## Tentative Schedule

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

Oded Padon  Ken McMillan  Aurojit Panda  Sharon Shoham

http://apanda.github.io/ivy/
System S is safe if all the reachable states satisfy the property

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

\[ \text{Inv} \Rightarrow \varphi = \neg \text{Bad} \text{ (Safety)} \]
\[ \text{Init} \Rightarrow \text{Inv} \text{ (Initiation)} \]
\[ \text{if } \sigma \models \text{Inv} \text{ and } T(\sigma, \sigma') \text{ then } \sigma' \models \text{Inv} \text{ (Consecution)} \]
Deductive Verification

Solver

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

Program P

Candidate Invariant I

Safety Property \( \varphi \)

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: 

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

Solver

at(3) ⇒ x ≥ 1

3: <1, -2>

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

¬ (at(3) ⇒ x ≥ 1)
Deductive Verification

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

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

Proof

at(3) ⇒ x ≥1 ∧ y ≥0

at(3) ⇒ x ≥1
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 diagnosability
  – 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

<table>
<thead>
<tr>
<th>Coq</th>
<th>Dafny</th>
<th>IVY</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant</td>
<td>User</td>
<td>User</td>
<td>User + System</td>
</tr>
<tr>
<td>Deduction</td>
<td>User</td>
<td>System (Z3) + “User”</td>
<td>System (EPR)</td>
</tr>
</tbody>
</table>
The SAT Problem

- Given a propositional formula (Boolean function)
  - $\varphi = (a \lor b) \land (\neg a \lor \neg b \lor c)$
- Determine if $\varphi$ is valid
- Determine if $\varphi$ is satisfiable
  - Find a satisfying assignment or report that such does not exist
- 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 holds on models proportional to the number of existential variables
EPR SAT

\[ \exists x_1, x_2 \cdot \forall y \cdot r(x_1, y) \iff r(y, x_2) \]

\[ =_{\text{sat}} \forall y \cdot r(c_1, y) \iff r(y, c_2) \]

\[ =_{\text{sat}} (r(c_1, c_1) \iff r(c_1, c_2)) \land (r(c_1, c_2) \iff r(c_2, c_2)) \]

\[ =_{\text{sat}} (P_{11} \iff P_{12}) \land (P_{12} \iff P_{22}) \]
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

Algorithmic Deductive Verification

Program P \( \exists^* \forall^* \)

Candidate Inductive Invariant I \( \forall^* \)

Property \( \forall^* \exists^* \)

VC gen

Verification Conditions:
1) Init \( \land \neg I \)
2) I(V) \( \land \neg \varphi(V) \)
3) \([P](V, V') \land I(V) \land \neg I(V')\)

SAT Solver

CTI

Proof
CTI based Generalization

- Program
- Candidate Inductive Invariant

Inductive Invariant Found

Inductive?

Yes

Display “minimal” CTI

No

Modify candidate invariant

Inductive Invariant Found

Diagnose CTI

User

Heuristics
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

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

  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.
**Leader Protocol**

**action send =**

- \( n_1 := * \)
- \( n_2 := * \)
- assume left\((n_1, n_2)\)
- pending.insert(id[n_1], n_2)

**action receive =**

- \( m, n_1 := pending.remove() \)
- if id[n_1] = m then
  - // found leader
  - leader.insert(n_1)
- else if id[n_1] \( \leq m \) then
  - // pass message
  - \( n_2 := * \)
  - assume left\((n_1, n_2)\)
  - pending.insert(m, n_2)

- \( \leq \)\((ID, ID)\) – total order on node id’s
- btw\((Node, Node, Node)\) – the ring topology
- id: \( Node \rightarrow ID \) – relate a node to its 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: $\text{id}_1 \leq \text{id}_2$
  - Reflexive, transitive, anti-symmetric, total
- Relation: $\text{btw}(n_1: \text{Node}, n_2: \text{Node}, n_3: \text{Node})$
  // there is an acyclic path from $n_1$ to $n_3$ via $n_2$
  - Similar axioms
  - $\text{left}(a, b) \equiv a \neq b \land \forall x: \text{Node. btw}(a, x, b) \implies (x=a \lor x=b)$
Sound and Complete Order Axioms

- **Relation** \( \leq (\text{ID}, \text{ID}) \)
- \( \forall x: \text{ID}. \ x \leq x \) // Reflexive
- \( \forall x, y, z: \text{ID}. x \leq y \land y \leq z \Rightarrow x \leq z \) // Transitive
- \( \forall x, y: \text{ID}. x \leq y \land y \leq x \Rightarrow x = y \) // Anti-symmetric
- \( \forall x, y: \text{ID}. x \leq y \lor y \leq x \) // Total
- \( \text{succ}(a, b) \equiv \text{le}(a, b) \land a \neq b \land \left( \forall x: \text{ID}. a \leq x \Rightarrow x = a \lor b \leq x \right) \)
Sound and Complete Ring Axioms

• Relation btw(n₁: Node, n₂: Node, n₃: Node) // there is an acyclic path from n₁ to n₃ via n₂
• ∀x, w: Node. btw(w, x, x) // Reflexive
• ∀x, y, z, w: Node. btw(w, x, y) ∧ btw(w, y, z) ⇒ btw(w, x, z) // Transitive
• ∀x, y, w: Node. btw(w, x, y) ∧ btw(w, y, x) ⇒ x = y // Anti-symmetric
• ∀x, y, w: Node. btw(w, x, y) ∨ btw(w, y, x) // Totality
• ∀x, y, z: Node. x ≠ y ∧ btw(x, y, z) ⇒ btw(y, z, x) // Cyclic permutation
• left(a, b) ≡ a ≠ b ∧ ∀x: Node. btw(a, x, b) ⇒ (x = a ∨ x = b)
Bounded Model Checking (BMC)

