Tentative Schedule

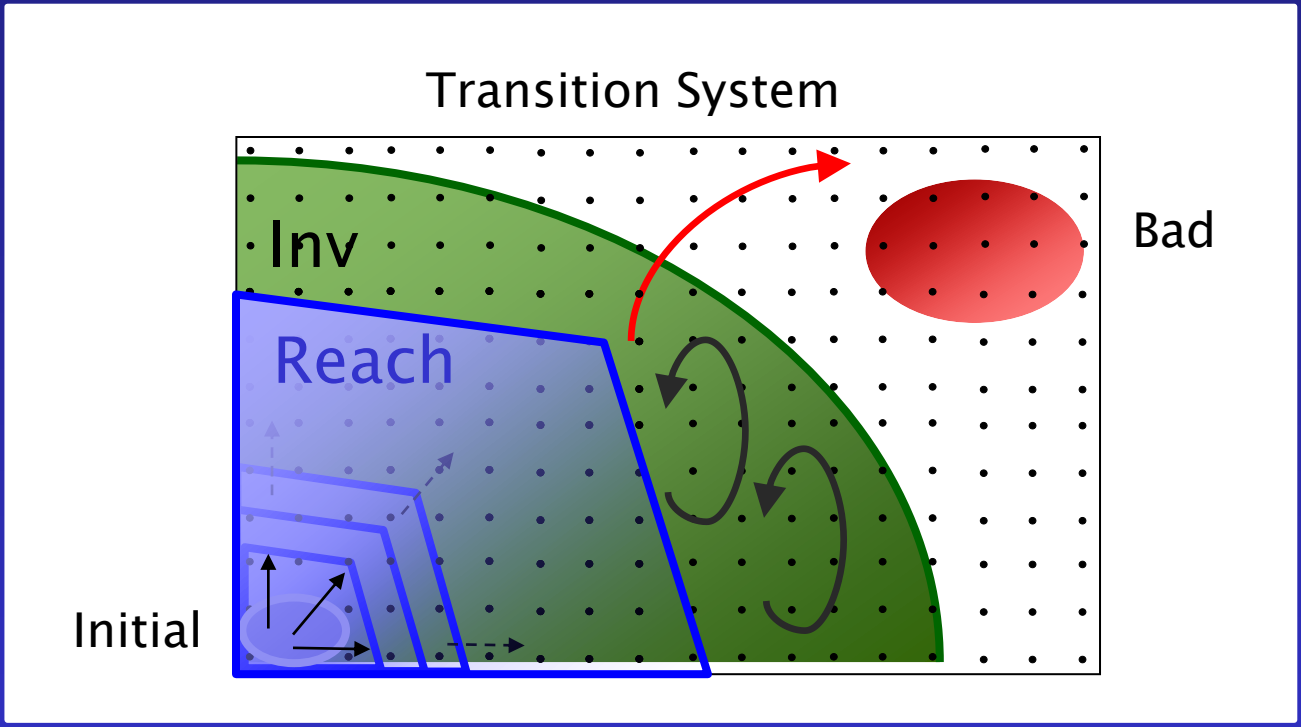
Date	Subject	Presenter
28/2	Introduction	Mooly
6/3	IVY	Mooly
13/3	Coq Tutorial	Oded Padon
20/3	Network Verification	Mooly
27/3	Comprehensive Formal Verification of an OS Microkernel	Elazar Gershuni
3/4	Ironclad	Vadim Stotland
10/4	Formal verification of a realistic compiler:	Omer Anson
1/5	A Design and Verification Methodology for Secure Isolated Regions	Yotam Feldman

http://apanda.github.io/ivy/

IVY: Interactive Safety Verification via Counterexample Generalization

Oded Padon **Ken McMillan** Aurojit Panda Sharon Shoham

Inductive Invariants



System S is **safe** if all the reachable states satisfy the property

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

Inv $\Rightarrow$ $\varphi = \neg$Bad (Safety)
Init $\Rightarrow$ Inv (Initiation)
if $\sigma \vDash$ Inv and $T(\sigma, \sigma')$ then $\sigma' \vDash$ Inv (Consecution)

Deductive Verification

Program
P

Candidate Invariant I

Safety
Property φ

Solver

Is there a behavior
of P that violates the inductiveness of I?

Counterexample to induction
(CTI)

Proof

Deductive Verification

```
1: x := 1;
2: y := 2;
while * do {
   3: assert x ≥1;
   4:  x:= x + y;
   5:  y := y + 1
}
6:
```

$at(3) \Rightarrow x \geq 1$

$at(3) \Rightarrow x \geq 1$

Solver

Is there a behavior of P that violates the inductiveness of I?

3: <1, -2>

x := x+y; y :=y+1

$\neg (at(3) \Rightarrow x \geq 1)$

Deductive Verification

```
1: x := 1;
2: y := 2;
while * do {
   3: assert x ≥1;
   4:  x:= x + y;
   5:  y := y + 1
}
6:
```

$at(3) \Rightarrow x \geq 1 \wedge y \geq 0$

$at(3) \Rightarrow x \geq 1$

Solver

Is there a behavior of P that violates the inductiveness of I?

Proof

IronFleet: Proving Practical Distributed Systems Correct [SOSP'15]

- Employs deductive verification
- Useful for verifying real systems

- Writing invariants is hard
- Deduction is hard
  - Quantifier alternations leads to matching loops
  - Complicated arithmetic

Challenges

1. Specifying safety properties
2. Inductive Invariants for Deductive Verification
   - Hard to express
   - Hard to change
   - Hard to infer
3. Deduction
   - Reasoning about inductive invariants
     - Undecidability of implication checking

Motivation

- Automatic safety verification is limited
  - The safety problem is undecidable
- Permit high-degree of automation while maintaining diaganosablity
  - The system should fail visibly
  - The user should never get stuck
    - Engage the user in verification

IVY's philosophy

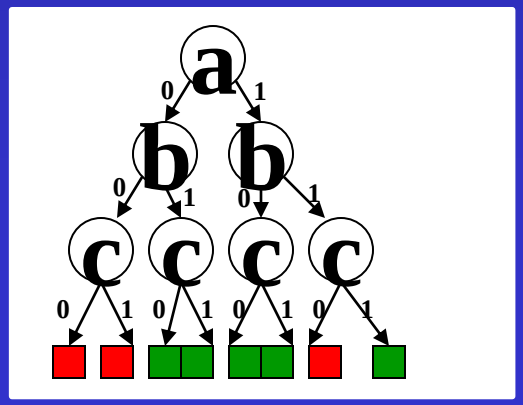
- Deduction is hard
  - Use a restricted logic - EPR
  - ☺ Sound and complete deduction
  - ☺ Turing complete modeling language
  - ☹ Limited safety and inductive invariants
    - ➢ Suffices for many distributed protocols
- Invariant inference is hard
  - Combine automated techniques with user interaction
  - Provide graphical UI for gradually strengthening the inductive invariant

