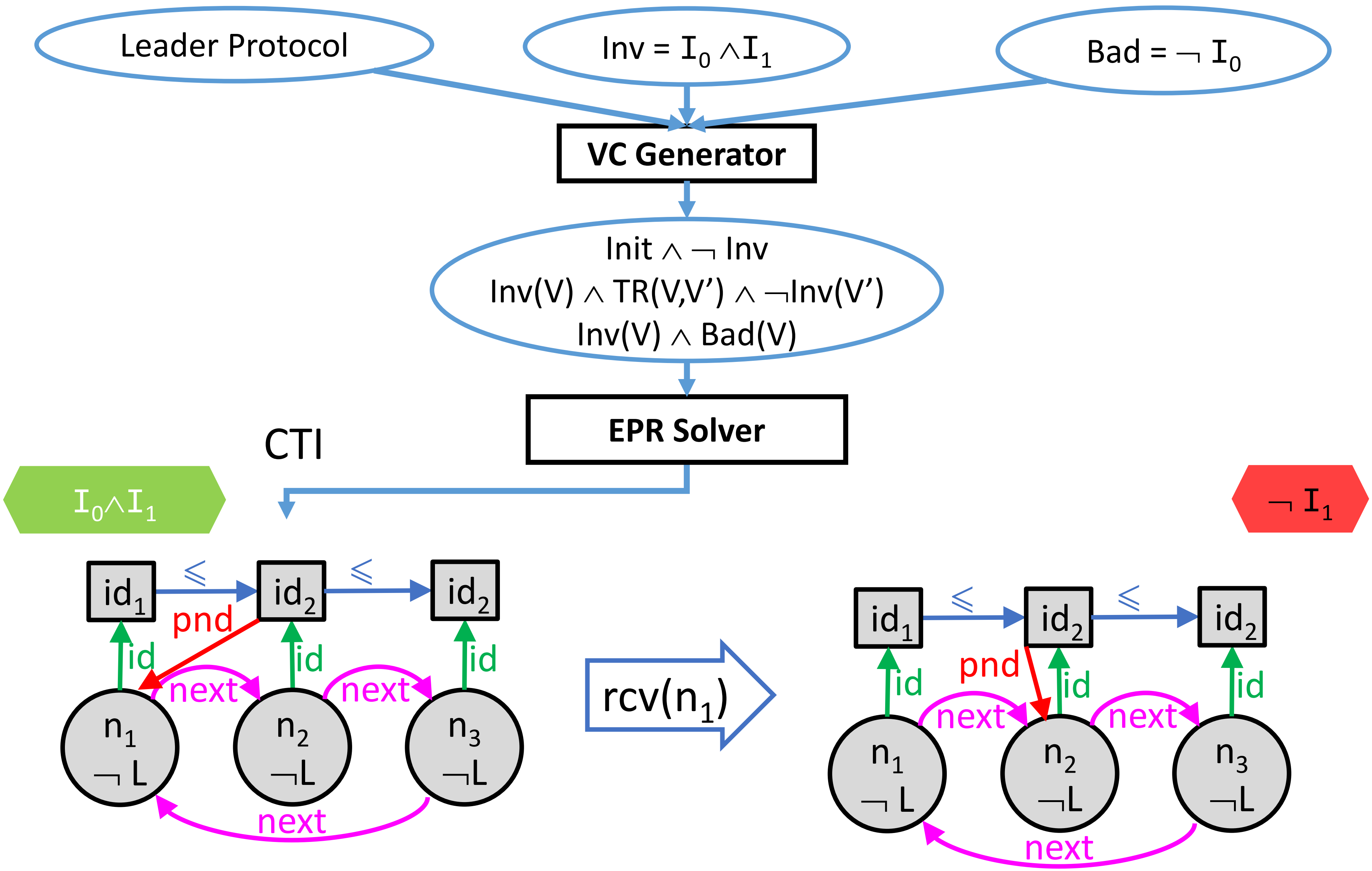
Lookd good, add to the invariant as I_1

$I_1 = \neg\exists n_1, n_2\text{: Node. } pnd(id[n_1], n_1) \wedge id[n]_1 \neq id[n_2] \wedge id[n_1] \leqslant id[n_2]$