Leader-Protocol \rightarrow Bound k \rightarrow VC gen \rightarrow Are there k transitions on an arbitrary ring with more than one leader? \rightarrow SAT Solver \rightarrow Unique leader

Counterexample \rightarrow Proof
BMC(4)

initial

pnd

id: 3
¬L
left

pnd

id: 3
¬L
left

pnd

id: 3
¬L
left

pnd

id: 3
¬L
left

pnd

id: 3
L
left

pnd

id: 3
L
left

pnd

id: 3
L
left

pnd

pnd

¬unique leader

snd

snd

rcv

rcv
Leader Protocol – 2\textsuperscript{nd} attempt

Axiom \( \forall x, y: \text{Node. } id[x] = id[y] \Rightarrow x = y \)

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)

Leader Protocol

I₀

VC gen

I₀=Unique Leader

CTI

id: 5
¬ L
pnd
5

id: 3
L
pnd

id: 5
L
pnd

id: 3
L
pnd

left

rcV
Generalizing CTIs

- 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
The leader should have the highest id

Project to \( \{L, \leq, id\} \)

\[ \forall n_1, n_2: \text{Node. } \neg (n_1 \neq n_2 \land \neg L(n_1) \land L(n_2) \land id[n_1] > id[n_2]) \]

BMC(2)
Generalizing CTIs(1)

Convert projection to conjecture

∀n₁, n₂: Node. ¬(n₁ ≠ n₂ ∧ ¬L(n₁) ∧ L(n₂) ∧ id[n₁] > id[n₂])

This doesn’t seem right….
The bound 2 is too low
Generalizing CTIs(1)

The leader should have the highest id

The leader should have the highest id

Project to \( \{L, \leq, id\} \)

\( \forall n_1, n_2: \text{Node}. \, \neg (n_1 \neq n_2 \land \neg L(n_1) \land L(n_2) \land id[n_1] > id[n_2]) \)

BMC(3) OK
Generalizing CTIs(1)

id: 5
¬L

id: 3
L

Convert projection to conjecture

∀n₁, n₂: Node. ¬(n₁ ≠ n₂ ∧ ¬L(n₁) ∧ L(n₂) ∧ id[n₁] > id[n₂])

Interp(3)

id: 5
id: 3
L

I₁ = ∀n₁, n₂: Node. ¬(L(n₂) ∧ id[n₁] > id[n₂])

This looks good, add to the invariant
Algorithmic Deductive Verification (2)

Leader Protocol

$\Box$ $I_0 \land I_1$

$\Box$ $I_0 = \text{Unique Leader}$

VC gen

SAT Solver

CTI

$pnd$

id: 3
$\neg L$

$pnd$

id: 5
$\neg L$

$pnd$

id: 3
$L$

$pnd$

id: 5
$\neg L$

$pnd$
Only the highest id can be self pending

Project to \(\{\text{pnd, id,} \leq\}\)

\[I_2 = \forall n_1, n_2: \text{Node. pending(id}[n_1], n_1) \Rightarrow \text{id}[n_2] \leq \text{id}[n_1] \]
Algorithmic Deductive Verification (3)

Leader Protocol

$I_0 \land I_1 \land I_2$

$I_0 =$ Unique Leader

VC gen

SAT Solver

CTI

id: 3
$\neg L$

id: 4
$\neg L$

id: 2
$\neg L$

id: 3
$\neg L$

id: 4
$\neg L$

id: 2
$\neg L$

pnd

pnd

pnd

pnd

pnd

pnd

rcv

left

left

left

left
Generalizing CTIs(3)

Project to \{pnd, id, \leq\}

Cannot overtake nodes with higher ids

BMC(3) NOT OK
Generalizing CTIs(3)

Project to \{pnd, id, btw, \leq\}

BMC(3)
I_3 = \forall n_1, n_2, n_3 : \text{Node. pending(id[n_1], n_3) \land btw(n_1, n_2, n_3) \land n_2 \neq n_3 \Rightarrow id[n_2] \leq id[n_1]}

Generalizing CTIs(3)
Algorithmic Deductive Verification(4)

Leader-Protocol

$\text{I}_0 \land \text{I}_1 \land \text{I}_2 \land \text{I}_3$

$I_0 =$ Unique Leader

VC gen

Verification Conditions:
1) Init $\land \neg I$
2) $I(V) \land \neg \varphi(V)$
3) $[P](V, V') \land I(V) \land \neg I(V')$

SAT Solver

Proof
# Verified Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model Types</th>
<th>Model Relations</th>
<th>Property Size (# Literals)</th>
<th>Invariant Size (# Literals)</th>
<th>CTI Gen. Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader in Ring</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>2</td>
<td>6</td>
<td>11</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>DB Chain Replication</td>
<td>4</td>
<td>13</td>
<td>11</td>
<td>35</td>
<td>7</td>
</tr>
<tr>
<td>Chord</td>
<td>1</td>
<td>13</td>
<td>35</td>
<td>46</td>
<td>4</td>
</tr>
<tr>
<td>Consensus</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>13</td>
<td>3</td>
</tr>
</tbody>
</table>
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
  - $\text{trusted}(h) = \exists h'. \text{in}(h') \land \text{sent}(h', h)$
  - Explore locality
- Restricted functions: Idempotent

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

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]

# Tentative Schedule

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