Expressiveness vs. Automation

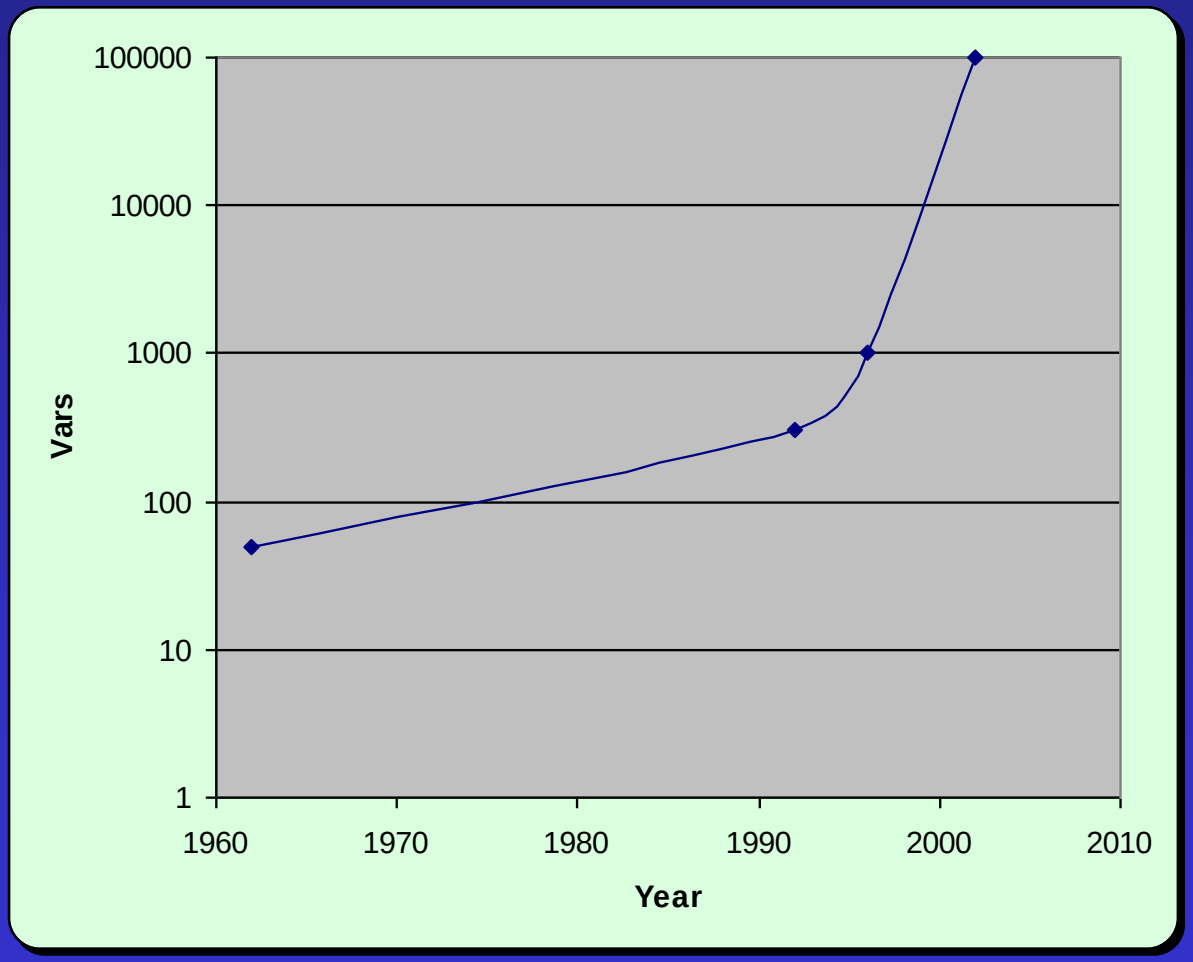
	Coq	Dafny	IVY	Static Analysis
Invariant	User	User	User + System	System
Deduction	User	System (Z3) + “User”	System (EPR)	System

The SAT Problem

- Given a propositional formula (Boolean function)
  - $\varphi = (a \vee b) \wedge (\neg a \vee \neg b \vee c)$
- Determine if φ is valid
- Determine if φ is satisfiable
  - Find a satisfying assignment or report that such does not exit
- For *n* variables, there are 2^n possible truth assignments to be checked



SAT made some progress…



Effectively Propositional Logic – EPR

a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic
  - Restricted quantifier prefix: $\exists^* \forall^* \varphi_{Q.F.}$
  - No $\forall^* \exists^*$
    - No recursive function symbols
    - No arithmetic
- Small model property
  - A formula is satisfiable iff it is holds on models proportional to the number of existential variables

EPR SAT

$$\exists x_1, x_2 . \forall y . r(x_1, y) \leftrightarrow r(y, x_2)$$

$$=_{sat} \forall y . \ r(c_1, y) \leftrightarrow r(y, \ c_2)$$

$$=_{sat} (r(c_1, c_1) \leftrightarrow r(c_1, c_2)) \wedge$$
$$(r(c_1, c_2) \leftrightarrow r(c_2, c_2))$$

$$=_{sat} (P_{11} \leftrightarrow P_{12}) \wedge (P_{12} \leftrightarrow P_{22})$$

Effectively Propositional Logic – EPR

a.k.a. Bernays-Schönfinkel-Ramsey class

- Limited fragment of first-order logic
  - Restricted quantifier prefix: $\exists^*\forall^*\ \varphi_{Q.F.}$
  - No $\forall^*\ \exists^*$
    - No recursive function symbols
    - No arithmetic
- Small model property
  - $\exists x_1,\ldots, x_n.\forall\ y_1,\ldots,y_m.\varphi_{Q.F.}$ has a model iff it has a model of at most n+k elements (k - number of constant symbols)
- Satisfiability is decidable
  - NEXPTIME complete
- Support from Z3, Iprover, Vampire

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

Algorithmic Deductive Verification

Program P
$\exists^* \forall^*$

Candidate Inductive Invariant I
$\forall^*$

Property φ
$\forall^*\exists^*$

VC gen

Verification Conditions:

