Modularity for Accurate Static Analysis of Smart Contracts

MOOLY SAGIV

CERTORA

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

The Smart Contract Spechub

JUNE 2019

And also…

Noam Rinetzky

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

James Wilcox

W UNIVERSITY of WASHINGTON

Ittai Abraham

Guy Golan-Gueta

vmware

David Dill

Yan Michalevsky

Yoni Zohar

Shelly Grossman

Marcelo Taube

CERTORA

Pain Point: Buggy & Untrusted Software Components

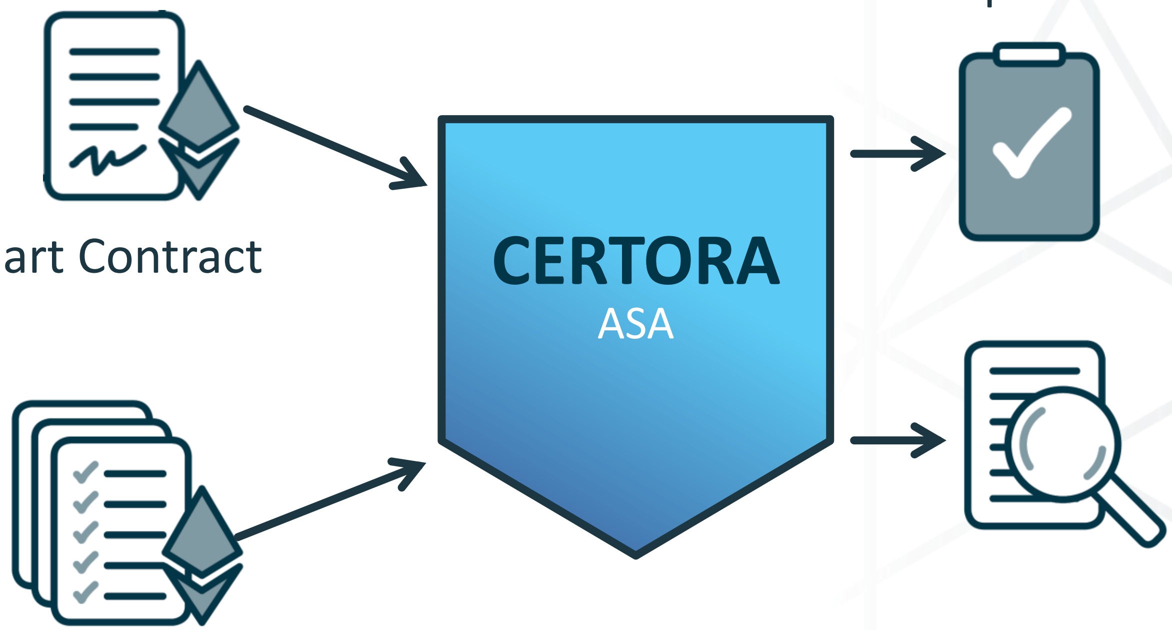
- Software is Eating the World
- Software applications are built of numerous disparate sources
  - unknown
  - untrusted
  - constantly evolving
- Correctness of code = safety, money, human life, …
- Even worse in blockchain
  - Immutability: code is law
  - Cryptocurrency: code is money

The Business Case

- Problem: Automated financial contracts
  - Bugs in contracts = money lost to adversaries for ever
- Pain: Very hard to find bugs in the contracts
  - Lots of examples where people have lost large amounts of money
  - Customers are willing to >= 6 figures for solutions
- Solution: Automatic Verification
  - Find bugs, or prove the absence of bugs (one or the other!)
  - Key enabler: **higher level specifications** that can be checked
    - Developers are willing to do most of the work, for reward
    - Standards ERC20, ERC721

Certora’s Mission

- Develop a library of reusable correctness rules
- Community effort
  - ~~Code is the law~~ → Spec is the law
  - No overdraft
  - If no transaction is executed then no cost
  - No radical currency changes



- Develop unique static analysis of code

Business Snapshot

- Top-tier paying customers:
  - Compound Finance

    "We installed Certora's technology and it is used daily by our software engineers to locate mind blowing bugs"

    Geoff Hayes

    CTO of Compound Finance
  - Coinbase

    "The Certora ASA surfaces problems before a contract is available on our platforms, to help us better inform our customers of risk. The ASA has already surfaced significant problems missed by expensive and unscalable manual audits."

    Shamiq Islam

    Head of Security at Coinbase Global
- Investors
  - Scott Shenker, repeat unicorn founder (Nicira, Databricks, Nefali)
  - Coinbase

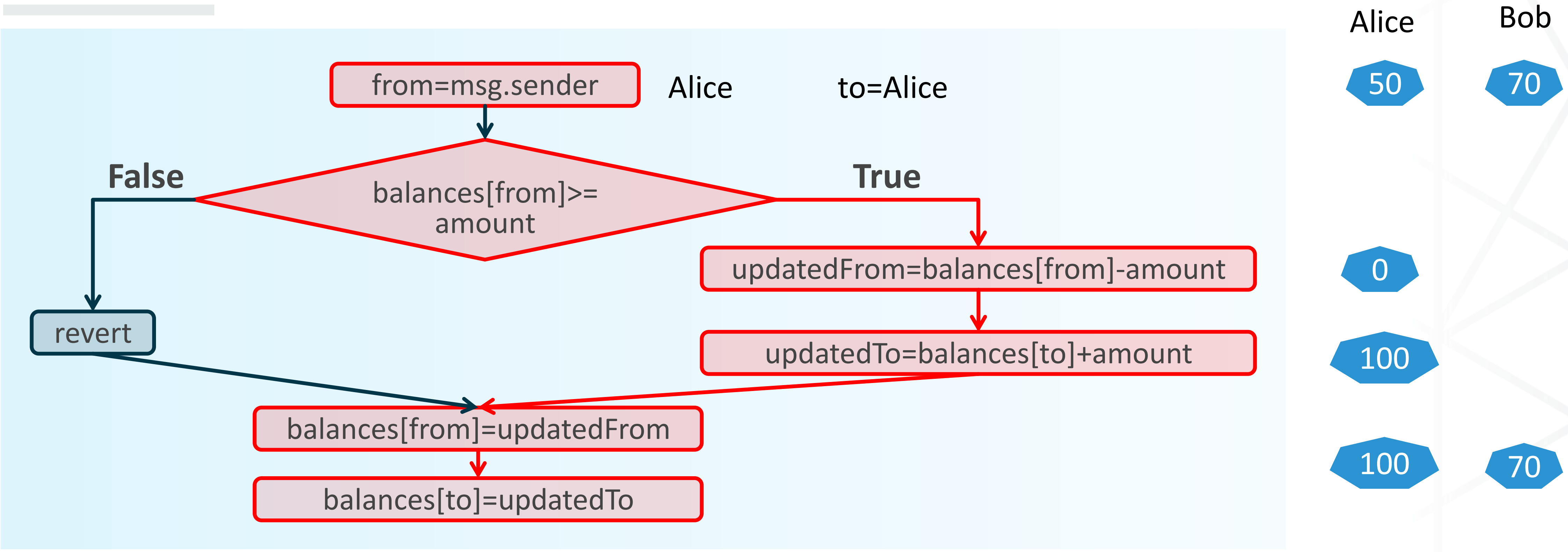
Toy ERC20 token

```
contract toyERC20 {
    mapping (address => uint) balances;
    constructor(address bank, uint initial_amount) {
        balances[bank] = initial_amount;
    }
    function transfer(address to, uint amount) {
        uint updatedFrom;
        uint updatedTo;
        address from = msg.sender;
        if (balances[from] >= amount) {
            updatedFrom = balances[from] - amount;
            updatedTo = balances[to] + amount;
        } else {  revert();  }
       balances[from] = updatedFrom;
       balances[to] = updatedTo;
    }
}
```

invariant $\Sigma_{\text{a: address}}$ balances[a]

CERTORA

Toy ERC20 token



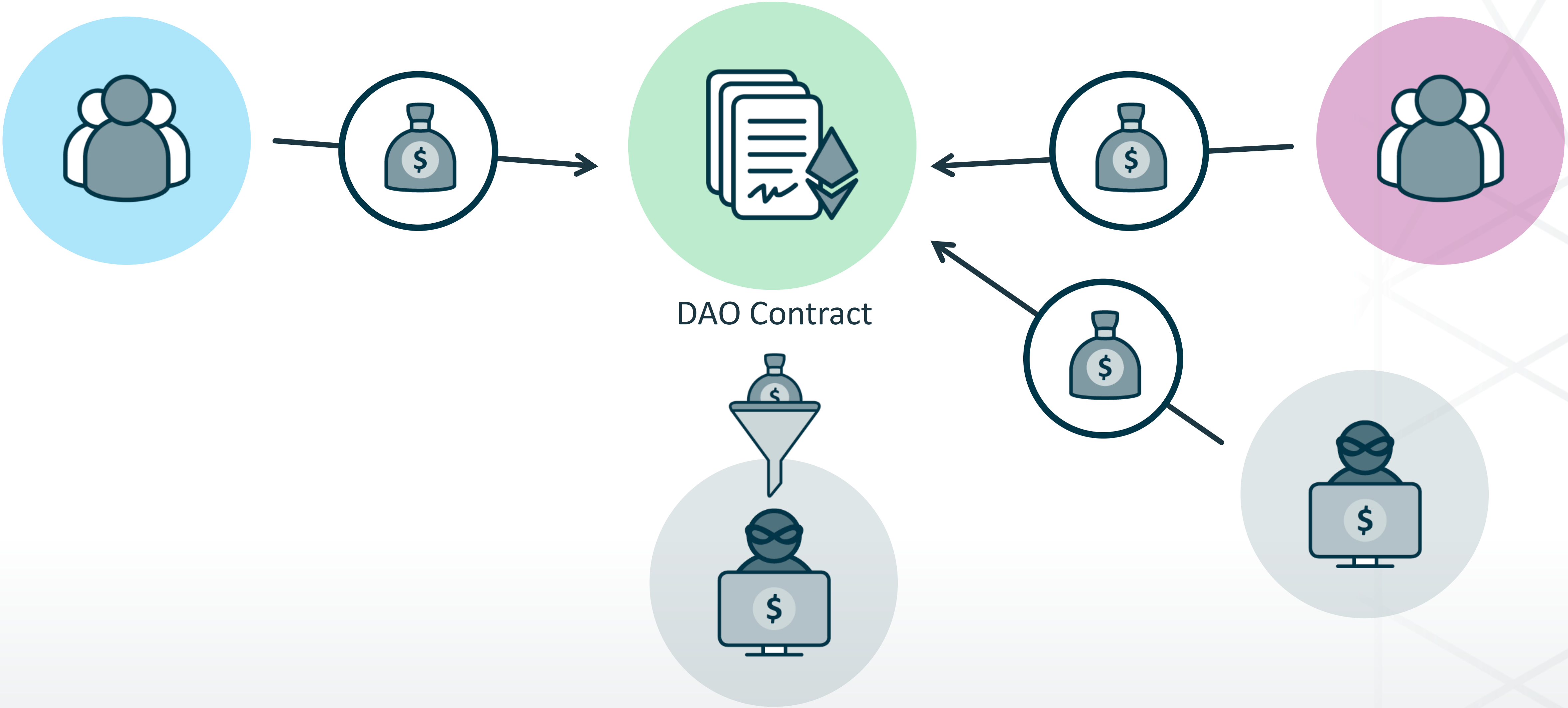
$\texttt{invariant}\ \Sigma_{\text{a: address}}\ \texttt{balances[a]}$

Fixed Toy ERC20 token

```
contract toyERC20 {
    mapping (address => uint) balances;
    constructor(address bank, uint initial_amount)
{
        balances[bank] = initial_amount;
    }
    function transfer(address to, uint amount) {
        uint updatedFrom;
        uint updatedTo;
        address from = msg.sender;
        require from != to ;
        if (balances[from] >= amount) {
                updatedFrom = balances[from]-amount;
                updatedTo = balances[to] + amount;
        } else {  revert();  }
         balances[from] = updatedFrom;
         balances[to] = updatedTo;
    }
}
```

invariant $\Sigma_{a:\ address}$ balances[a]

Reentrancy attacks



Reentrancy attacks

```
DAO_withdraw(to) {
 if (shares[to] > 0) {
  to.send(shares[to]);
  shares[to] = 0 ;
 }
}
```

```
f () {
  DAO(x).withdraw(me)
}
```

Immune Reentrancy attacks(Atomicity)

```
DAO_withdraw(to) {
 if (shares[to] > 0) {
  to.send(shares[to]);
  shares[to] = 0 ;
 }
}
```

```
DAO_withdraw(to) {
 if (shares[to] > 0) {
  shares[to] = 0 ;
  to.send(shares[to]);

 }
}
```

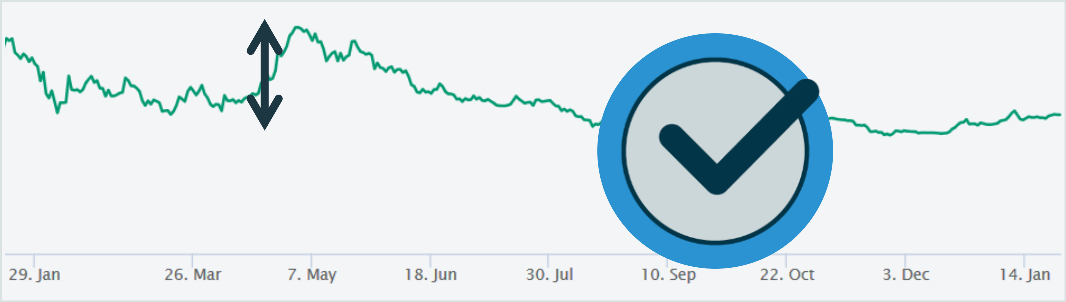
Atomicity[POPL'18]