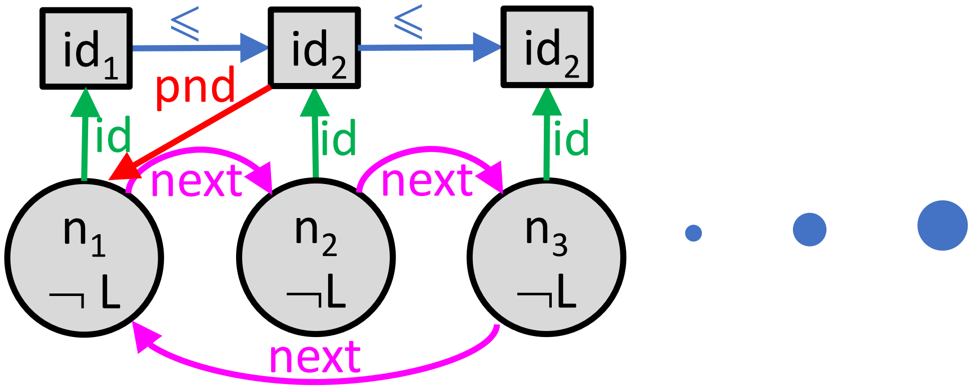
EPR Solver

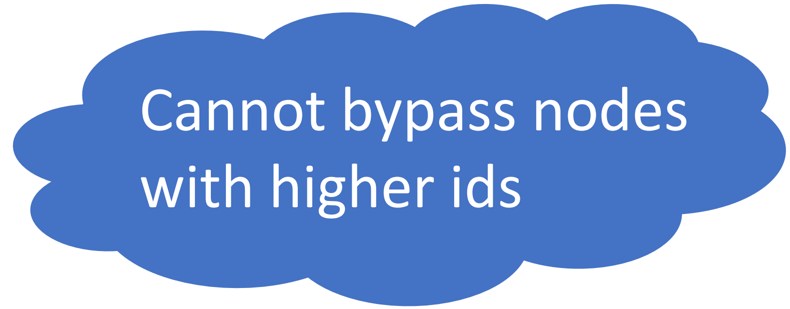
Proof +
Minimal UNSAT core

Algorithmic Deductive Verification (2)



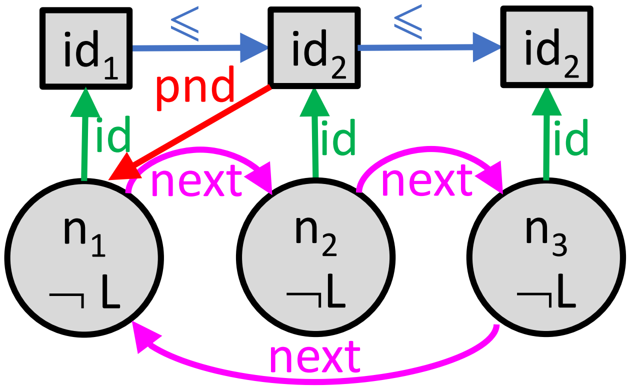
Generalize from CTI (2)





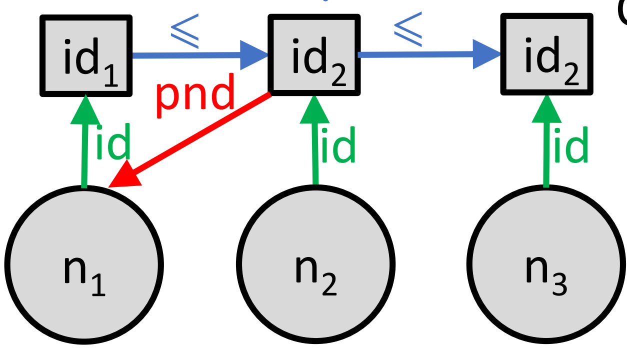
1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node's own id
3. A node that receives its own id becomes the leader