1) $\text{Init} \wedge \neg I$

2) $I(V) \wedge \neg \varphi(V)$

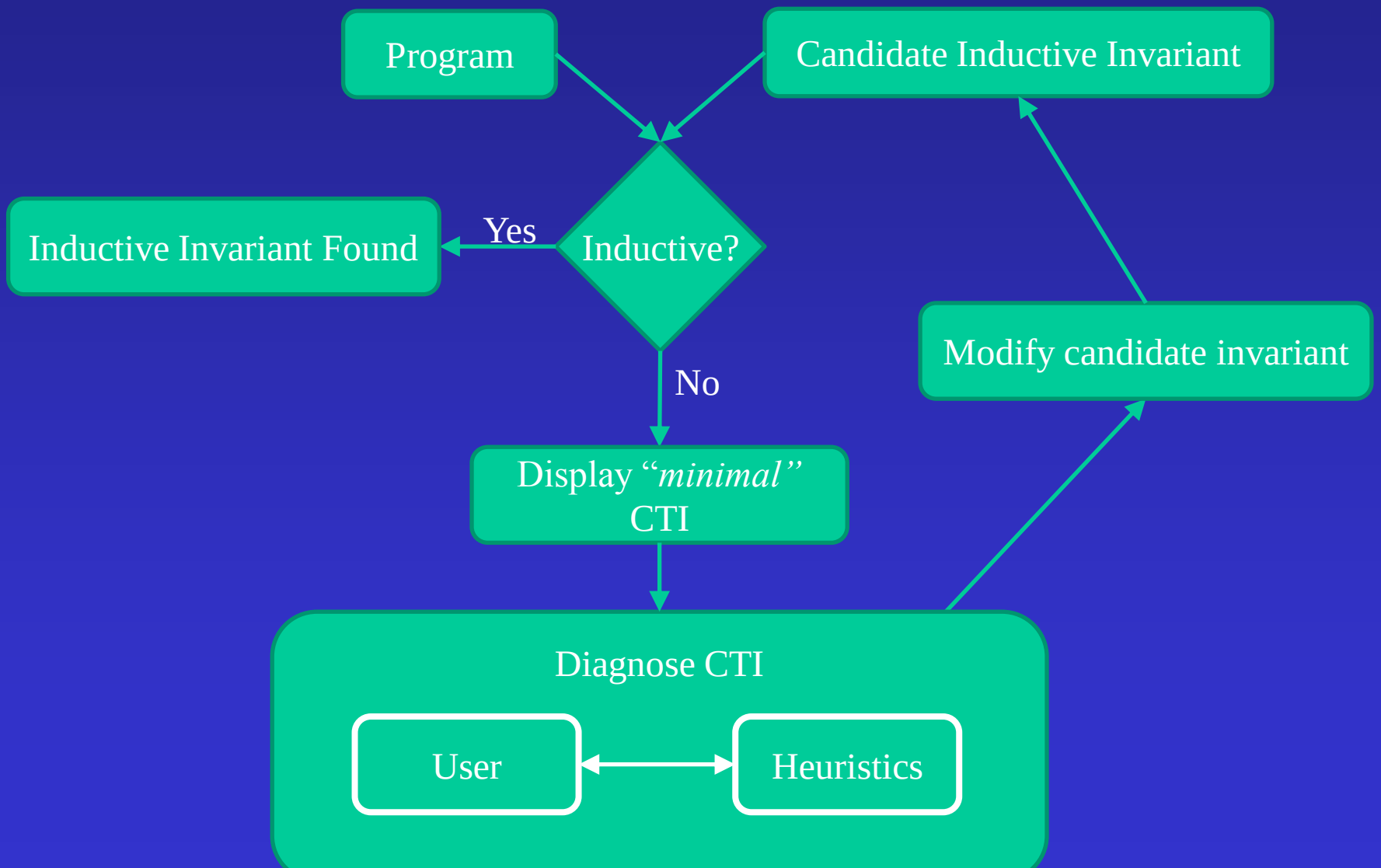
3) $[\![P]\!](V, V') \wedge I(V) \wedge \neg I(V')$

SAT Solver

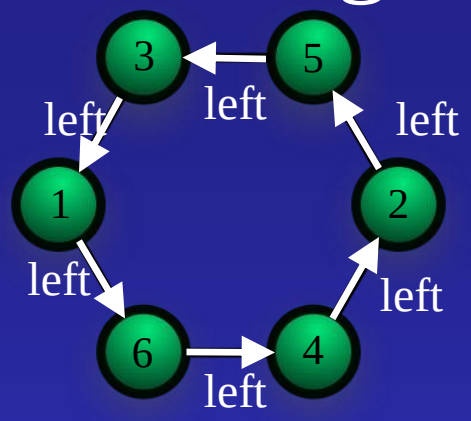
CTI

Proof

CTI based Generalization



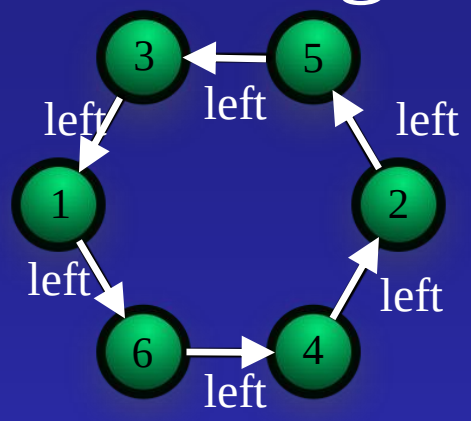
Example: Leader Election in a Ring



- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:
  - Each node sends its id to the left
  - A node that receives a massage passes it (to the left) 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 left
  - A node that receives a massage passes it (to the left) if the id in the message is higher than the node's own id
  -
- 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' ecentralized *extrema-*

Leader Protocol

```
action send =
  n1 := *
  n2 := *
  assume left(n1, n2)
  pending.insert(id[n1], n2)
```

```
action receive =
  m, n1 := pending.remove()
  if id[n1] = m then
    // found leader
    leader.insert(n1)
  else if id[n1]⩽m then
    // pass message
    n2 := *
    assume left(n1, n2)
    pending.insert(m, n2)
```

- ⩽ (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

Sound and Complete Domain Axioms

- Total orders on IDs
- Relation: $id_1 \leqslant id_2$
  - Reflexive, transitive, anti-symmetric, total
- Relation: $btw(n_1\text{: Node}, n_2\text{: Node}, n_3\text{: Node})$
  // there is an acyclic path from n_1 to n_3 via n_2
  - Similar axioms
  - $left(a, b) \equiv a \neq b \wedge \forall x\text{: Node}.\ btw(a, x, b) \Rightarrow (x=a \vee x=b)$