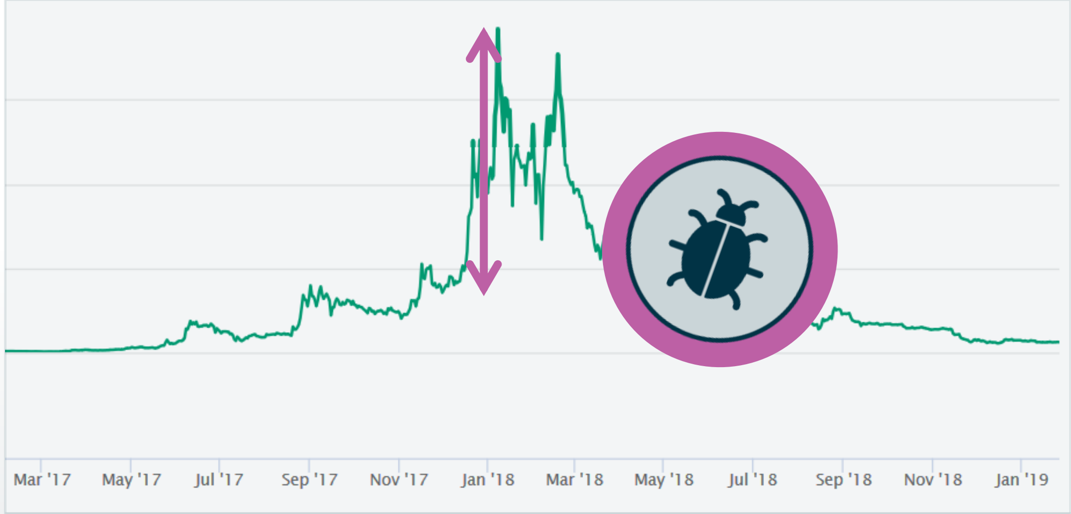
- Contracts that are vulnerable to reentrancy attacks are dangerous to use
  - Sensitive to changes in the EVM
  - Constantinople fork postponement
- **Most precise method**
- Guarantee atomicity in presence of callbacks

Math is the law

Geoff Hayes | CTO Compound

- **Goal:** enable human mitigation of money theft
- **Requirement:** Price changes must be less than 10% every hour



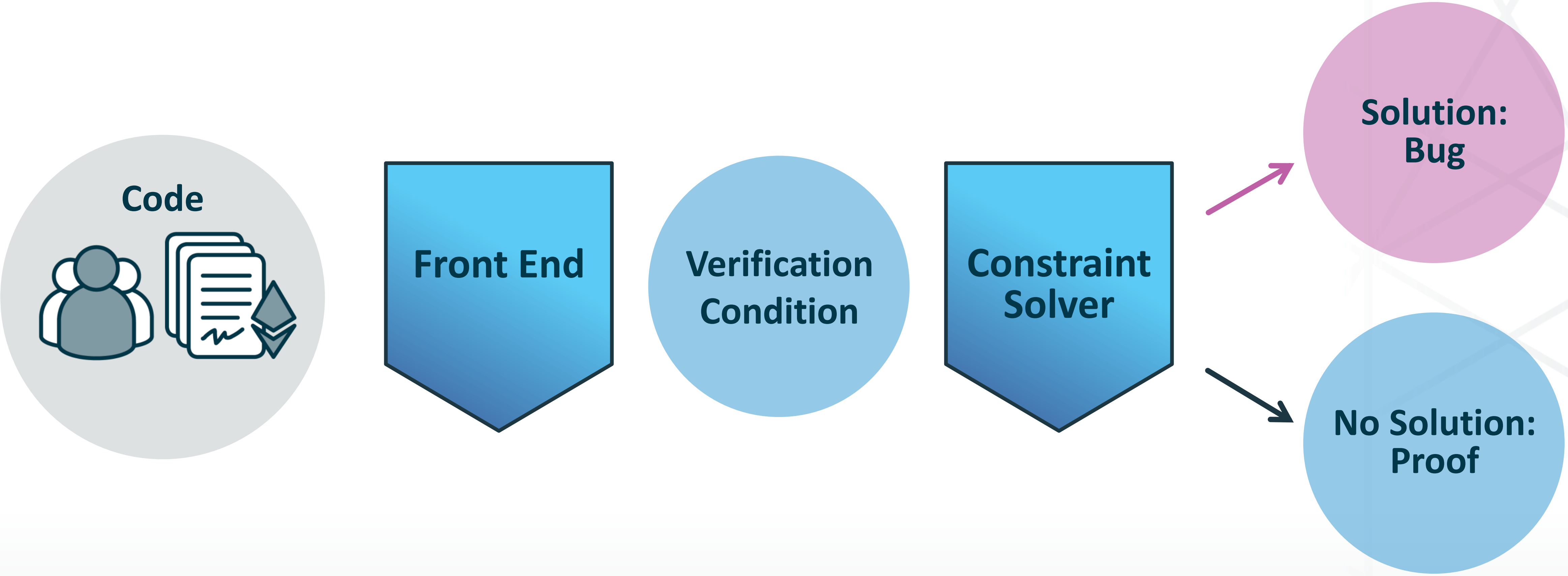



```
(err, onePlusMaxSwing) = addExp(one, maxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// max = anchorPrice * (1 + maxSwing)
(err, max) = mulExp(anchorPrice, onePlusMaxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// If price > anchorPrice * (1 + maxSwing)
// Set price = anchorPrice * (1 + maxSwing)
if (greaterThanExp(price, max)) {
    return (Error.NO_ERROR, true, max);
}

(err, oneMinusMaxSwing) = subExp(one, maxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// min = anchorPrice * (1 - maxSwing)
(err, min) = mulExp(anchorPrice, oneMinusMaxSwing);
// We can't overflow here or we would have already overflowec
assert(err == Error.NO_ERROR);

// If  price < anchorPrice * (1 - maxSwing)
// Set price = anchorPrice * (1 - maxSwing)
if (lessThanExp(price, min)) {
    return (Error.NO_ERROR, true, min);
}
```

For all t1,t2. |t2-t1| < 1 hour, |p2-p1| < 0.1p1

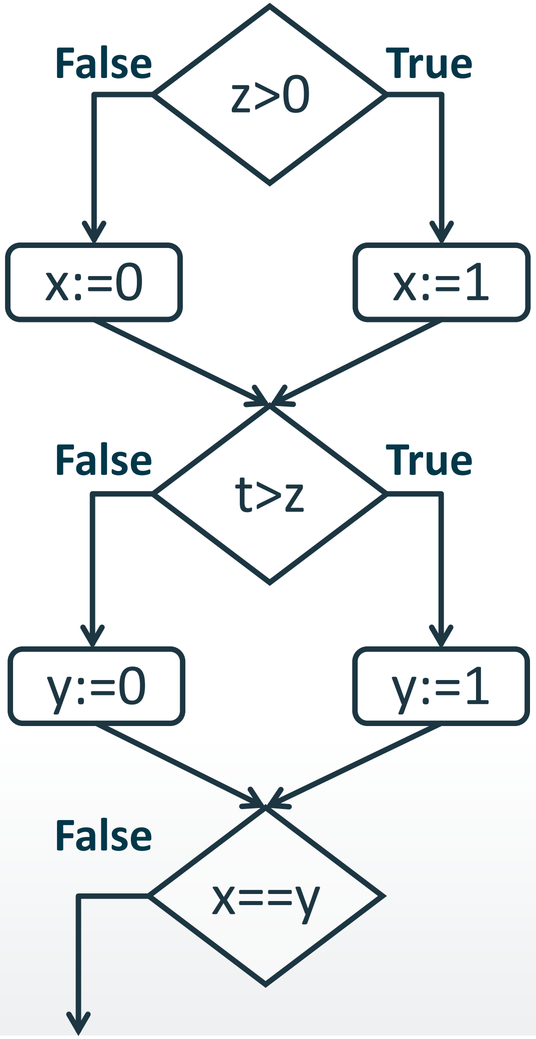
CERTORA

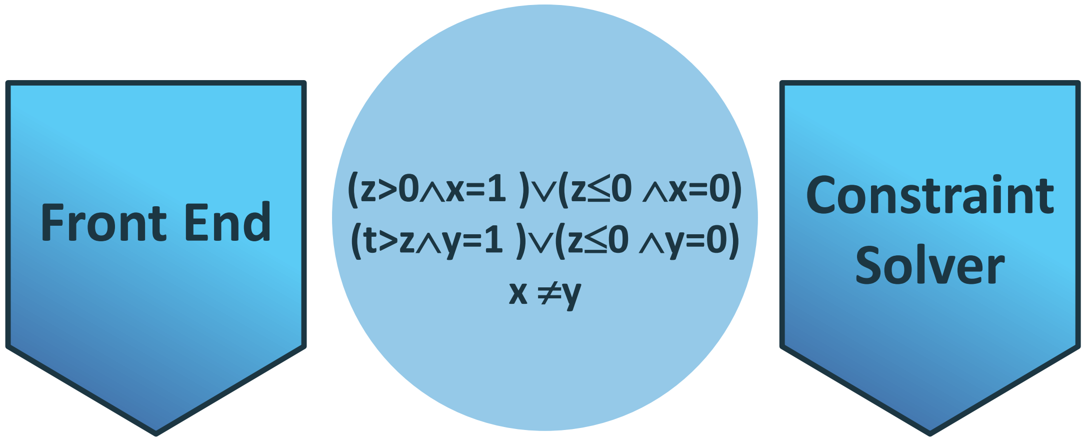
ASA via Constraint Solving

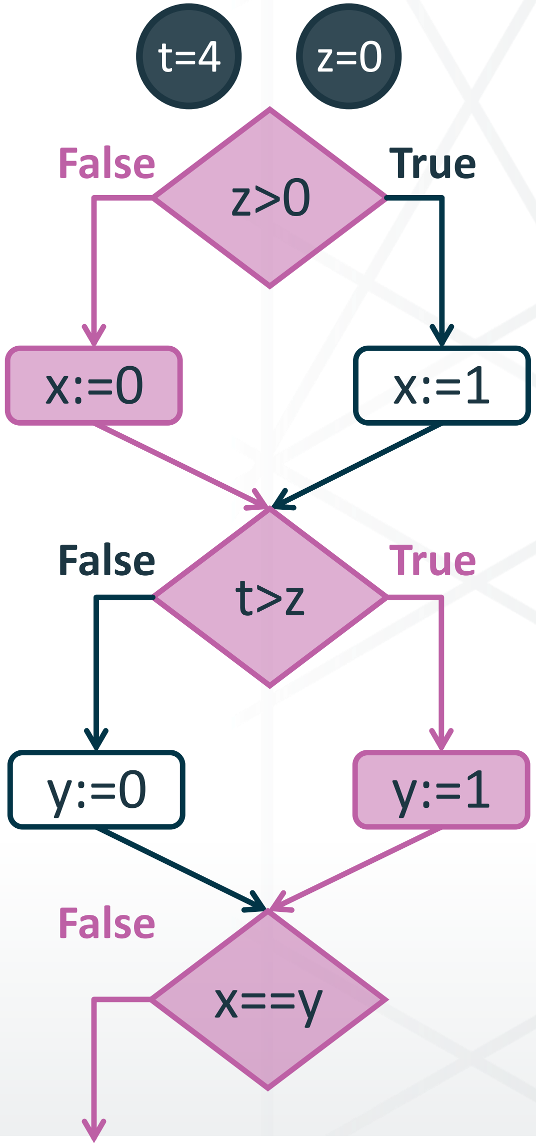


[Z3] Microsoft Research, [CVC4] Stanford University, [Yices] Stanford Research Institute **SMT***

ASA via Constraint Solving







PLDI Philadelphia 2018

Modularity for decidability of deductive verification with applications to distributed systems

Mooly Sagiv

European Research Council
Established by the European Commission

Tel Aviv University

Contributors

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

TEL AVIV UNIVERSITY

UCLA

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

Microsoft Research

TEL AVIV UNIVERSITY

TEL AVIV UNIVERSITY

James R. Wilcox, Doug Woos

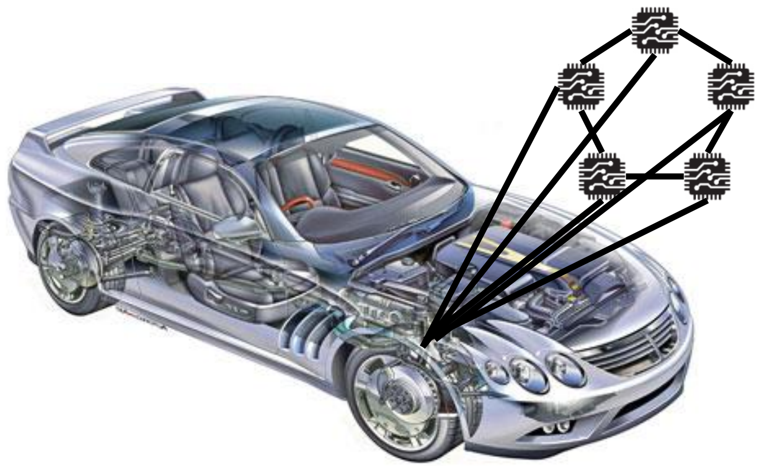
UNIVERSITY of WASHINGTON

And Also

Anindya Benerjee | Yotam Feldman | Neil Immerman | Aurojit Panda

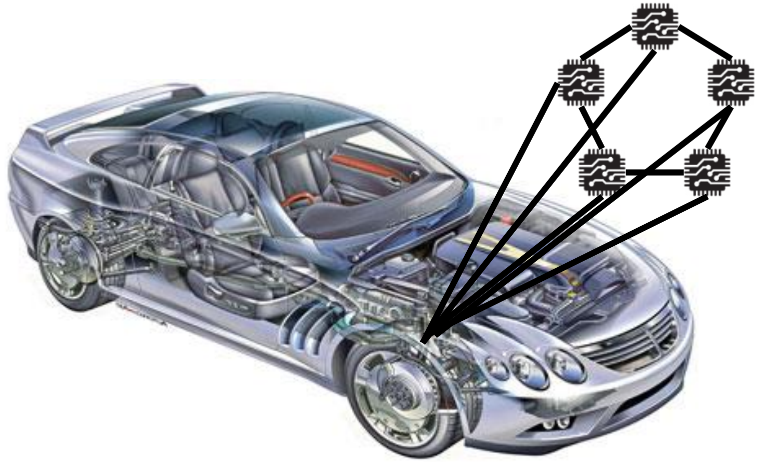
Shachar Itzhaky | Aleks Nanevsky | Orr Tamir | Robbert van Renesse

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-Pee
for Internet Appli

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Kar
Hari Balakrishnan, *Memb*

Attractive features of Chord include i
correctness, and provable performanc
concurrent node arrivals and departure

CCR'12

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.