Generalize from CTI (2)



Cannot bypass nodes with higher ids

Project to {pnd,⩽, id}



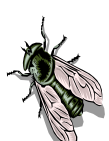
$$C_2 = \neg\exists n_1, n_2, n_3 : \text{Node}.\ \neq(n_1,n_2,n_3) \wedge \neq(id[n_1],id[n_2],id[n_3]) \wedge id[n_1] \leqslant id[n_2] \leqslant id[n_3] \wedge pnd(id[n_2], n_1)$$

BMC(3)

BMC VC Generator (K=3, C_2)

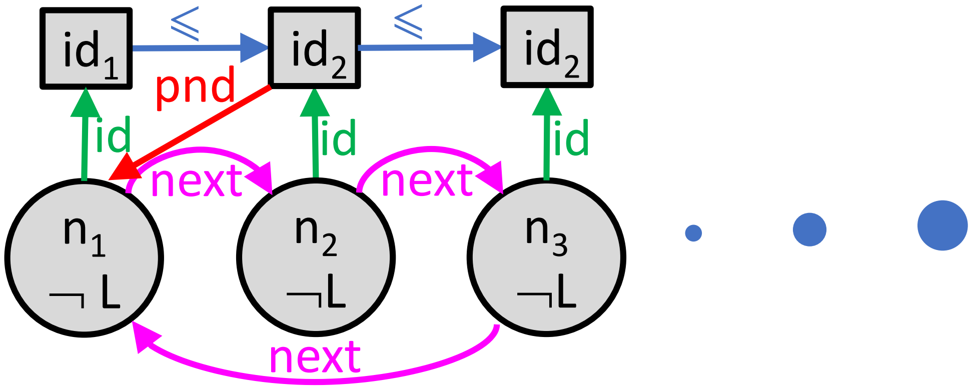
$Init(V_0) \wedge TR(V_0,V_1) \wedge TR(V_1,V_2) \wedge TR(V_1,V_3) \wedge \neg C_2(V_3)$

EPR Solver



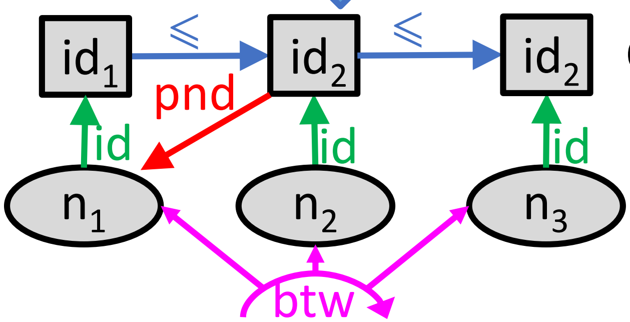
Counterexample Trace

Generalize from CTI (2)



Cannot bypass nodes with higher ids

Project to {pnd, ⩽, id, btw}



$$
\begin{aligned}
C_2 = \neg\exists n_1, n_2, n_3 : \text{Node}.\ & \neq(n_1,n_2,n_3) \land \\
& \neq(id[n_1],id[n_2],id[n_3]) \land \\
& id[n_1] \leqslant id[n_2] \leqslant id[n_3] \land \\
& pnd(id[n_2], n_1) \land btw(n_1, n_2, n_3)
\end{aligned}
$$

BMC(3)