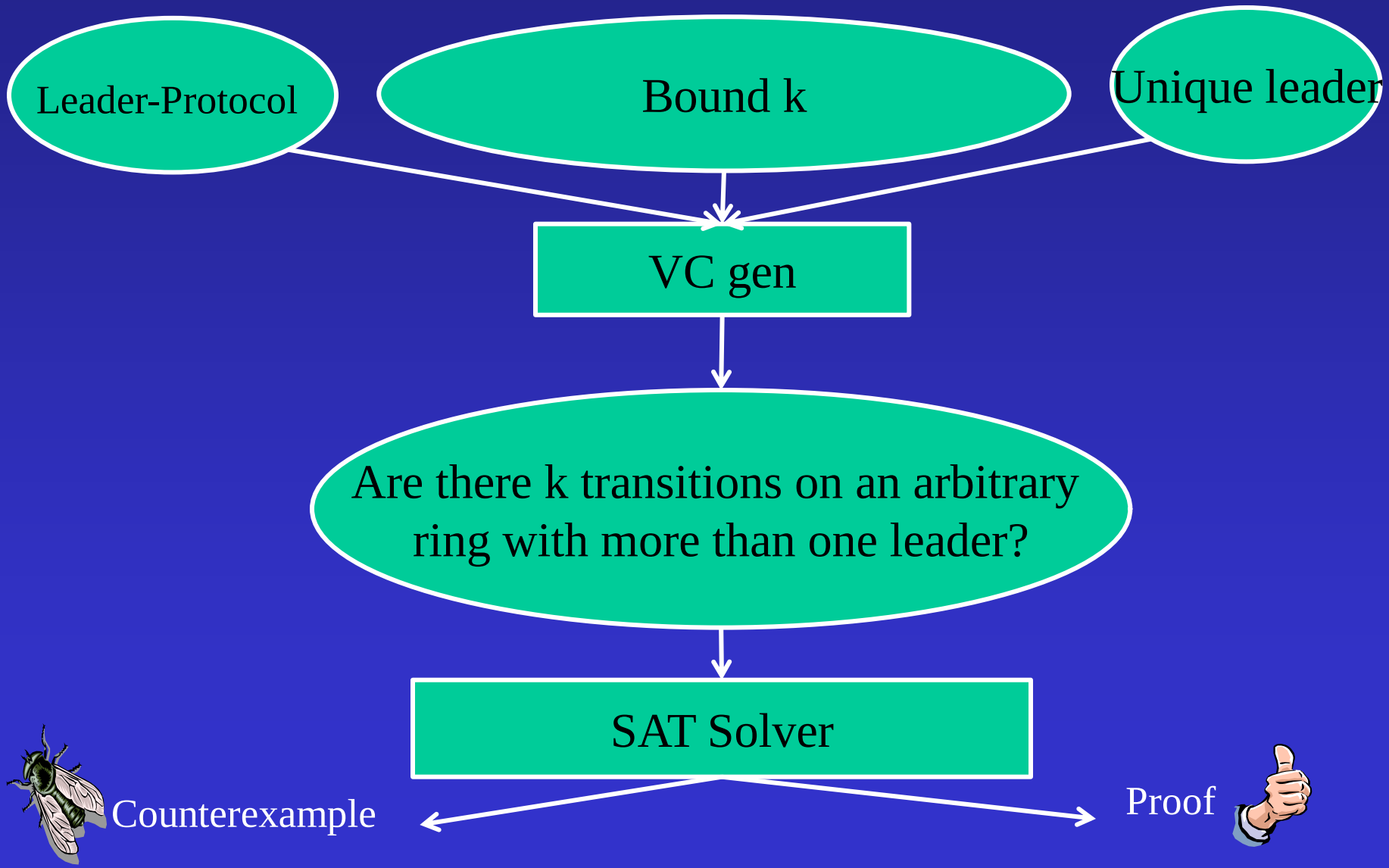
Sound and Complete Order Axioms

- Relation $\leqslant$ (ID, ID)
- $\forall$x: ID. x $\leqslant$ x // Reflexive
- $\forall$x, y, z: ID. x $\leqslant$ y) $\wedge$y $\leqslant$ z $\Rightarrow$x $\leqslant$ z // Transitive
- $\forall$x, y: ID. x $\leqslant$ y $\wedge$y $\leqslant$ x $\Rightarrow$ x = y // Anti-symmetric
- $\forall$x, y: ID. x $\leqslant$ y $\vee$y $\leqslant$ x // Total
- succ(a, b) $\equiv$ le(a, b) $\wedge$a$\neq$b $\wedge$
  ($\forall$x: ID. a $\leqslant$ x $\Rightarrow$ x =a $\vee$b $\leqslant$ x)

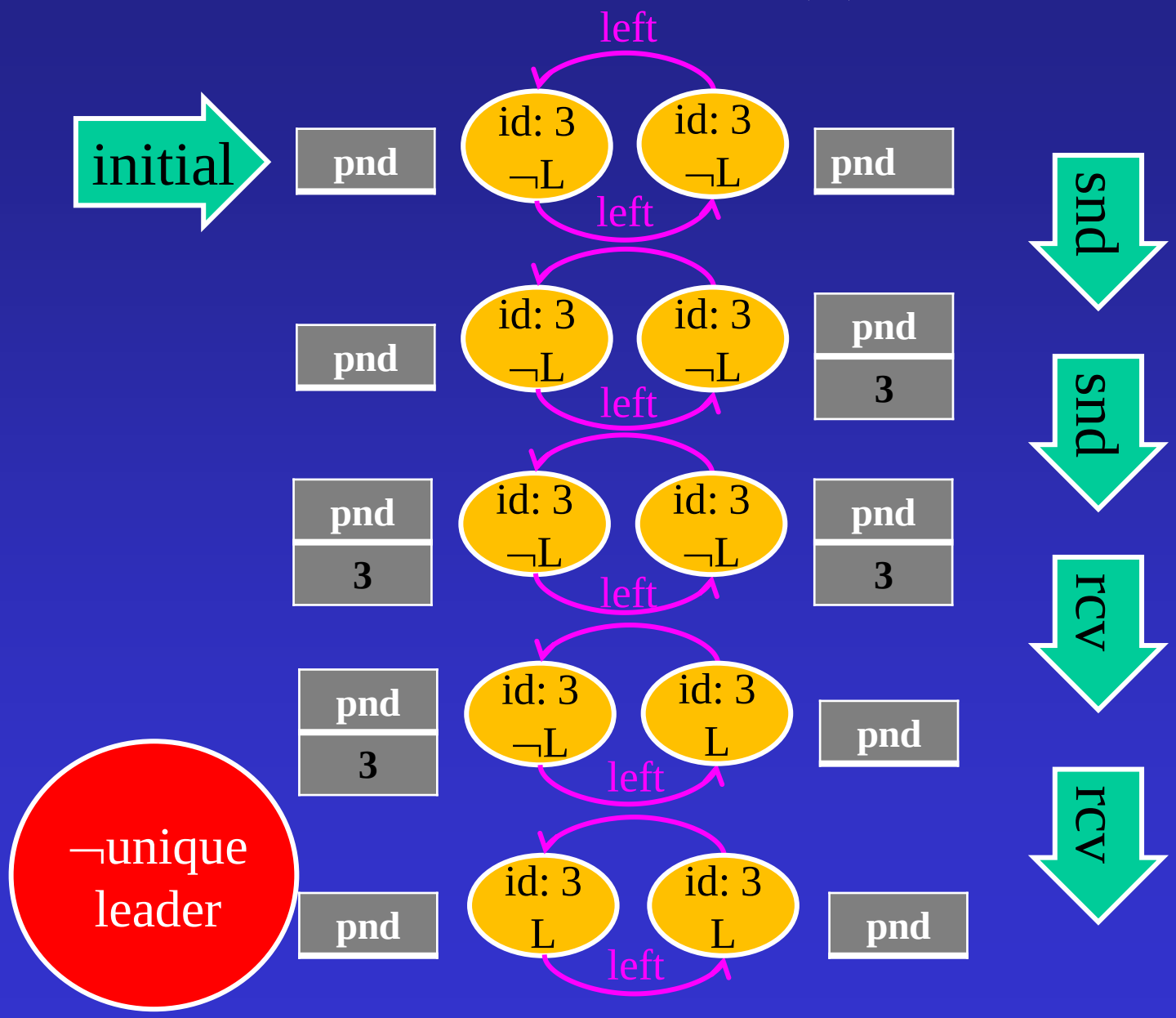
Sound and Complete Ring Axioms

- Relation btw(n_1: Node, n_2: Node, n_3: Node)
  // there is an acyclic path from n_1 to n_3 via n_2
- $\forall$x, w: Node. btw(w, x, x) // Reflexive
- $\forall$x, y, z, w: Node. btw(w, x, y) $\wedge$btw(w, y, z) $\Rightarrow$btw(w, x, z)
  // Transitive
- $\forall$x, y, w: Node. btw(w, x, y) $\wedge$btw(w, y, x) $\Rightarrow$ x = y
  // Anti-symmetric
- $\forall$x, y, w: Node. btw(w, x, y) $\vee$btw(w, y, x) // Totality
- $\forall$x, y, z: Node. x $\neq$y $\wedge$btw(x, y, z) $\Rightarrow$btw(y, z, x)
  // Cyclic permutation
- left(a, b) $\equiv$ a $\neq$b $\wedge$ $\forall$x: Node. btw(a, x, b) $\Rightarrow$(x=a $\vee$x=b)

Bounded Model Checking(BMC)



BMC(4)


initial
pnd
id: 3
¬L
id: 3
¬L
pnd
left
left
snd
pnd
id: 3
¬L
id: 3
¬L
pnd
3
left
snd
pnd
3
id: 3
¬L
id: 3
¬L
pnd
3
left
rcv
pnd
3
id: 3
¬L
id: 3
L
pnd
left
rcv
pnd
id: 3
L
id: 3
L
pnd
left
¬unique
leader