SOSP'07

Best Paper Award

Zyzzyva: Speculative Byzantine Fault Tolerance

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

Zyzzyva is a state machine replication protocol based on ...rotocols: (1) agreement, (2) view change, and (3) ...ement protocol orders requests for exe- ...ew change protocol coordinates

CACM'08

Zyzzyva: Sp... Faul...

ACM Transactions on Computer Systems '09

Zyzzyva: 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 Zyzzyva does not guarantee safety.

What about correctness of the low level implementation?

Decidable Reasoning for Verification: How Far Can You EPR?

Oded Padon

PhD Thesis

http://www.cs.tau.ac.il/~odedp

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

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

[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

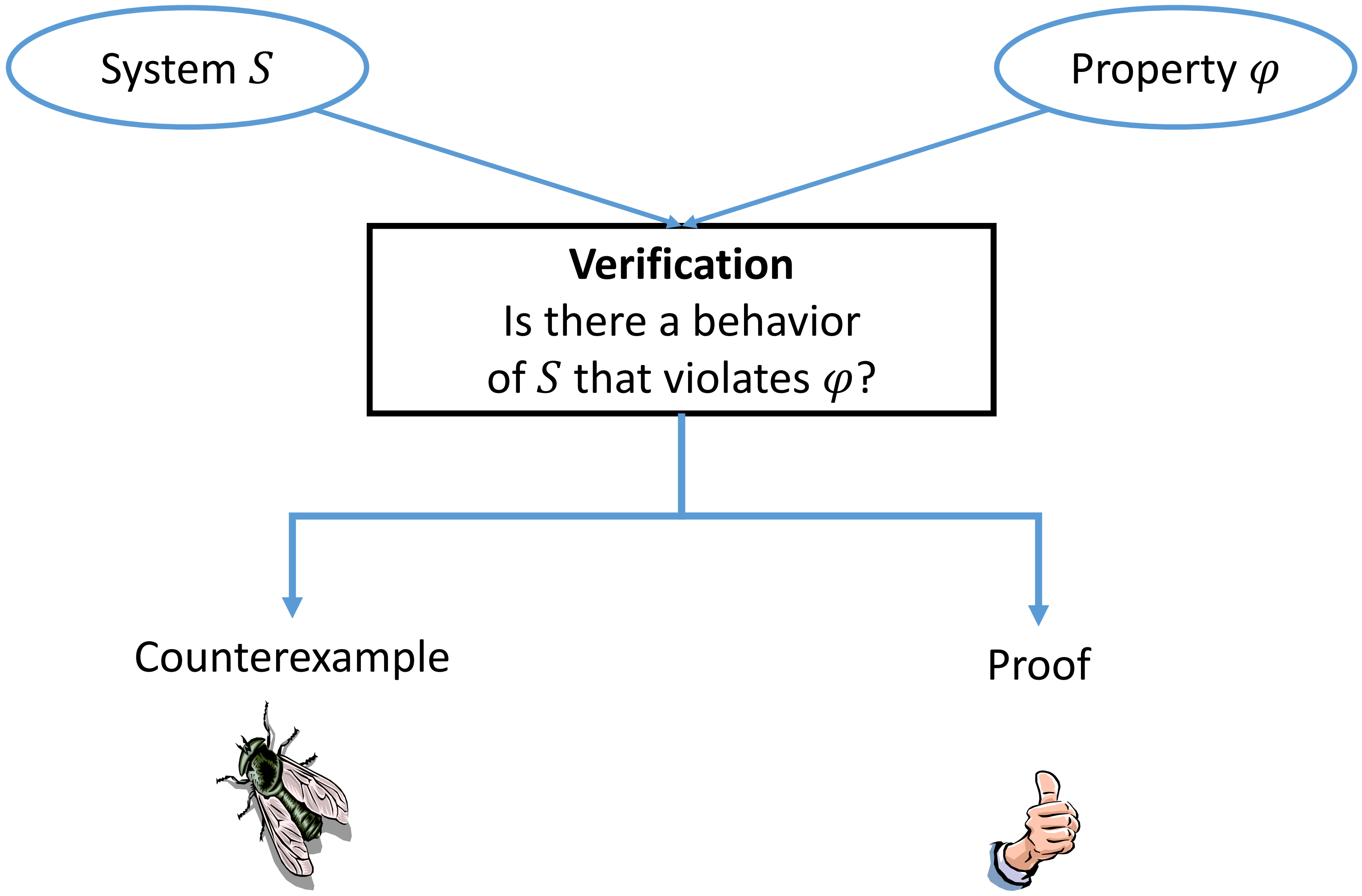
[POPL'18] Oded Padon, Jochen Hoenicke, Giuliano Losa, Andreas Podelski, MS, Sharon Shoham
Reducing liveness to safety in first-order logic. PACMPL 2(POPL): 26:1-26:33 (2018)

[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

[FMCAD'18] Oded Padon, Jochen Hoenicke, Kenneth L. McMillan, Andreas Podelski, MS, Sharon Shoham:

Temporal Prophecy for Proving Temporal Properties of Infinite-State Systems. FMCAD 2018: 1-11

Automatic verification of infinite-state systems



Naïve period in program verification 70’s

ROBERT W. FLOYD

ASSIGNING MEANINGS TO PROGRAMS[1]

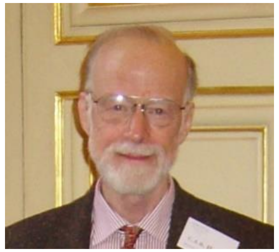
INTRODUCTION

This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion

An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen’s University of Belfast, Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This in-



Programming Languages — T.A. Standish, Editor

Guarded Commands, Nondeterminacy and Formal Derivation of Programs

Edsger W. Dijkstra
Burroughs Corporation

“Program testing can be used to show the presence of bugs, but never to show their absence!” Dijkstra (1970)

Disillusionment in program verification 80's

Program P

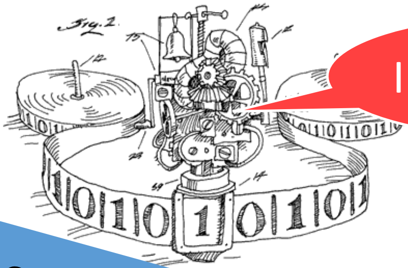
Safety Property φ

Verification

Is there a behavior

...that violates φ?

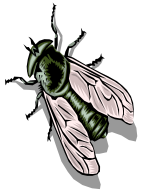
Rice's Theorem ➔



I can't decide!

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

Counterexample



Unknown

Proof

[POPL'78, CACM'79] R.A. DeMillo, R.J. Lipton, A. J. Perlis: Social Processes and Proofs of Theorems and Programs

Challenges in program verification

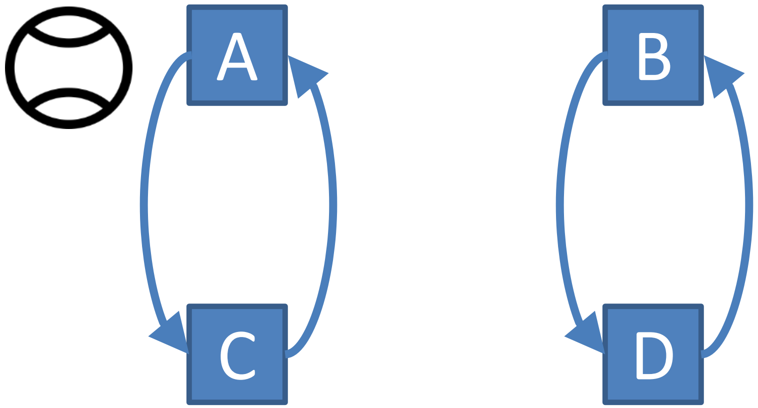
- Specifying program behavior
- Asymptotic complexity of program verification
  - The halting problem
  - Rice theorem
  - The ability of simple programs to represent complex behaviors
- The complexity of realistic systems
  - Huge code
  - Heterogeneous code
  - Missing code

Mathematical Induction

- P(n) is a property of natural number n
- To show that P(n) holds for every n, it suffices to show that:
  - P(0) is true
  - If P(m) is true then P(m+1) is true for every number m
- In logic
  - $(P(0) \wedge \forall m \in N.\ P(m) \Rightarrow P(m+1)) \Rightarrow \forall n \in N.\ P(n)$

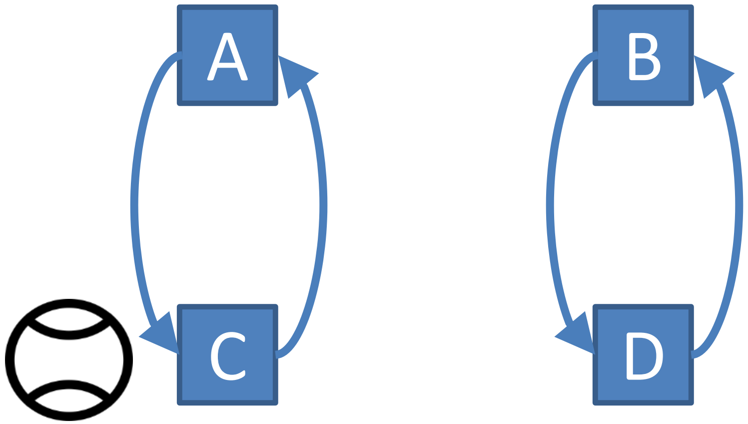
Induction on a ball game

- 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?



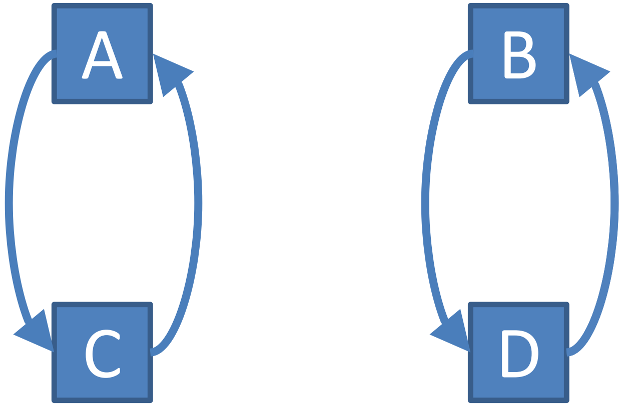
Induction on a ball game

- 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?