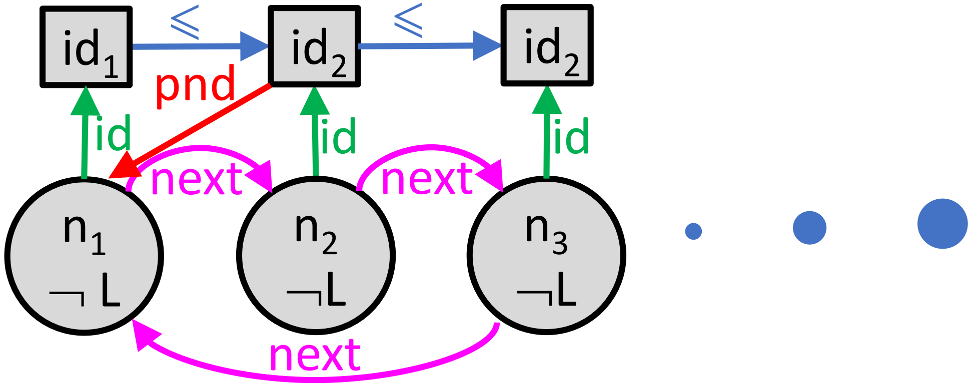
BMC VC Generator (K=3, C_2)

$Init(V_0) \land TR(V_0,V_1) \land TR(V_1,V_2) \land TR(V_1,V_3) \land \neg C_2(V_3)$

EPR Solver

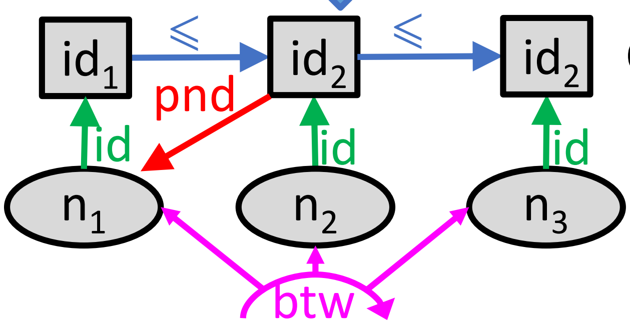
Proof 

Generalize from CTI (2)



Cannot bypass nodes with higher ids

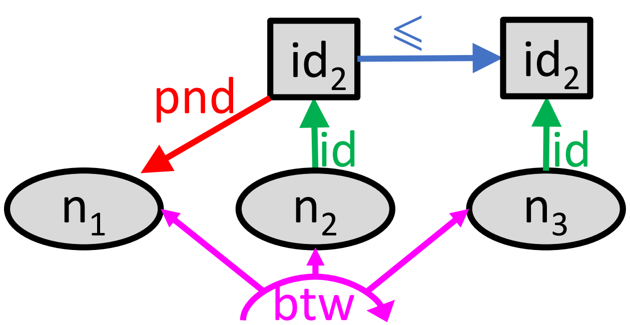
Project to {pnd,⩽, id,btw}



$$C_2 = \neg\exists n_1, n_2, n_3 : \text{Node}.\ {\neq}(n_1,n_2,n_3) \wedge {\neq}(id[n_1],id[n_2],id[n_3]) \wedge id[n_1] \leqslant id[n_2] \leqslant id[n_3] \wedge pnd(id[n_2], n_1) \wedge btw(n_1, n_2, n_3)$$