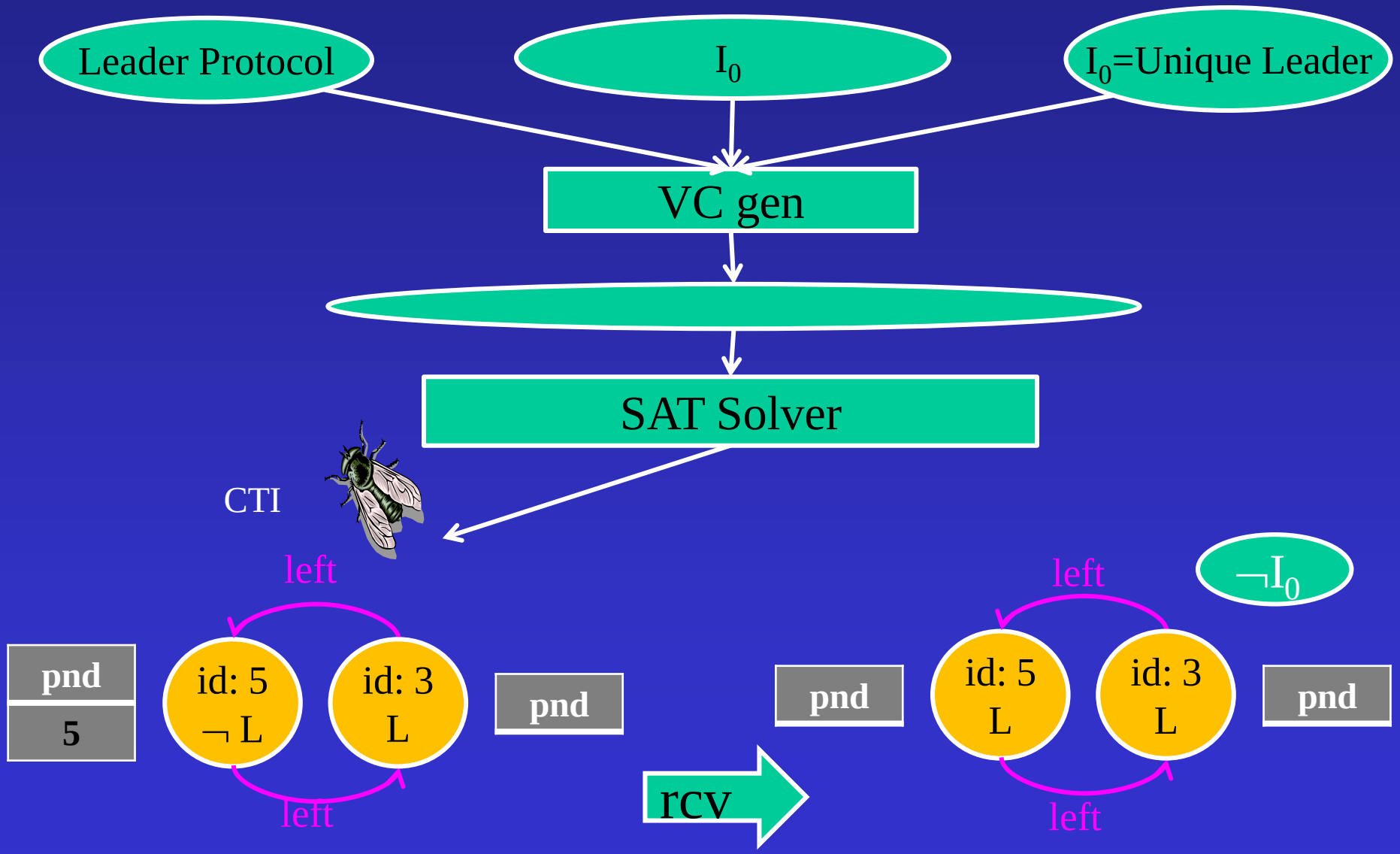
Leader Protocol – 2nd attempt

Axiom $\forall x, y$: Node. $id[x] = id[y] \Rightarrow x = y$

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

Algorithmic Deductive Verification(1)

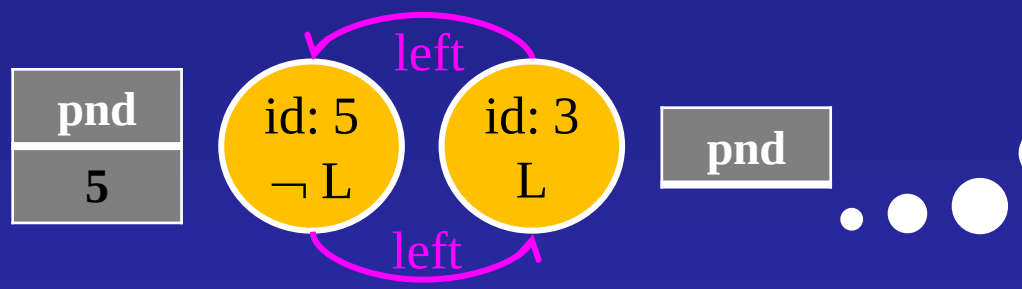


Generalizing CTIs

EPR SAT calls

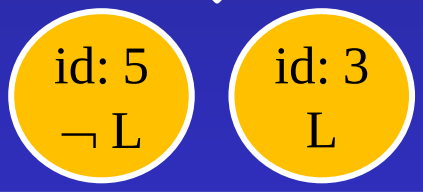
- IVY provides
  - Projection of a state on sub-relations
  - Convert a projection of a state to a conjecture
  - BMC(k)
    - Check if a conjecture holds k steps from init
  - Interpolate(k)
    - Strengthen a conjecture while keeping it true k steps from init

Generalizing CTIs(1)



The leader should have the highest id

Project to {L,⩽, id}



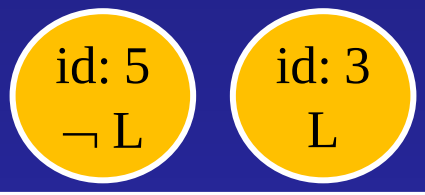
Convert projection to conjecture

$\forall n_1, n_2$: Node. $\neg(n_1 \neq n_2 \wedge \neg L(n_1) \wedge L(n_2) \wedge id[n_1] > id[n_2])$

BMC(2)

OK

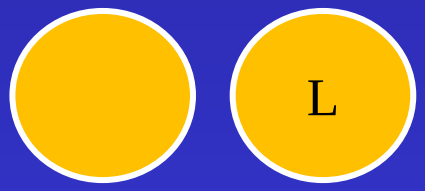
Generalizing CTIs(1)



Convert projection to conjecture

$\forall n_1, n_2$: Node. $\neg(n_1 \neq n_2 \wedge \neg L(n_1) \wedge L(n_2) \wedge id[n_1] > id[n_2])$

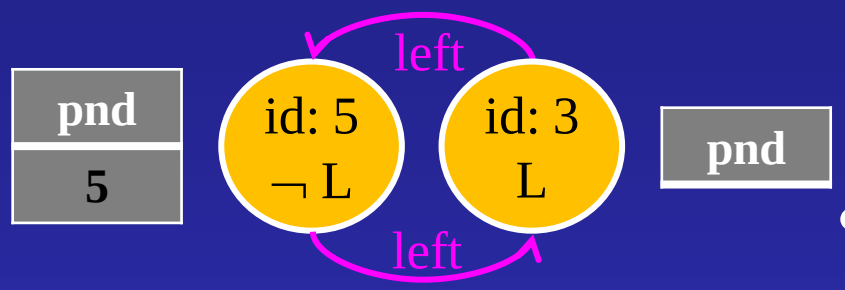
Interp(2)