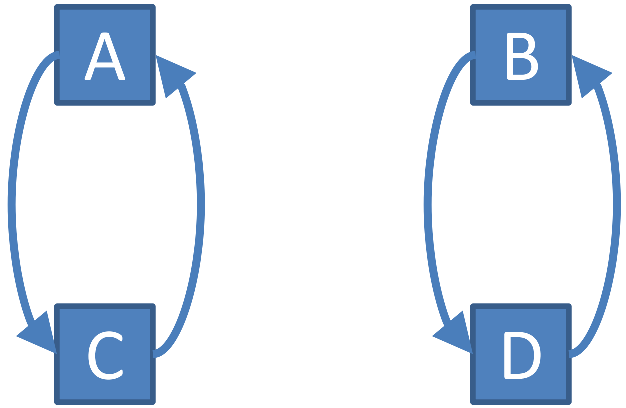
Formalizing with induction

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



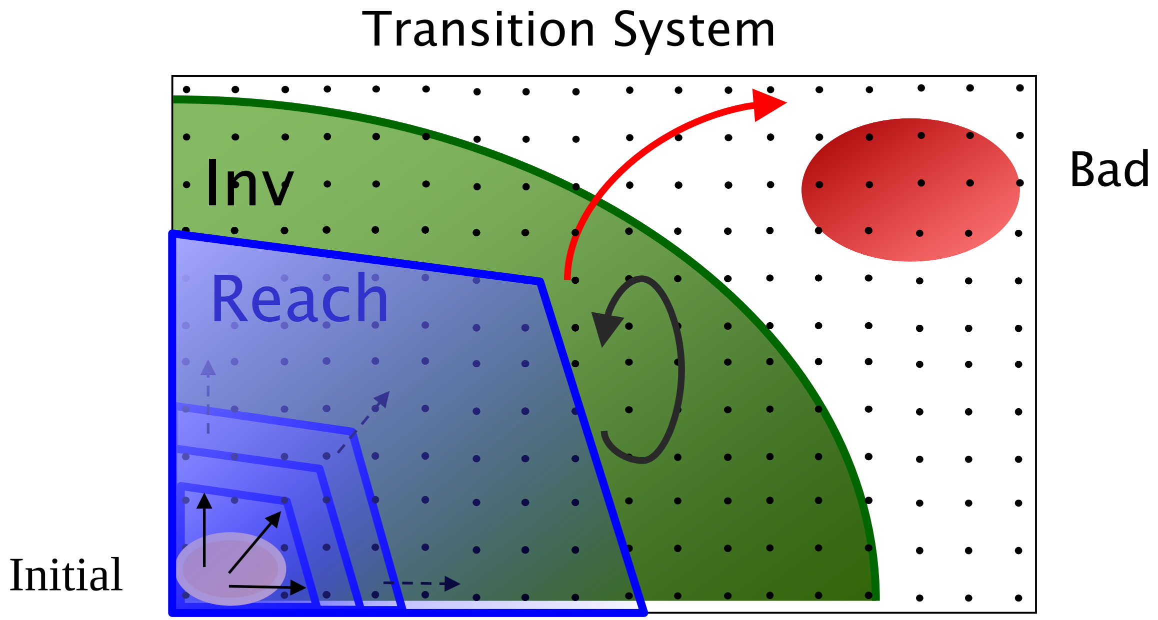
Formalizing with induction

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



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

Inductive Invariants



The program is **safe** if all the reachable states satisfy the property

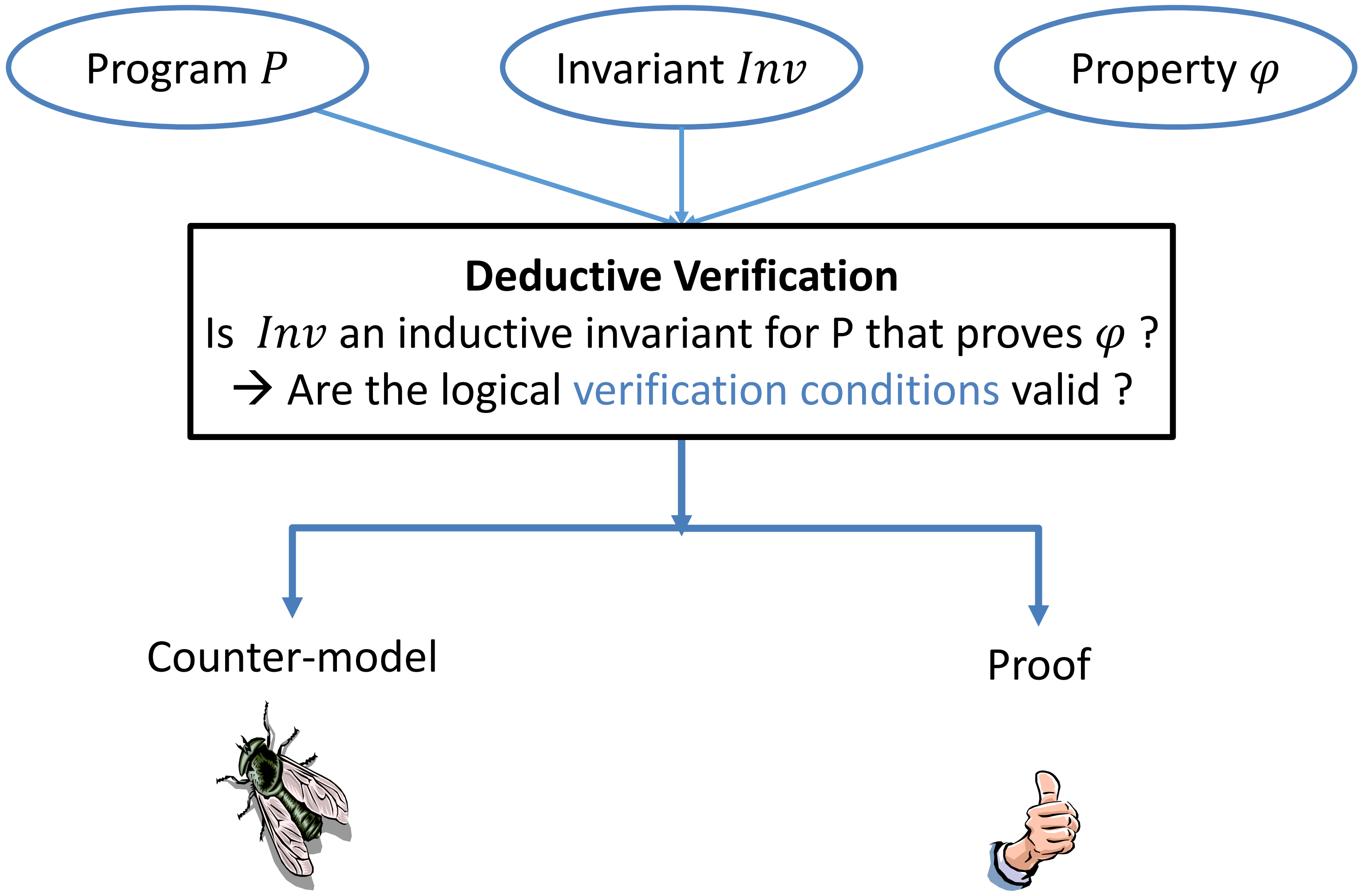
The program is safe with respect Bad iff there exists an inductive invariant Inv satisfying:

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

$Init \subseteq Inv$ (Initiation)

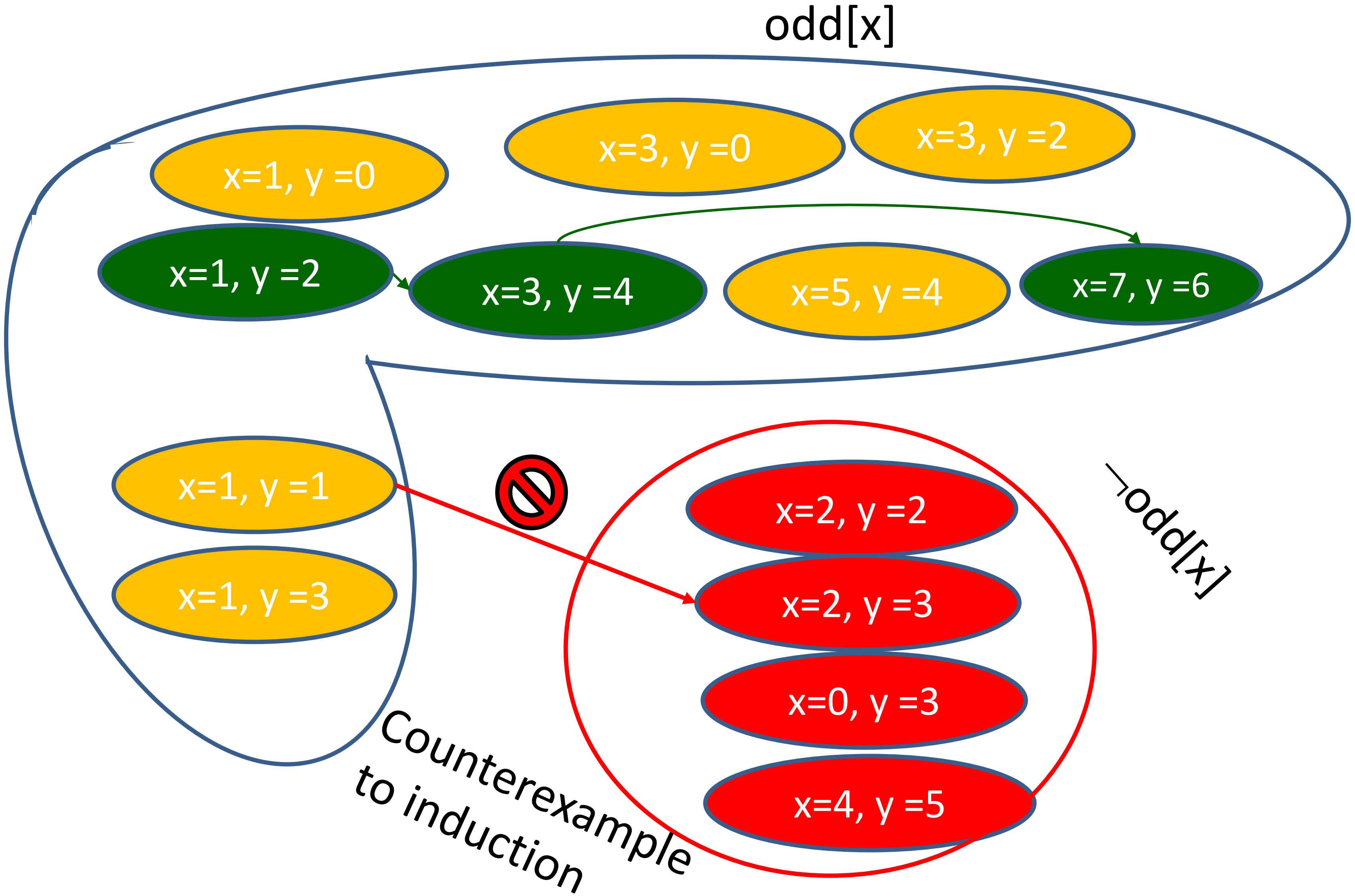
if $\sigma \in Inv$ and $\sigma\, T\, \sigma'$ then $\sigma' \in Inv$ (Consecution)