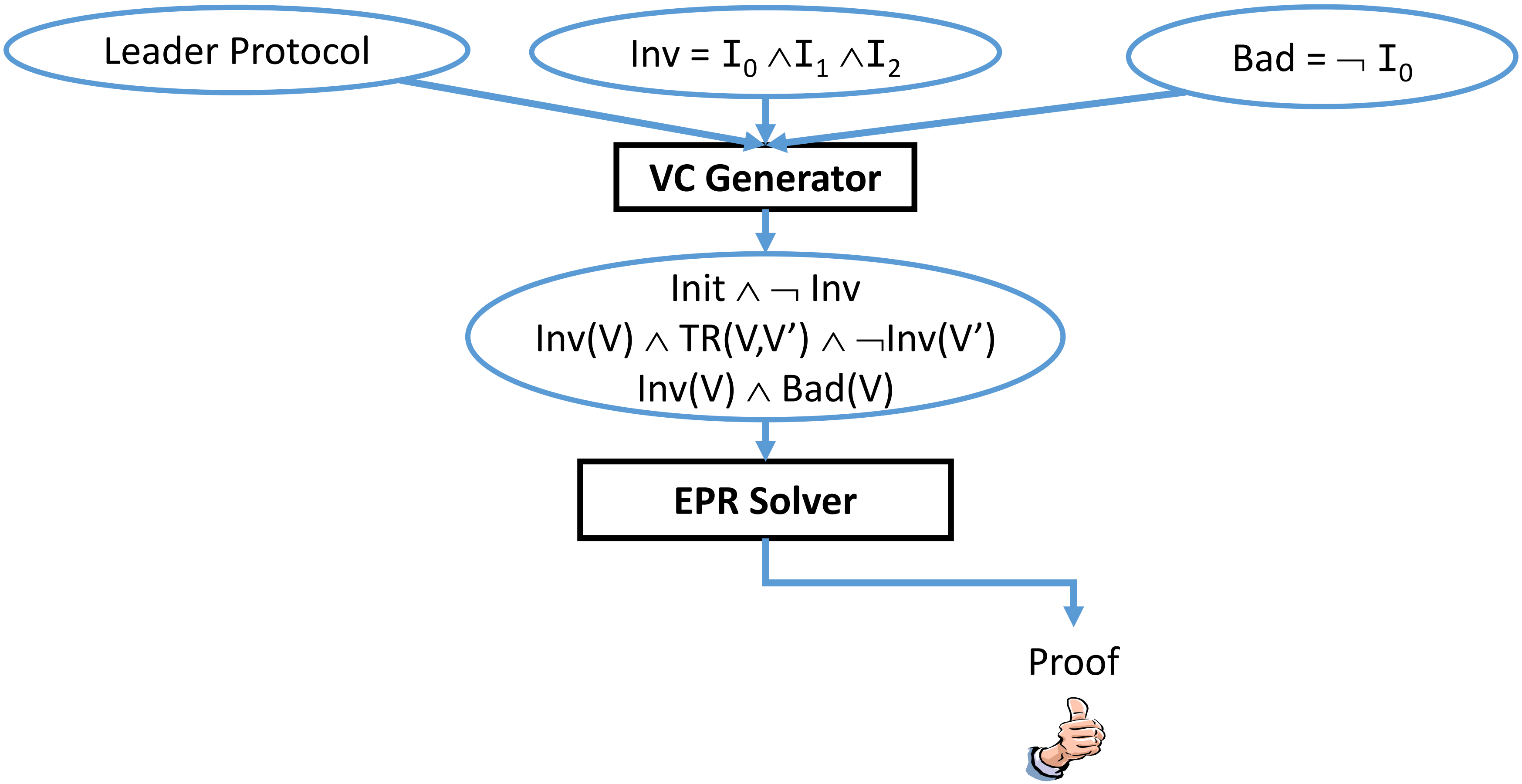
Interp(3)

This looks good, add to the invariant as I_2



$$I_2 = \neg\exists n_1, n_2, n_3 : \text{Node}.\ btw(n_1, n_2, n_3) \wedge pnd(id[n_2], n_1) \wedge id[n_2] \leqslant id[n_3]$$

Algorithmic Deductive Verification (3)



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

Completeness and Interaction Complexity

- Any generalization from CTI adds one universal clause to the invariant
- The invariant is constructed in CNF
- If there is a universal invariant with N clauses, it can be obtained by the user in N generalization steps
  - Assuming the user is optimal
- If the user is sub-optimal, backtracking (weakening) may be needed