$\forall n_1, n_2$: Node. $\neg(n_1 \neq n_2 \wedge L(n_2))$

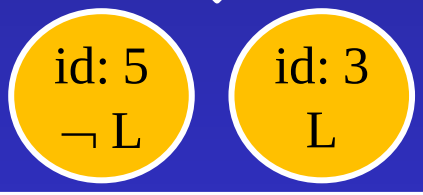
This doesn't seem right….
The bound 2 is too low

Generalizing CTIs(1)



The leader should have the highest id

Project to {L,⩽, id}



Convert projection to conjecture

$\forall n_1, n_2$: Node. $\neg(n_1 \neq n_2 \wedge \neg L(n_1) \wedge L(n_2) \wedge id[n_1] > id[n_2])$

BMC(3)

OK

Generalizing CTIs(1)

id: 5
$\neg$ L

id: 3
L

Convert projection to conjecture

$\forall n_1, n_2$: Node. $\neg(n_1 \neq n_2 \wedge \neg L(n_1) \wedge L(n_2) \wedge id[n_1] > id[n_2])$

Interp(3)

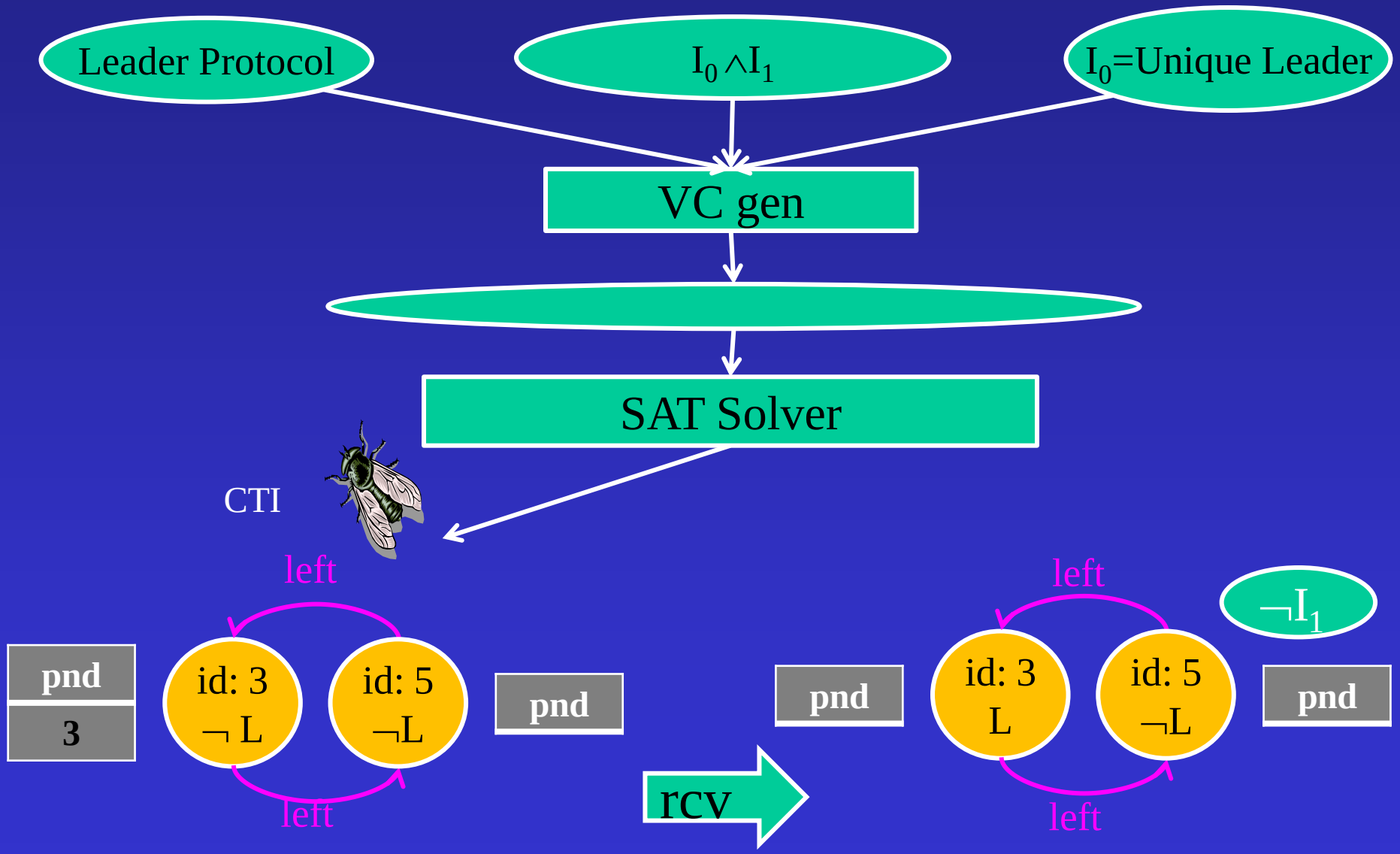
id:5

id:3
L

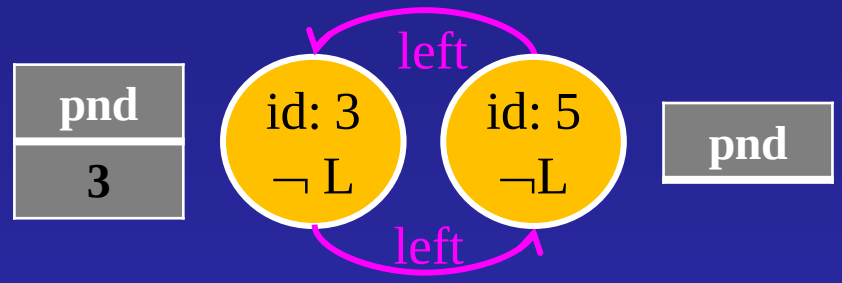
$I_1 = \forall n_1, n_2$: Node. $\neg(L(n_2) \wedge id[n_1] > id[n_2])$

This looks good, add to the invariant

Algorithmic Deductive Verification(2)

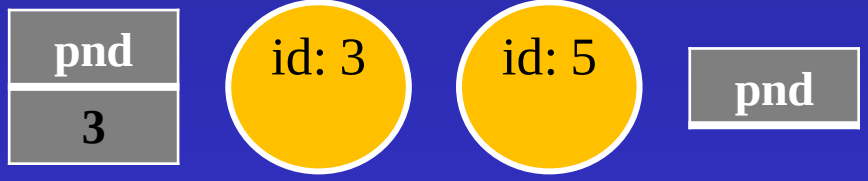


Generalizing CTIs(2)