Deductive verification



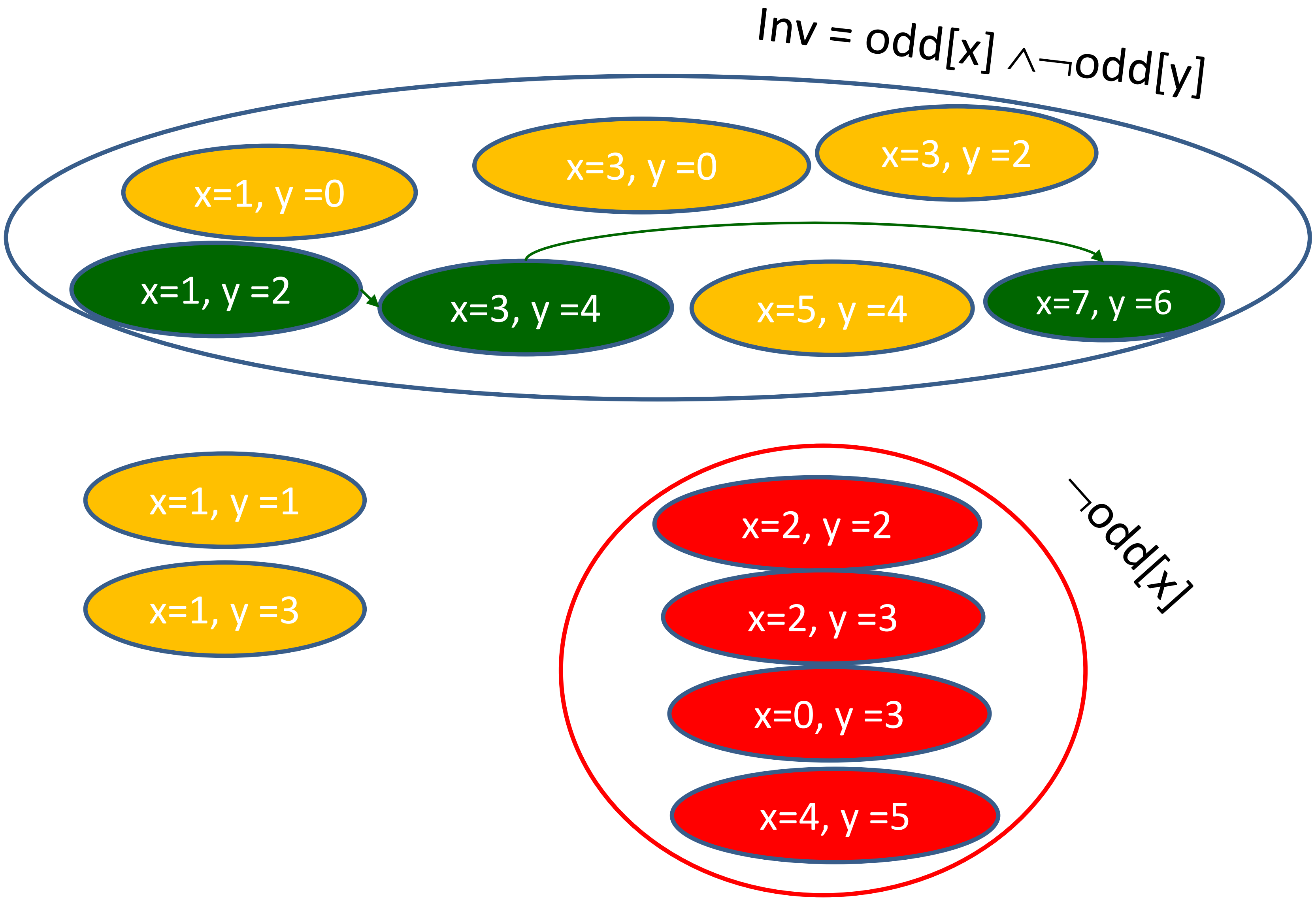
Simple Example: inductive Invariants

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

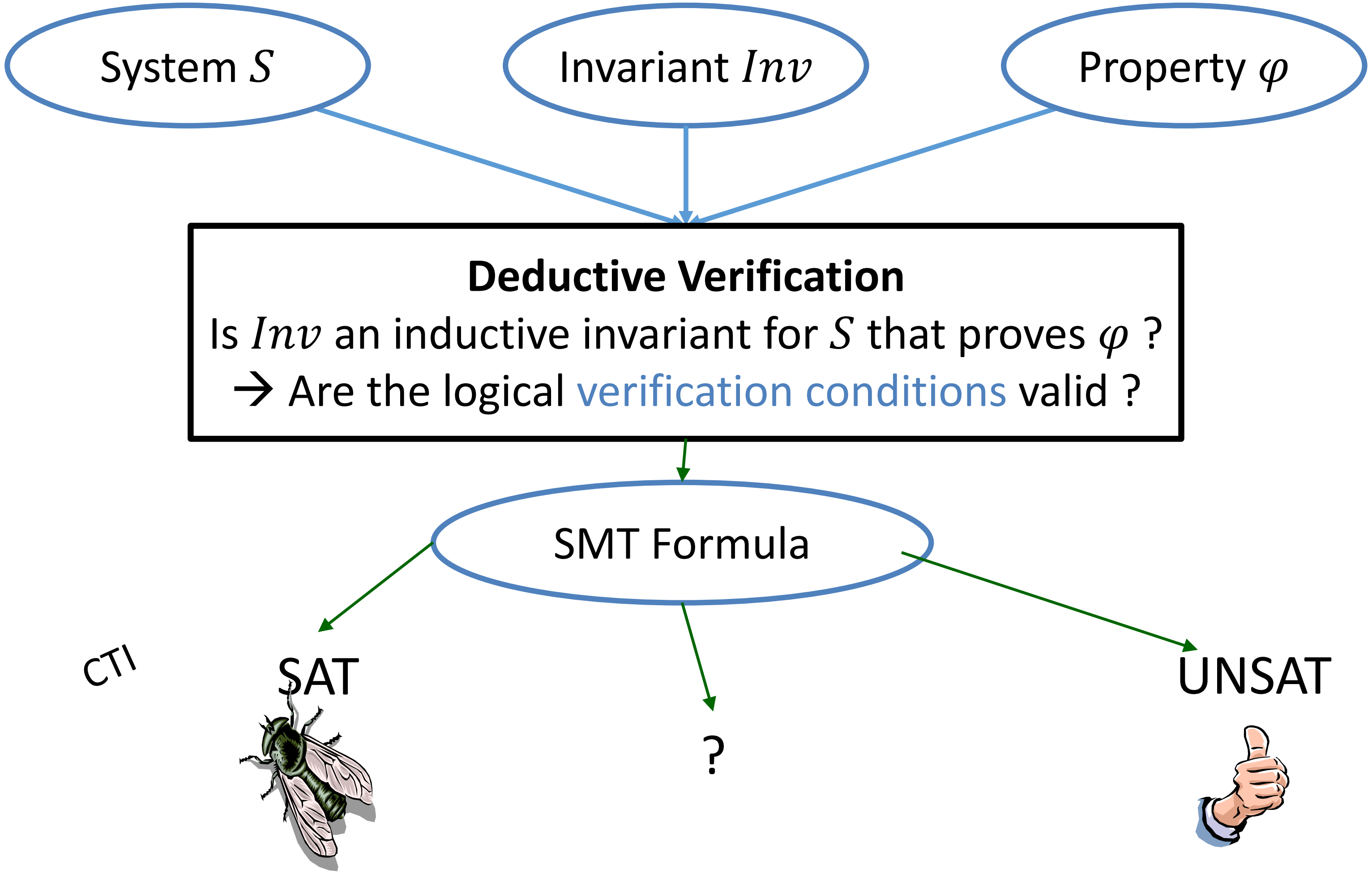


Simple Example: inductive Invariants

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



Dafny [Leino'17]



K. Rustan M. Leino: Accessible Software Verification with Dafny. IEEE Software 34(6): 94-97 (2017)

Deductive verification

System S

Invariant Inv

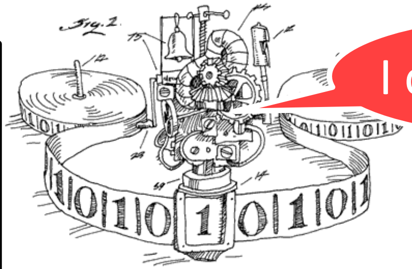
Property φ

Deductive Verification

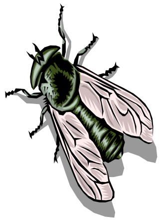
Is Inv an inductive invariant for S that proves φ ?

→ Are the logical verification conditions valid ?

Church's Theorem



Counter-model

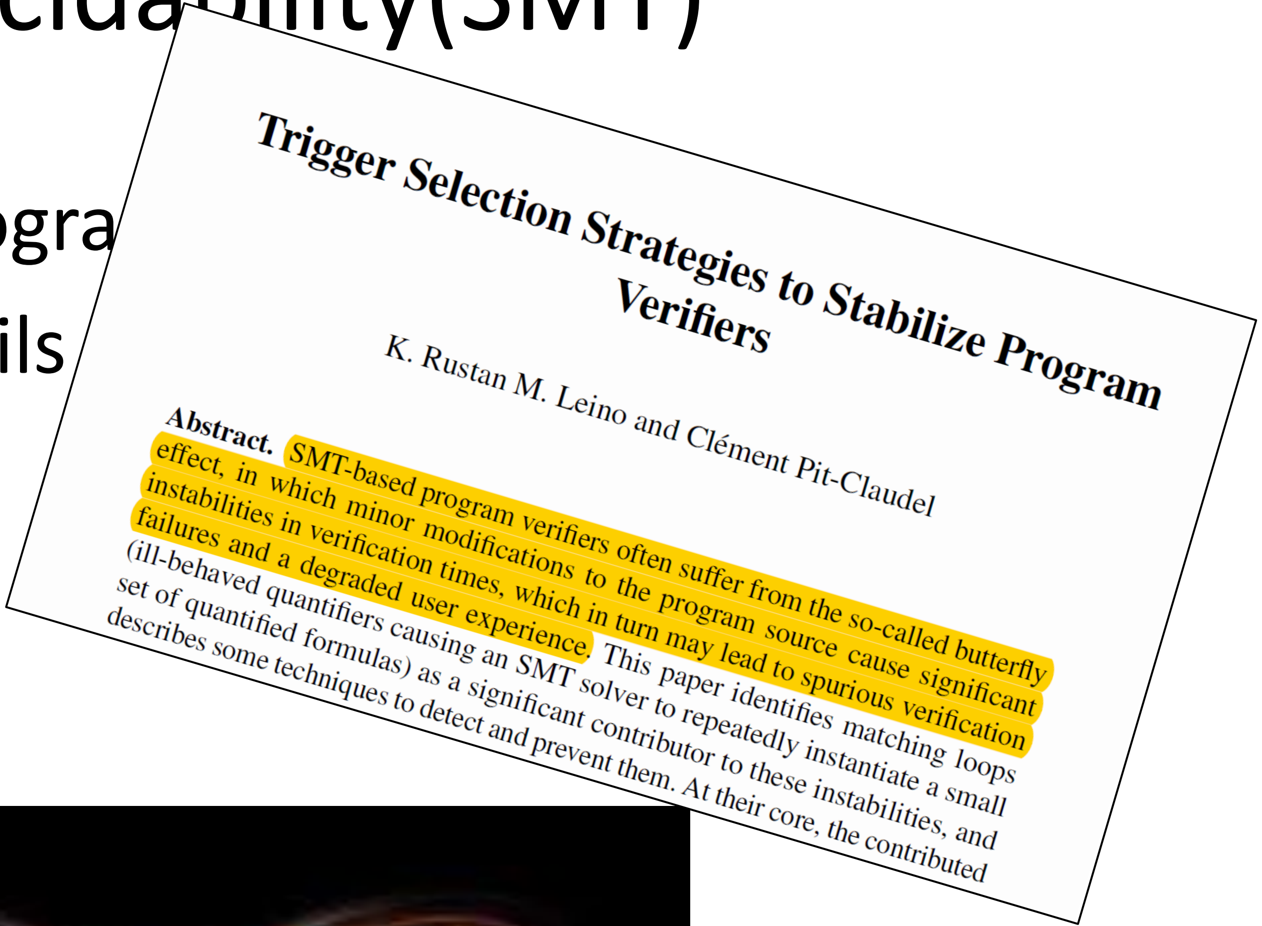


Unknown / Diverge

Proof

Effects of undecidability(SMT)

- The verifier may fail on tiny progra
- No explanation when tactics fails
  - Counterproofs



Trigger Selection Strategies to Stabilize Program Verifiers

K. Rustan M. Leino and Clément Pit-Claudel

Abstract. SMT-based program verifiers often suffer from the so-called butterfly effect, in which minor modifications to the program source cause significant instabilities in verification times, which in turn may lead to spurious verification failures and a degraded user experience. This paper identifies matching loops (ill-behaved quantifiers causing an SMT solver to repeatedly instantiate a small set of quantified formulas) as a significant contributor to these instabilities, and describes some techniques to detect and prevent them. At their core, the contributed

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

Proof Assistants

Ultimately limited by human

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

Ivy

Decidable deduction
Finite counterexamples
proof/code: ~0.2

Ultimately limited by undecidability

Decidable Models
Model Checking
Static Analysis

Expressiveness

Automation

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

- Limited fragment of first-order logic without theories
    - No function symbols
    - Restricted quantifier prefix: $\exists^{*}\forall^{*}\varphi_{QF}$
        - No $\forall\exists$

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

EPR Satisfiability

$$\exists x, y. \forall z.\ r(x,z) \leftrightarrow r(z,y)$$

$$=_{\mathrm{SAT}}\ \forall z.\ r(c_1,z) \leftrightarrow r(z,c_2)$$

$$=_{\mathrm{SAT}}\ \big(r(c_1,c_1) \leftrightarrow r(c_1,c_2)\big) \land \big(r(c_1,c_2) \leftrightarrow r(c_2,c_2)\big)$$

$$=_{\mathrm{SAT}}\ (p_{11} \leftrightarrow p_{12}) \land (p_{12} \leftrightarrow p_{22})$$

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

- Limited fragment of first-order logic without theories
  - No function symbols
  - Restricted quantifier prefix: $\exists^*\forall^*\varphi_{QF}$
- Finite model property
  - A formula is satisfiable iff it has a model of size:
    # constant symbols + # existential variables
- Complexity:
  - NEXPTIME-complete
  - Σ_2^P if relation arities are bounded by a constant
  - NP if quantifier prefix is also bounded by a constant

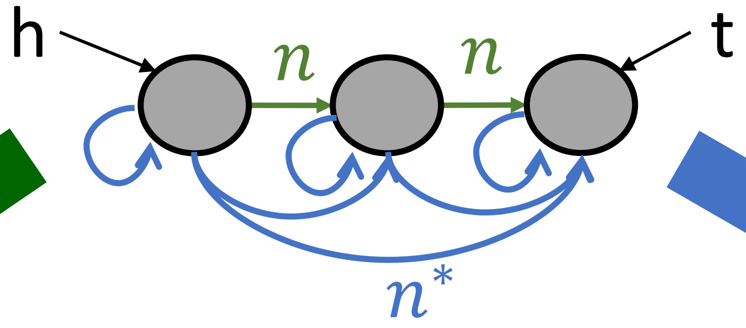
F. Ramsey. On a problem in formal logic. Proc. London Math. Soc. 1930

EPR++

- EPR++ allow acyclic function and quantifier alternations
  - E.g., $f: A \to B$ without $g: B \to A$
  - Maintains small model property of EPR
  - Finite complete instantiations
- But what can you possibly express in such a restricted logic?
  - Transtive closure over deterministic paths
  - Set cardinalities
  - Avoiding quantifier alternations
  - Encoding liveness and LTL [POPL'18]

Key idea: representing deterministic paths

[Shachar Itzhaky PhD, SIGPLAN Dissertation Award 2016]



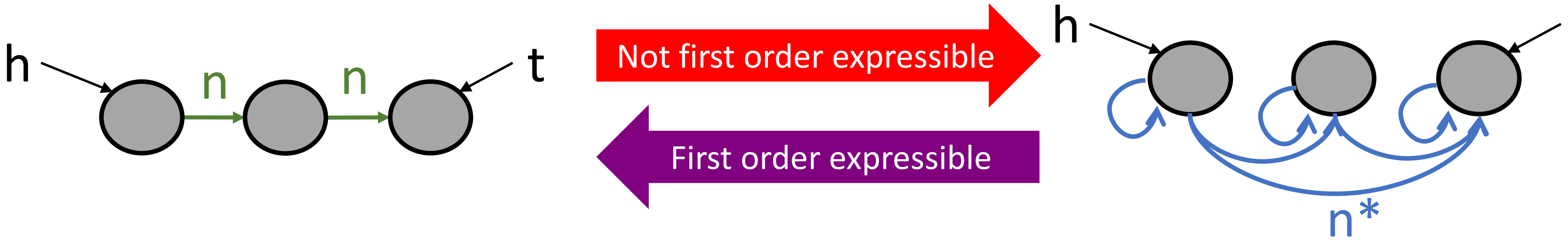
Alternative 1: maintain n

- n^* defined by transitive closure of n
- not definable in first-order logic

Alternative 2: maintain n^*

- n defined by transitive reduction of n^*
- Unique due to outdegree ≤ 1
- Definable in first order logic

$$n(x,y) \equiv n^*(x,y) \land x \neq y \land$$
$$\forall z. n^*(x,z) \rightarrow z = y \lor z = x$$