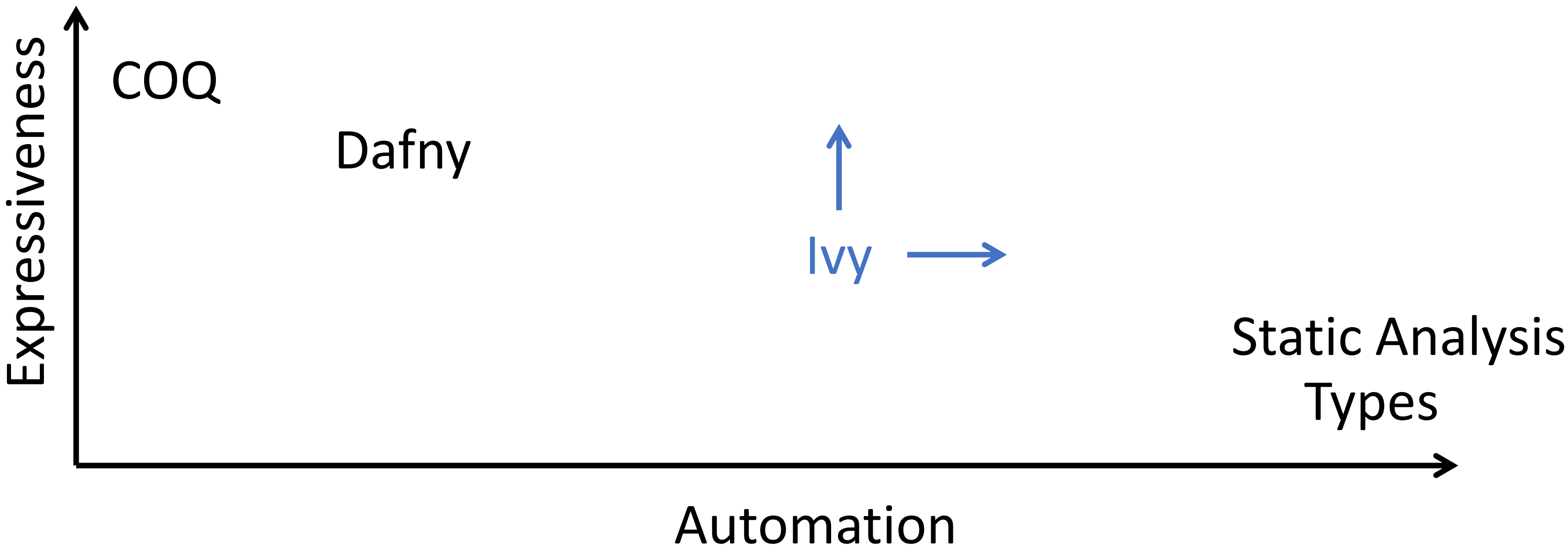
Verified Protocols

Protocol	Model Types	Relations & Functions	Property (# Literals)	Invariant (# Literals)	CTI Gen. Steps
Leader in Ring	2	5	3	12	3
Learning Switch	2	5	11	18	3
DB Chain Replication	4	13	11	35	7
Chord	1	13	35	46	4
Lock Server 500 Coq lines [Verdi]	5	11	3	21	8 (1h)
Distributed Lock 1 week [IronFleet]	2	5	3	26	12 (1h)
Paxos	Work in progress				
Raft					

Ivy Summary & Lessons Learned

- Ivy:
  - RML – modeling language that makes deduction decidable
  - Interactive generalization for finding inducting invariants
  - Application to the domain of distributed protocols
- User intuition and machine heuristics complement each other:
  - User has the ability to ignore irrelevant facts and intuition that leads to *better generalizations*
  - Machine is better at finding bugs and corner cases
- The safety of many protocols can be proven w/o reasoning about arithmetic operations or set cardinalities
  - Many important properties can be captured in a (parametric) model that abstracts the actual numerical values
  - Unbounded topologies
  - Unique paths (ring, trees, …)

Current & Future Work



- More expressiveness, keeping decidability
  - System, Spec, Invariant (proof)
- Verifying more systems (Paxos and Raft are next)
- Inferring inductive invariants (more) automatically
- Better theoretical understanding of limitations and tradeoffs