Only the highest id can be self pending

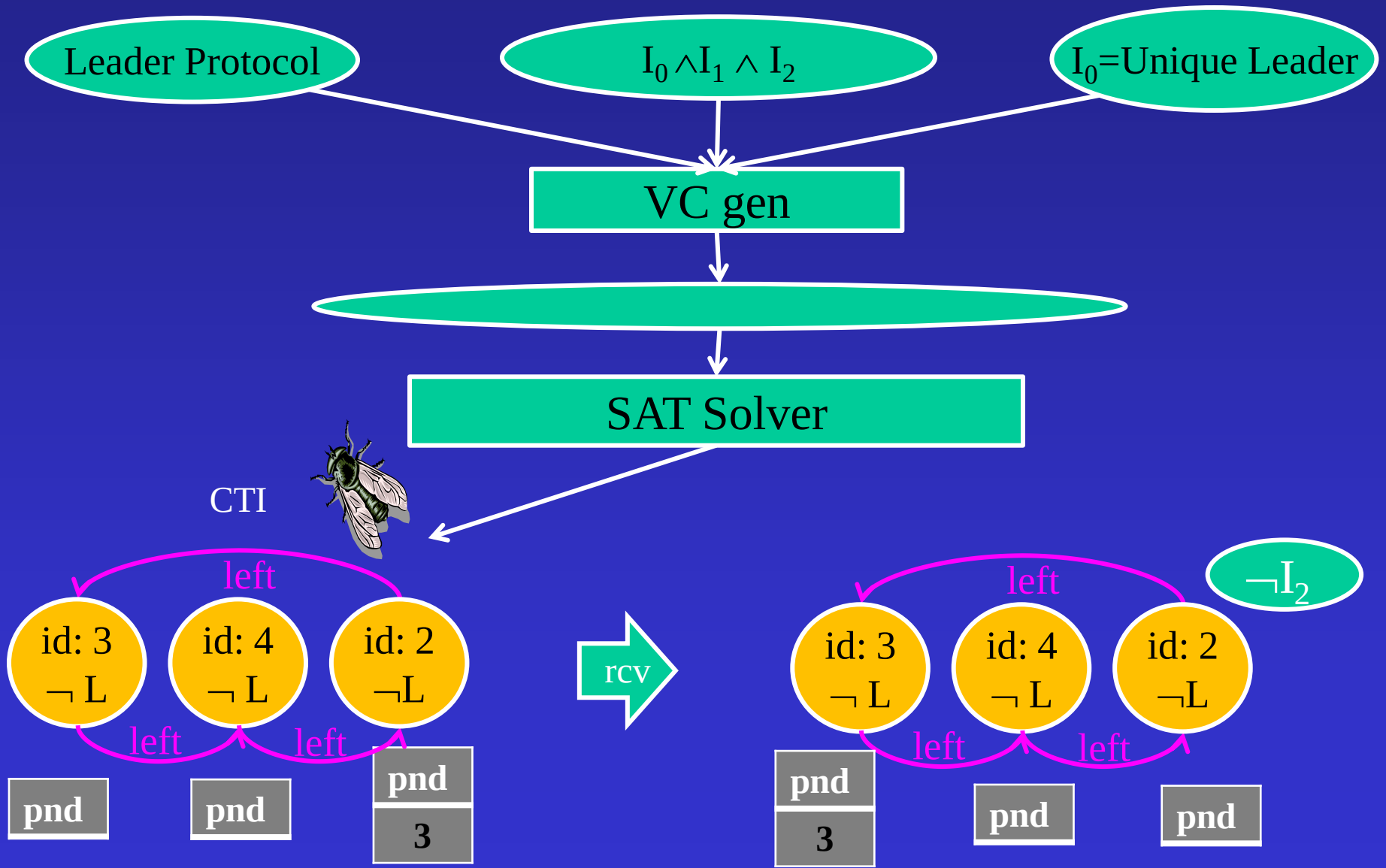
Project to {pnd, id, ⩽}



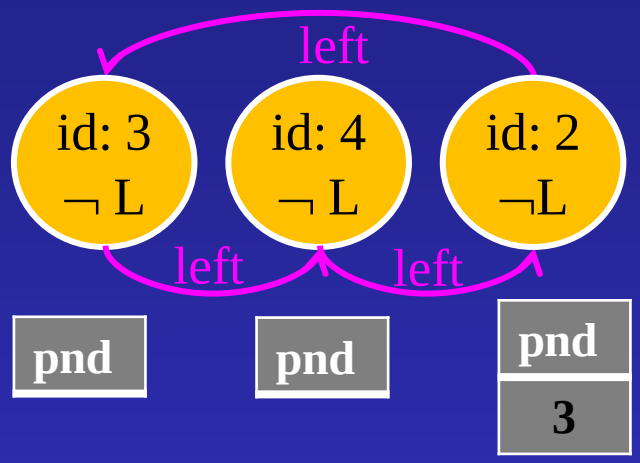
Interp(3)

$I_2 = \forall n_1, n_2$: Node. pending(id[n_1], n_1) $\Rightarrow$ id[n_2] $\leqslant$ id[n_1]

Algorithmic Deductive Verification(3)

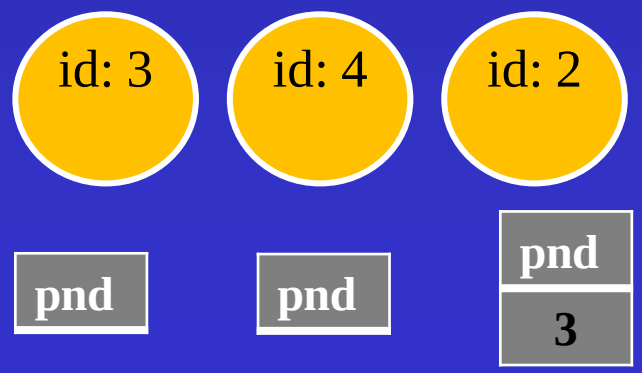


Generalizing CTIs(3)