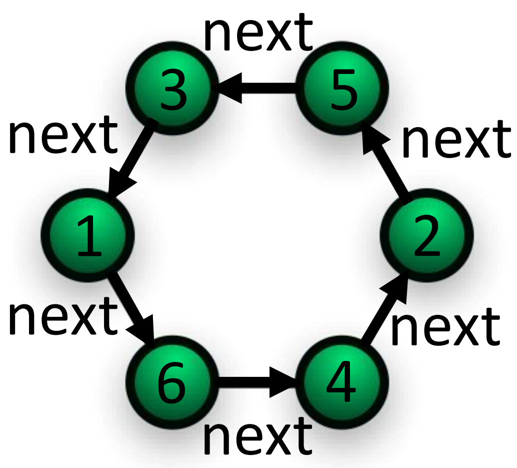
Ivy's principles

- 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)

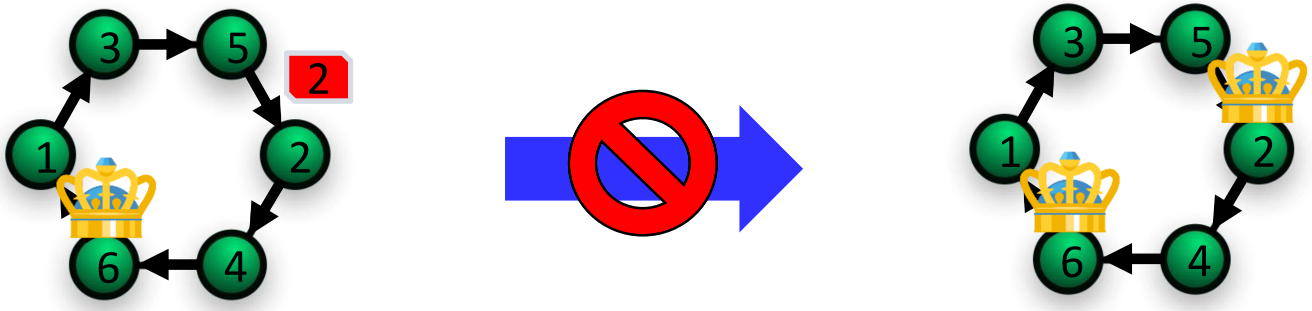
Languages and verification

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

Example: Leader election in a ring

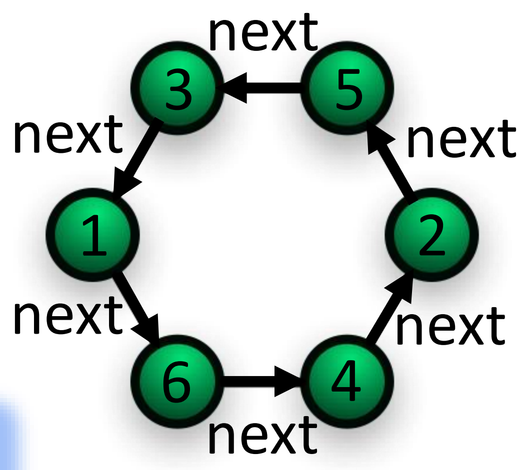


- Unidirectional ring of nodes, unique numeric ids
- Protocol:
  - Each node sends its id to the next
  - Upon receiving a message, a node passes it (to the next) if the id in the message is higher than the node's own id
  - A node that receives its own id becomes a leader
- Theorem: The protocol selects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



[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



- 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 i
  - A no
- Theorem

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.

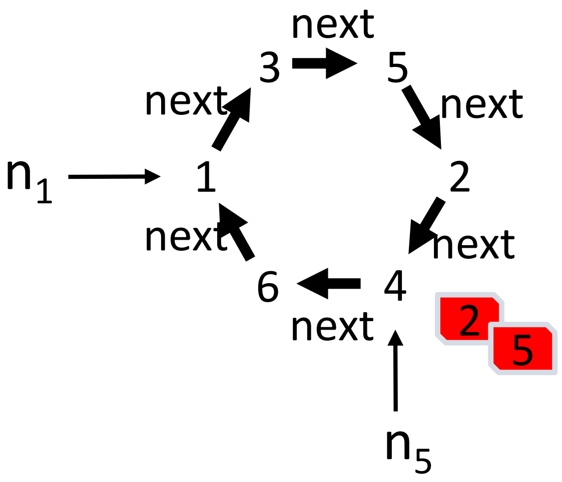
[CACM'79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes

Leader election protocol – first-order logic

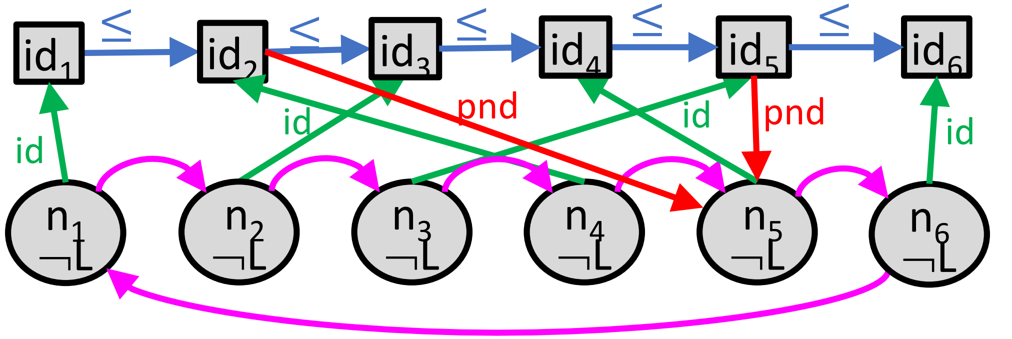
- $\leqslant$ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

Axiomatized in first-order logic

protocol state



first-order structure



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

Leader election protocol – first-order logic

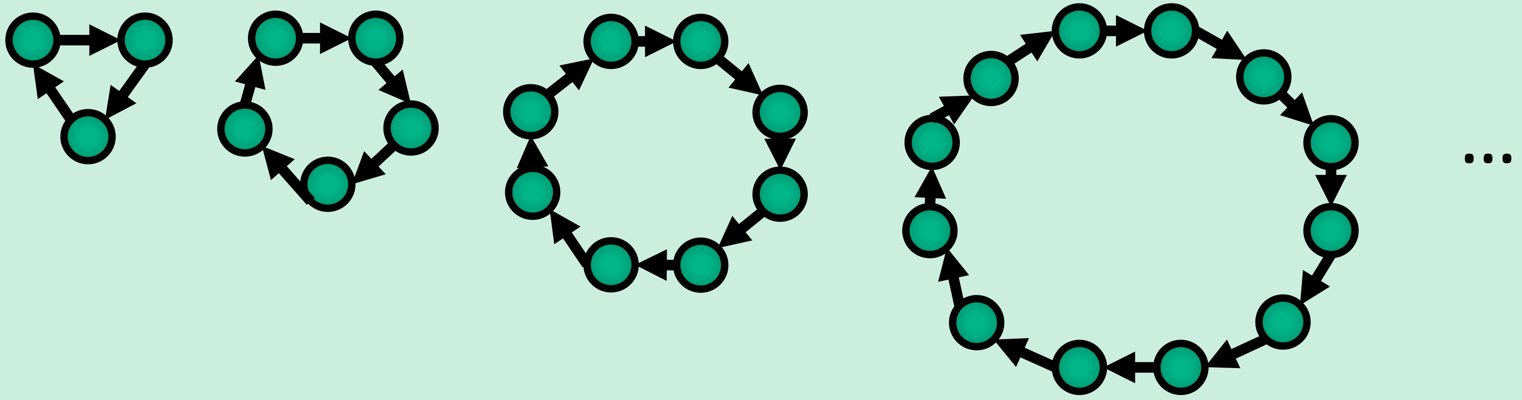
- $\leqslant$ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

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

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

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

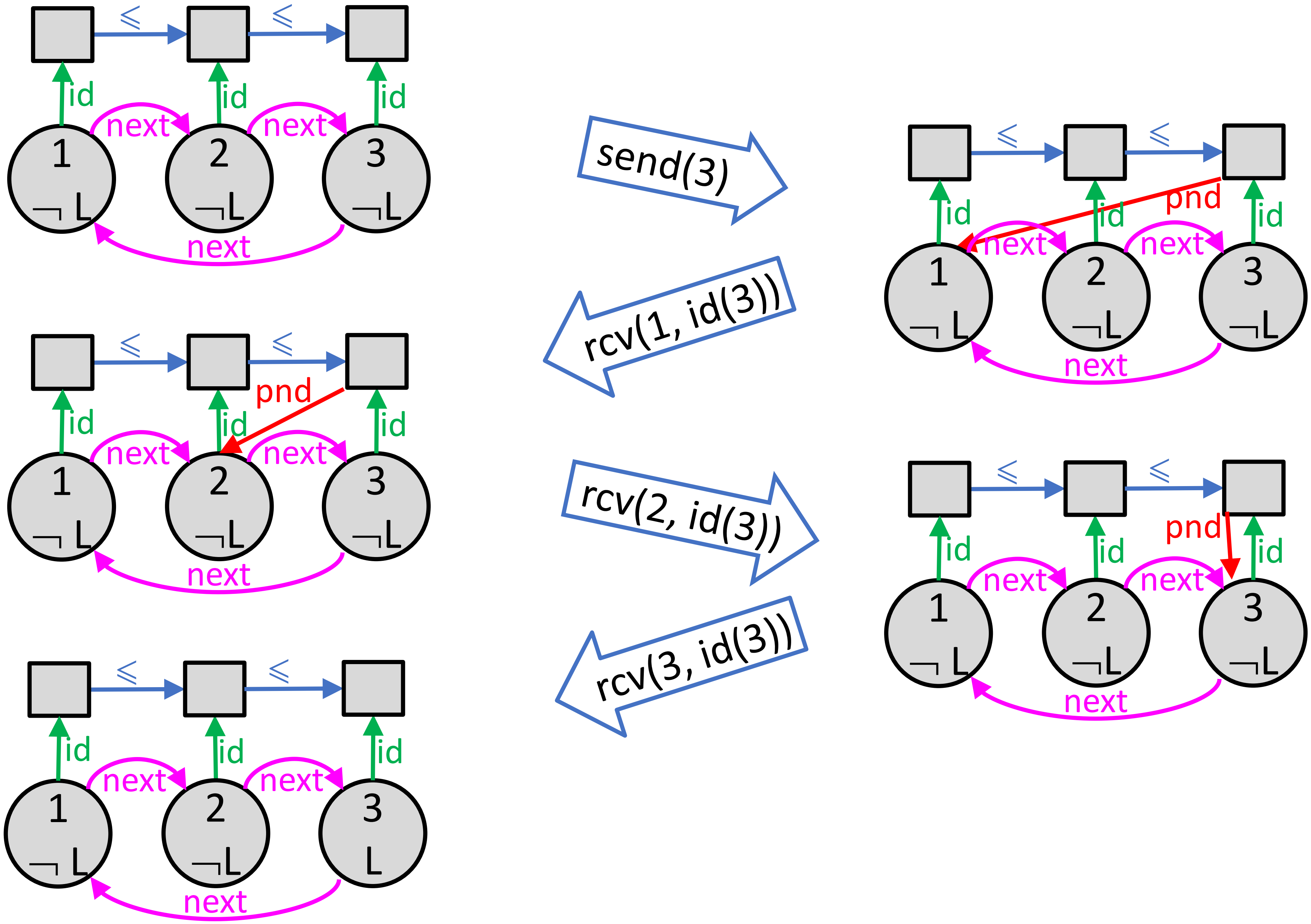
```
action receive(n: Node, m: ID) = {
  requires pending(m, n);
  if id(n) = m then
    // found leader
    leader(n) := true
  else if id(n) ⩽  m then
    // pass message
    “s := next(n)”;
    pending(m, s) := true
}
```

TR(send):

∃n,s: Node. “s = next(n)” ∧ ∀x:ID,y:Node. pending'(x,y)↔(pending(x,y)∨(x=id(n)∧y=s))

Bad:

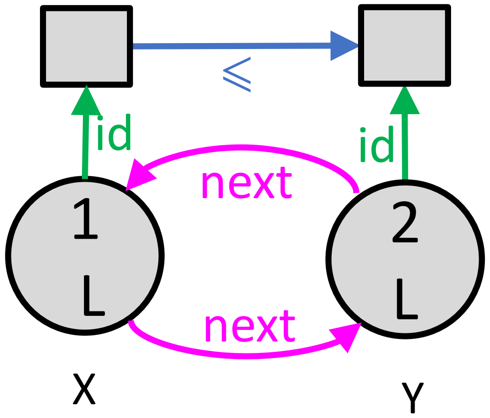
assert I0 = ∀ x,y: Node. **leader**(x)∧**leader**(y) → x = y

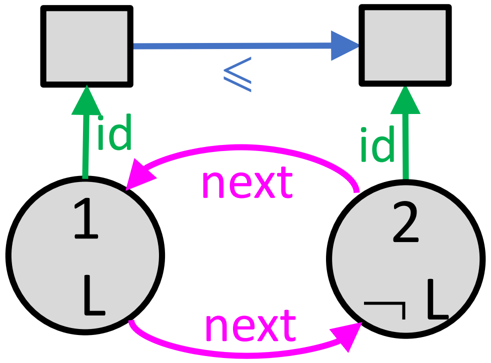


⩽
id
next
1
2
3
¬L
L
pnd
send(3)
rcv(1, id(3))
rcv(2, id(3))
rcv(3, id(3))