Cannot overtake nodes with higher ids

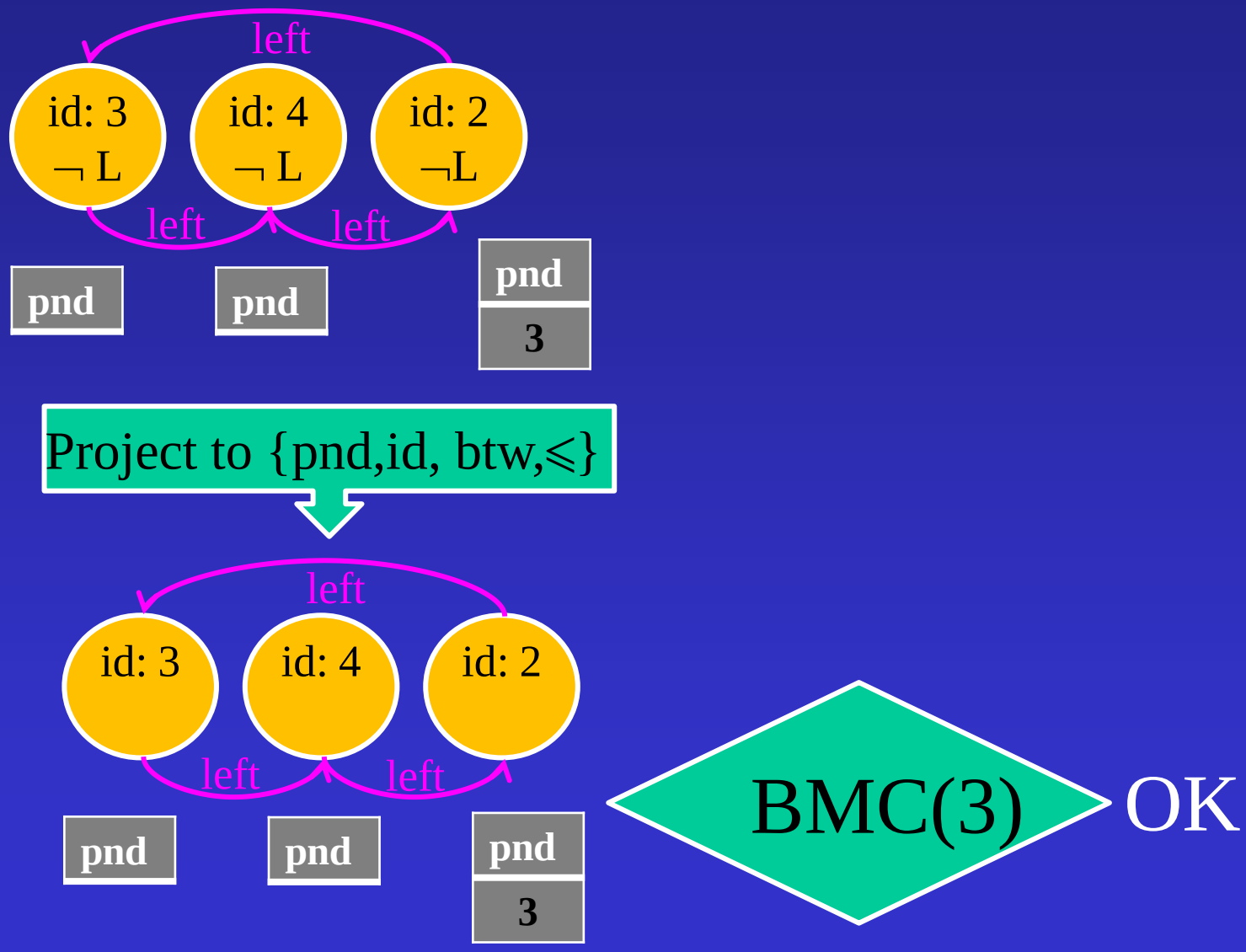
Project to {pnd,id,⩽}



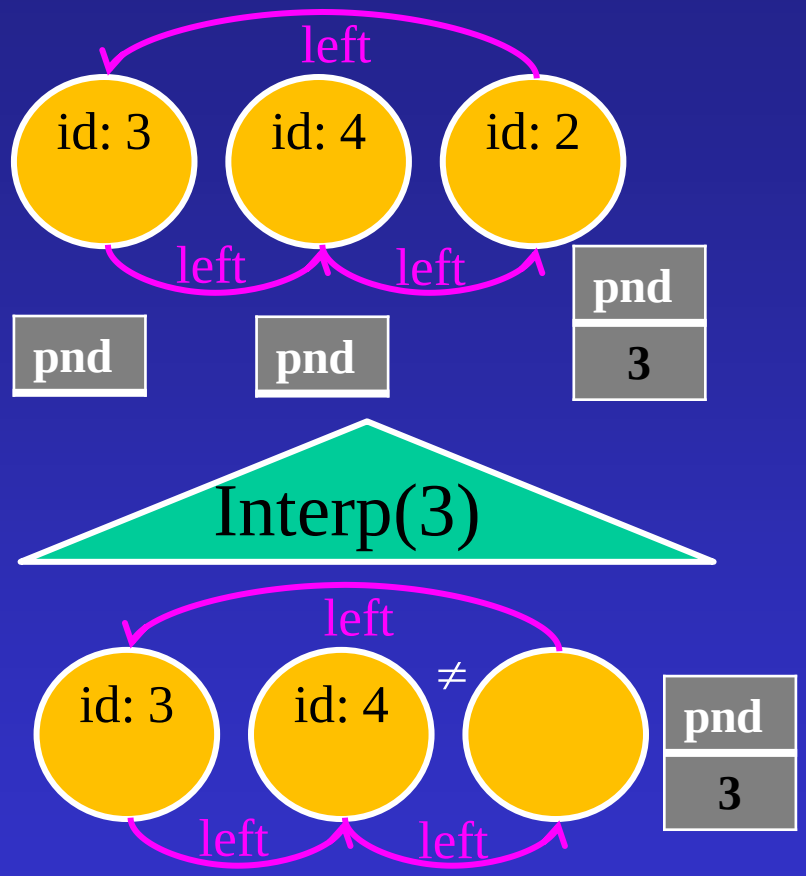
BMC(3)

NOT OK

Generalizing CTIs(3)



Generalizing CTIs(3)



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

Algorithmic Deductive Verification(4)

Leader-Protocol

$I_0 \wedge I_1 \wedge I_2 \wedge I_3$

I_0=Unique Leader

VC gen

Verification Conditions:

1) $Init \wedge \neg I$

2) $I(V) \wedge \neg \varphi(V)$

3) $[\![P]\!](V, V') \wedge I(V) \wedge \neg I(V')$

SAT Solver

Proof

Verified Protocols

Protocol	Model Types	Model Relations	Property Size (# Literals)	Invariant Size (# Literals)	CTI Gen. Steps
Leader in Ring	2	5	3	9	3
Learning Switch	2	6	11	15	5
DB Chain Replication	4	13	11	35	7
Chord	1	13	35	46	4
Consensus	3	7	5	13	3

Current Work

- User study
- Improved UI
- Support for modular reasoning
- Applications
  - More protocols:
    - RAFT, PAXOS
  - Filesystem
- Testing distributed protocols

Increasing Expresiveness (EPR++)

- Used derived "materialized" relations
  - n* = TC[n] where n is deterministic
  - trusted(h) = ∃h'. in(h') ∧sent(h', h)
  - Explore locality
- Restricted functions: Idempotent

[CAV'13] S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, S: Effectively-Propositional Reasoning about Reachability in Linked Data Structures
[POPL'14] S. Itzhaky, A. Banerjee, N. Immerman, O. Lahav, A. Nanevski, S: Modular reasoning about heap paths via effectively propositional formulas

Increasing Speed & Automation

- More automated generalization [CAV'15, POPL'16]
  - But still diagnosable
- Better SAT solvers for EPR
  - Can we do better than NEXPTIME?
    - Explore domain knowledge

[CAV'15] A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetzky, S. Shoham: Property-directed inference of universal invariants or proving their absence
[POPL'16] O. Padon, N. Immerman, A. Karbyshev, S. Shoham, S: Decidability of inferring inductive invariants

Lessons Learned so far

- 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
  - Many important properties can be captured in a (parametric) model that abstracts the actual numerical values
  - Unbounded topologies
  - Unique paths

The road ahead

- "Supervised" formal verification
  - Find the right interaction model for supervised formal verification of expressive properties
  - Ask questions in the language of the domain expert, not in the language of the verification system
- Push the limits of symbolic reasoning
- Quantitative/Probabilistic correctness
- Realistic applications
  - Cloud & Networks [PLDI'14, TACAS'16, Sub'16]
  - Data-intensive systems (streaming, big data) [ERC]

[PLDI'14] T. Ball. N. Bjorner. A. Gember, S. Itzhaky, A. Karbyshev, S, M. Schapira, A. Valdarsky: towards verifying controller programs in software-defined networks
[TACAS'16] Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, S, S. Shenker and S. Shoham. Some Complexity Results for Stateful Network Verification
[Sub'16] A. Panda, O. Lahav, K. Argyraki, S, S. Shenker:
Verifying Isolation Properties in the Presence of Middleboxes

Tentative Schedule

Date	Subject	Presenter
28/2	Introduction	Mooly
6/3	IVY	Mooly
13/3	Coq Tutorial	Oded Padon
20/3	Network Verification	Mooly
27/3	Comprehensive Formal Verification of an OS Microkernel	Elazar Gershuni
3/4	Ironclad	Vadim Stotland
10/4	Formal verification of a realistic compiler:	Omer Anson
1/5	A Design and Verification Methodology for Secure Isolated Regions	Yotam Feldman

Backup Slides