Representing Sets of States with First Order Formulas

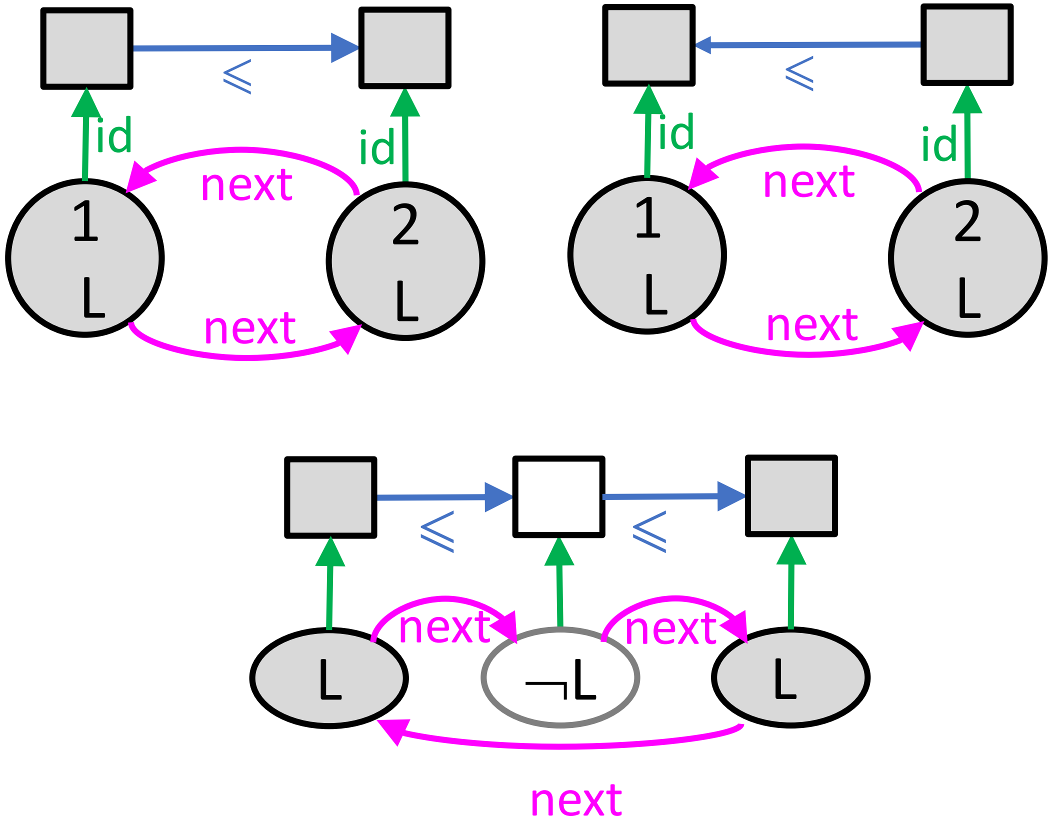
- Configurations with at least two leaders
  - $\exists$ X,Y: Node. **leader**(X)$\wedge$**leader**(Y) $\wedge$X$\neq$ Y





Representing Sets of States with First Order Formulas

- Configurations with at least two leaders
  - `∃ X,Y: Node. leader(X)∧leader(Y) ∧ X ≠ Y`



…

Leader election protocol – inductive invariant

Safety property: I_0

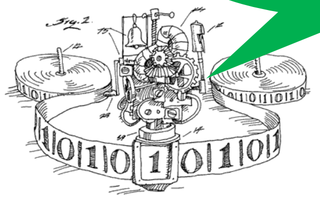
$I_0 = \forall x, y\text{: Node. } \textbf{leader}(x) \land \textbf{leader}(y) \rightarrow x = y$

Inductive invariant: $Inv = I_0 \land I_1 \land I_2 \land I_3$

$I_1 = \forall n_1, n_2\text{: Node. } \textbf{leader}(n_2) \rightarrow id[n_1] \leqslant id[n_2]$ — The leader has the highest ID

$I_2 = \forall n_1, n_2\text{: Node. } pending(id[n_2], n_2) \rightarrow id[n_1] \leqslant id[n_2]$ — Only the leader can be self-pending

$I_3 = \forall n_1, n_2, n_3\text{: Node. } btw(n_1, n_2, n_3) \land pending(id[n_2], n_1) \rightarrow id[n_3] \leqslant id[n_2]$ — Cannot bypass higher nodes

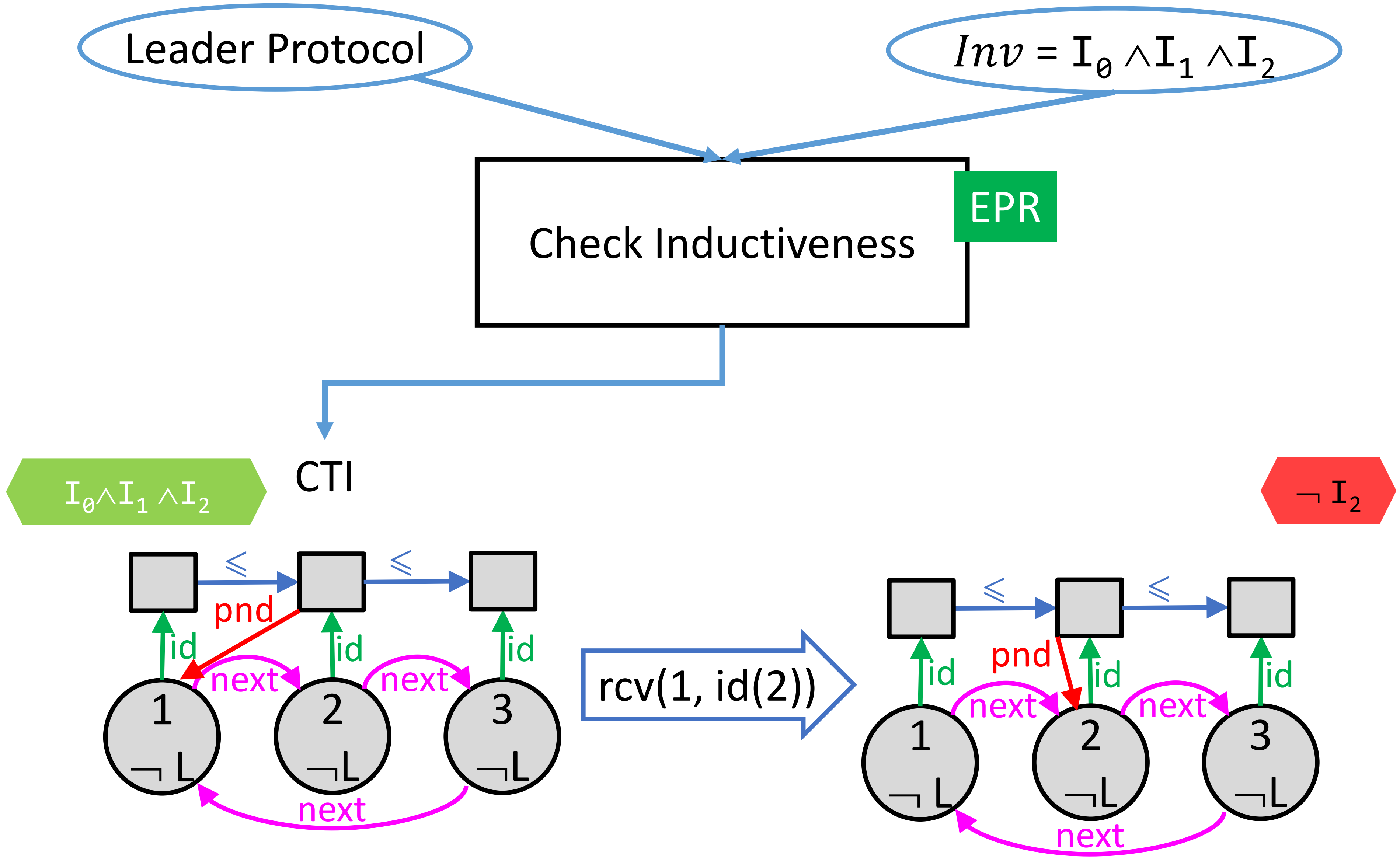


I *can* decide EPR!

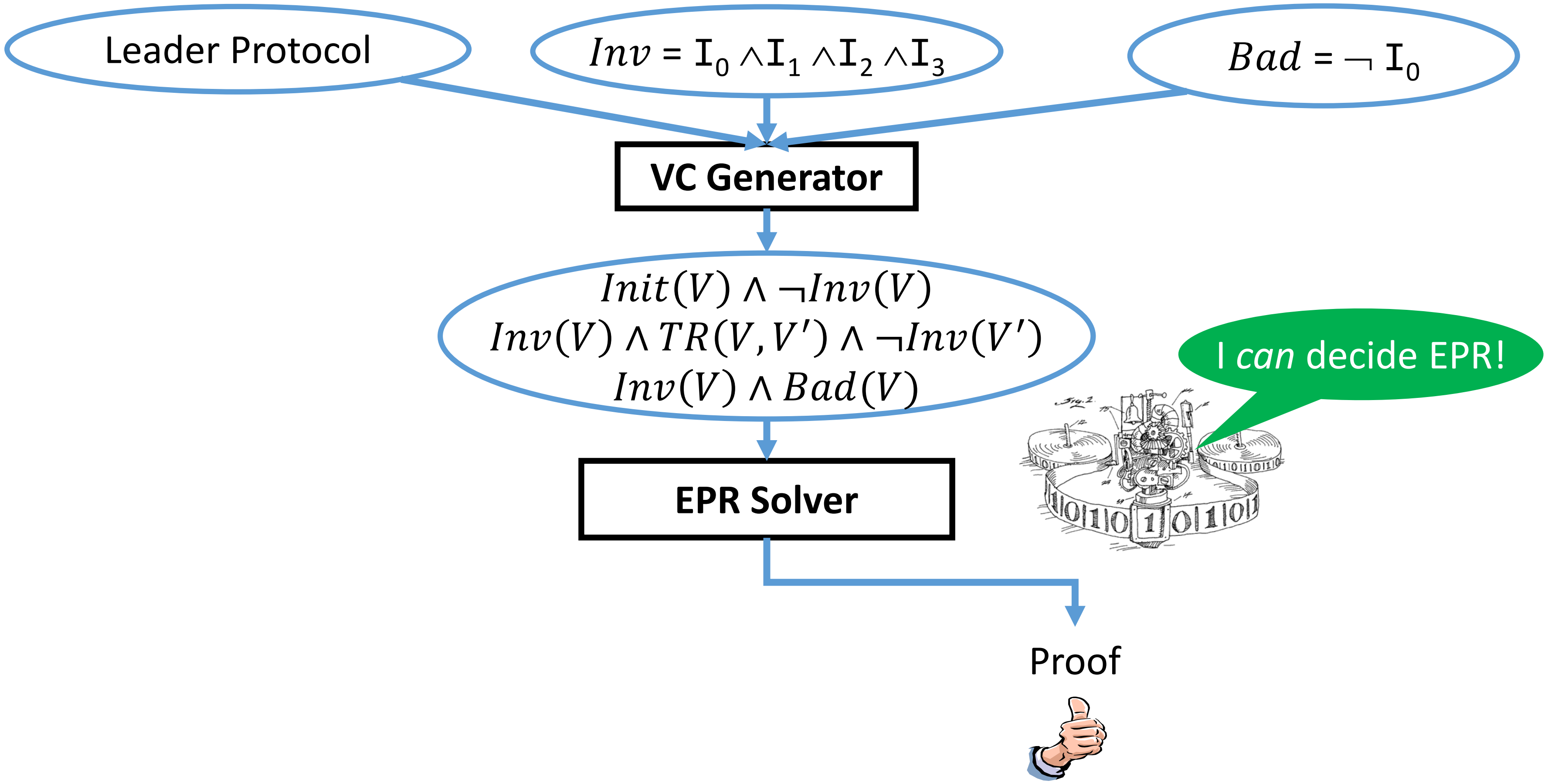
VC Generator → $Init(V) \land \neg Inv(V)$; $Inv(V) \land TR(V, V') \land \neg Inv(V')$; $Init(V) \land Bad(V)$ → **EPR Solver** → Proof

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

Ivy: check inductiveness

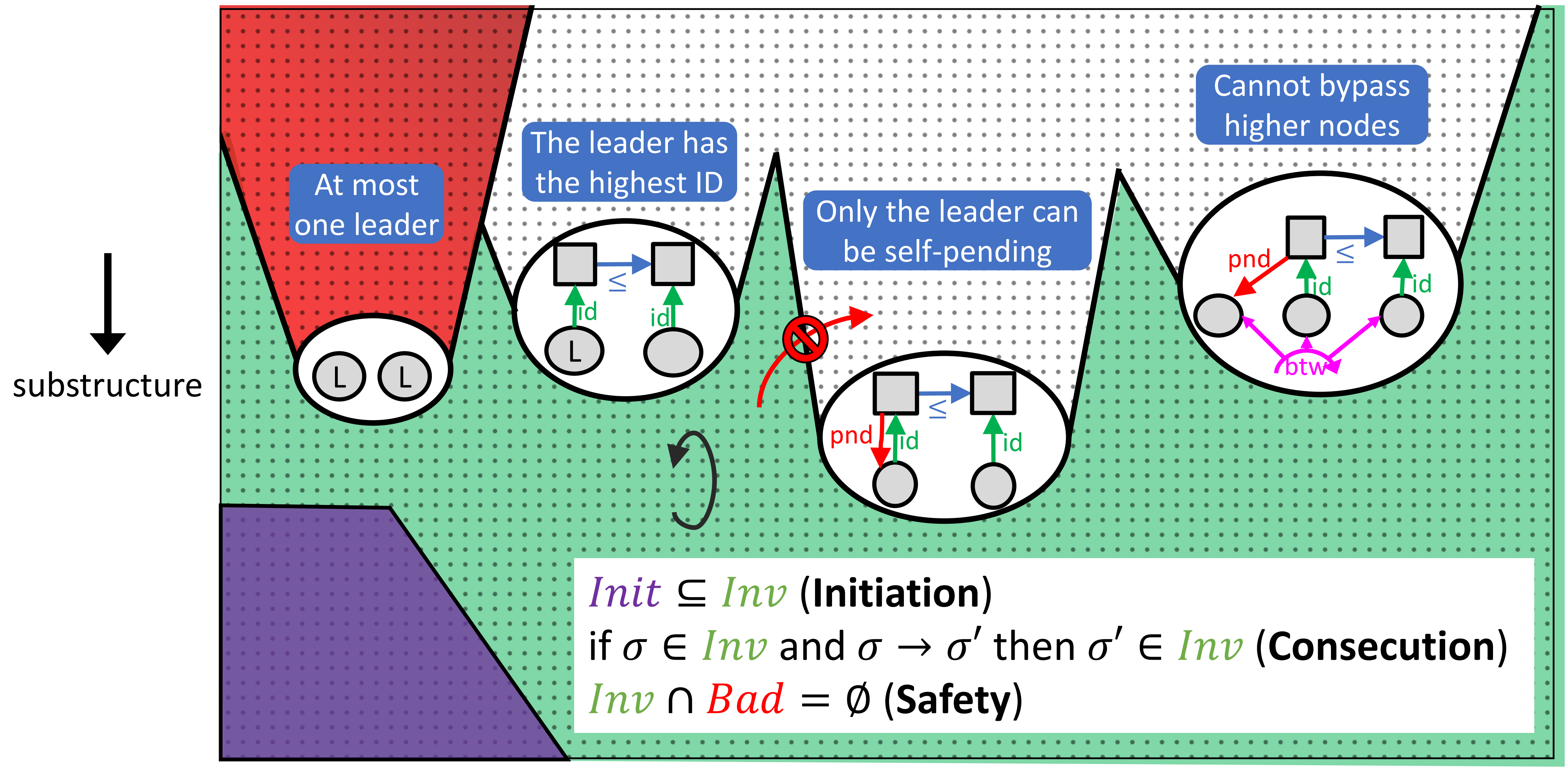


Ivy: check inductiveness



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

∀* invariant – excluded substructures



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

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

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

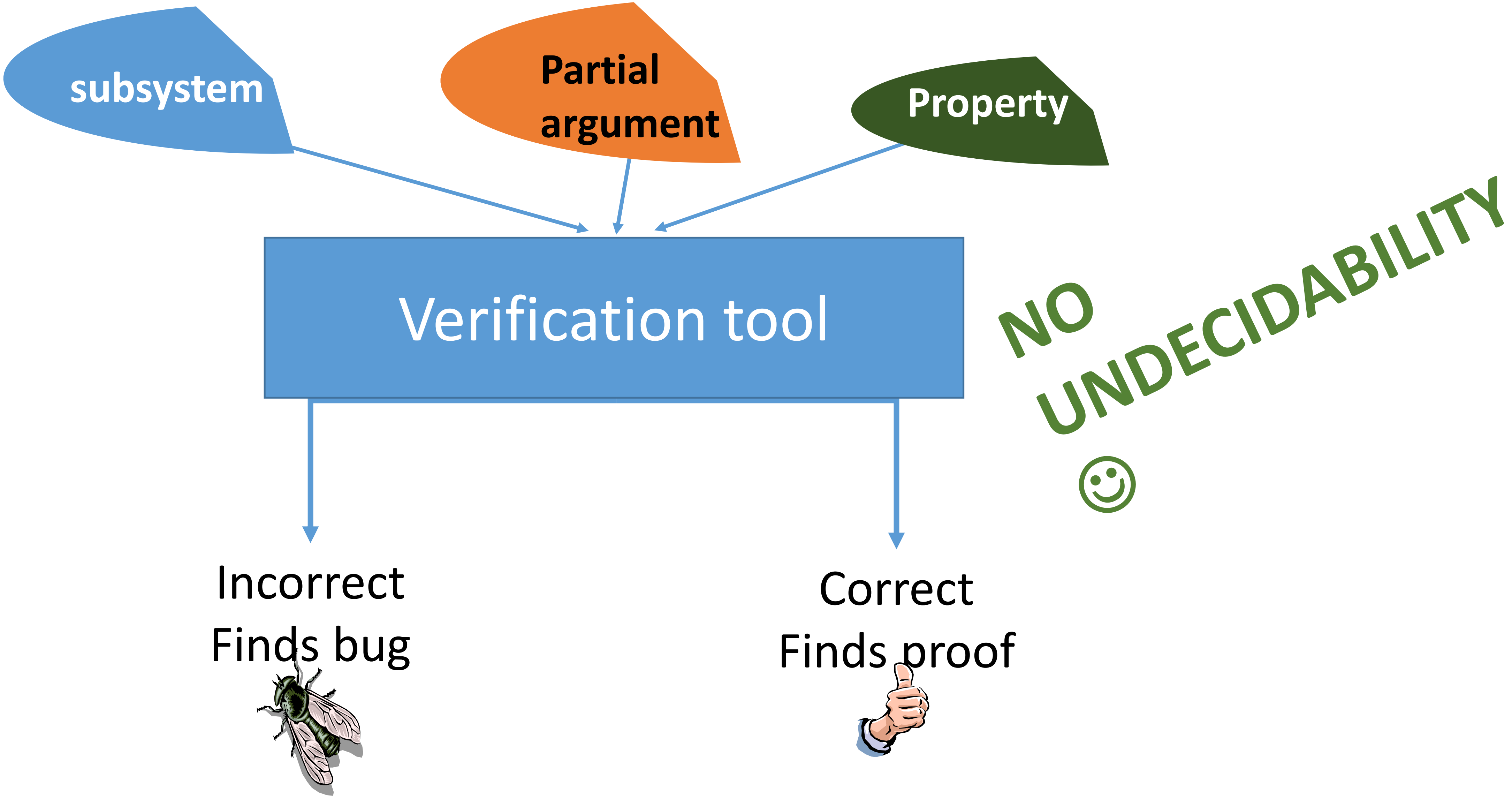
Modularity

Original system

Original inductive argument

Original property

Separate Verification of each module



An ADT for pid sets

datatype set(pid) = {

relation member (pid, set)

relation majority(set)

procedure empty returns (s:set)

procedure add(s:set,e:pid) returns (r:set)

specification {

procedure empty ensures $\forall p. \neg \mathrm{member}(p, \mathrm{s})$

procedure add ensures $\forall p. \mathrm{member}(p, r) \leftrightarrow (\mathrm{member}(p, \mathrm{s}) \vee p = e)$

property [maj] $\forall s, t. majority(s) \wedge majority(t) \rightarrow \exists p.\ \mathrm{member}(p, s) \wedge \mathrm{member}(p, t)$

}

}

We have hidden the cardinality and arithmetic

The key is to recognize that the protocol only needs property maj

Implementation of the set ADT

- Standard approach
  - Implement operations sets using array representation
    member(p, s)≡ ∃i. repr(s)[i] = p
  - Define cardinality of sets as a recursive function | |: set →int
    - majority(s)≡ |s| + |s| > |all|
  - Prove lemma by induction on |all|

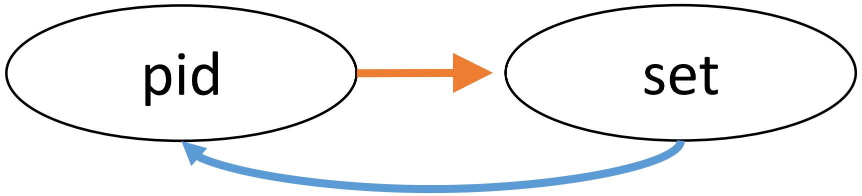
$$\forall s,t.\, |s| + |t| > |\mathrm{all}| \rightarrow \exists p.\, member(p,s) \wedge member(p,t)$$

- The lemma implies property maj
- All the verification conditions are in EPR**++**+limited arithmetic (FAU)

Quantifier alternation cycles

- Protocol state

  voters: $\underline{\text{pid} \rightarrow \text{set}}$



- Property maj

  $$\underline{\forall s, t: \text{set}.\ \exists p: \text{pid}.}\ \ \text{majority}(s) \wedge \text{majority}(t) \Rightarrow \text{member}(p, s) \wedge \text{member}(p, t)$$

- Solution: Harness modularity
  - Create an abstract protocol model that doesn't use voters
  - Prove an invariant using maj, then use this as a lemma to prove the concrete protocol implementation

Abstract protocol model

relation voted(pid, pid)
relation isleader(pid)
var quorum: set

procedure vote(v : pid, n : pid) = {
    require $\forall\, m.\, \neg \mathrm{voted}(\mathrm{v}, m)$;
    voted(v,n) := true;
}

procedure make_leader(n : pid, s : set) = {
    require $\mathrm{majority(s)}$;
    require $\forall m.\, \mathrm{member}(m, \mathrm{s}) \rightarrow \mathrm{voted}(m, \mathrm{n})$;
    isleader(n) := true;
    quorum := s;
}

Invariant:

- one leader: $\forall n, m.\, isleader(n) \wedge isleader(m) \rightarrow n = m$
- voted is a partial function: $\forall \mathrm{p}, n, m.\ \mathrm{voted(p,n)} \wedge \mathrm{voted(p,m)} \rightarrow n = m$
- leader has a quorum: $\forall n, m.\, isleader(n) \wedge member(m, quorum) \rightarrow voted(m, n)$

Provable in $\mathrm{EPR{+}{+}}$

Implementation

- Uses real network vote messages
- Decorated with ghost calls to abstract model
- Uses abstract mode invariant in proof

```
relation already_voted(pid)
handle req(p:pid, n:pid) {
    if ¬already_voted(p) {
        already_voted(p) := true;
        send vote(p,n);
        ghost abs.vote(p,n);
    }
}
```

call to abstract model must satisfy precondition

In place of property maj, we use the *one leader* invariant of the abstract model

$$\forall p, n. abs.voted(p,n) \rightarrow already\_voted(p)$$

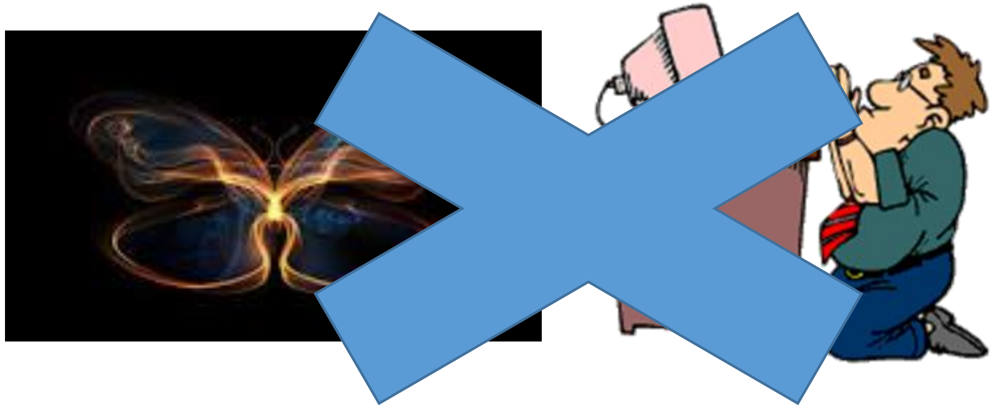
$$\forall p, n. network.vote(p,n) \leftrightarrow abs.voted(p,n)$$

$$\forall n. leader(n) \leftrightarrow abs.isleader(n)$$

...

Proof using Ivy/Z3

- For each module, we provide suitable inductive invariants
  - Reduces the verification to EPR++ verification conditions
    - the sub verification problems
- Each module's VC's in decidable fragment
  - Support from Z3
  - If not, Ivy gives us an explanation, for example a function cycle
- Z3 can quickly and reliably prove all the VC's



Proof Length

Protocol	System/Project	LOC	# manual proof	Ratio
RAFT	Coq/Verdi	530	50,000	94
	Ivy	560	200	0.36
MULTIPAXOS	Dafny/IronFleet	3000	12,000	4
	Ivy	330	266	0.8

Verification Effort

Protocol	System/Project	Human Effort	Verification Time
RAFT	Coq/Verdi	3.7 years	-
	Ivy	3 months (from ground up)	Few min
MULTIPAXOS	Dafny/IronFleet	Several years	6hr in cloud
	Ivy	1 month (pre-verified model)	few minutes on laptop

IVY summary

- A system with the following properties
  - Proof Automation
    - Can be used by non-experts
  - Transparency
    - The user either get CTI or an error message that the verification condition falls outside the decidable fragments
- Used to verify small but intricate distributed protocols all the way from the design to the implementation
- Publically available https://github.com/Microsoft/ivy