From TVLA to IVY

Mooly Sagiv
REPS at 60
Edinburgh, September 11, 2016
The TVLA Team
The TVLA Principles

• Concrete semantics expressed as evolving relations/graphs in $\text{FO}^{\text{TC}}$
  – Celebrate unboundedness
  – No arithmetic

• Abstract Interpretation with Canonical Partially Disjunctive Abstraction

• Effective heuristics for automatic postcondition calculation
  – Kleene evaluation
  – Focus
  – Coerce
  – Differencing
Example: Concrete Interpretation

\[ x = \text{NULL} \]

\[ t = \text{malloc(..);} \]

\[ t \rightarrow \text{next} = x; \]

\[ x = t \]

\[ \text{return } x \]
return x

Example: Abstract Interpretation

x = NULL

F
T

\text{t = malloc(..);}

\text{t \rightarrow next = x;}

x = t

return x

empty

T

F
Lessons learned

• $\text{FO}^{\text{TC}}$ is powerful
• But $\text{FO}^{\text{TC}}$ reasoning is complicated
  • Decidability of implications
  • Scalability of disjunctive abstractions
  • TVLA heuristics are effective for experts and specialized domains
    • Effective for shape analysis of common data structures and graduate students
    • But hard to apply by non experts and complicated clients
    • CEGAR has limited power
The IVY System

Kenneth McMillan

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

Oded Padon
IVY Principles

• Concrete semantics expressed as evolving relations/graphs in Effectively Propositional Logic (EPR)
  • Explore locality of updates
  • Simulate FOTC via differencing

• Interactive interference of conjunctive invariants

• [Abstract interpretation is coming]
Ivy: Safety Verification by Interactive Generalization

Oded Padon, Kenneth McMillan, Aurojit Panda, Sharon Shoham

PLDI 2016
http://microsoft.github.io/ivy/
Ivy: Safety Verification by Interactive Generalization

• Verification of distributed systems

• Modeling infinite-state systems in a way which allows decidable automated reasoning (EPR)

• Interactive discovery of inductive invariants
System $S$ is safe if no bad state is reachable
System $S$ is safe iff there exists an inductive invariant $\text{Inv}$ s.t.:

\begin{align*}
\text{Init} \subseteq \text{Inv} & \quad (\text{Initiation}) \\
\text{if } \sigma \in \text{Inv} \text{ and } \sigma \overset{\cdot}{\rightarrow} \sigma' & \text{ then } \sigma' \in \text{Inv} & (\text{Consecution}) \\
\text{Inv} \cap \text{Bad} = \emptyset & \quad (\text{Safety})
\end{align*}
Challenges for Deductive Verification

1. Formal specification:
   • Modeling the system
   • Formalizing the safety property

2. Inductive Invariants
   • Hard to specify manually
   • Hard to infer automatically

3. Deduction – Checking inductiveness
   • Undecidability of implication checking
     • Unbounded state, arithmetic, quantifier alternation
Existing Approaches for Verification of Infinite-State Systems

• Automated invariant inference
  • Abstract Interpretation
  • Ultimately limited 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 provides inductive invariant and proves it
  • Huge effort (10-100 lines of proof per line of code)
Our Approach in Ivy

• Restrict the specification language for decidability
  • Deduction is decidable with SAT solvers
  • Challenge: verify complex systems using a restricted language
    • Solution: domain specific axioms

• Finding inductive invariants (still undecidable):
  • Combine automated techniques with human guidance
  • Graphical user interaction
  • Key: generalization from counterexamples to induction
  • Decidability allows reliable automated checks
Relational Modeling Language (RML)

• Designed to make verification tasks decidable
  • Yet expressive enough to model systems
• Turing-Complete
• Universally quantified inductive invariants are decidable to check
• System state described by finite (unbounded) relations
• No numerics
• Simple (quantifier-free) updates
• Universally quantified axioms (domain specific)
  • Total orders, partial orders, lists, trees, rings, quorums, ...
# Languages for verification

<table>
<thead>
<tr>
<th>Language</th>
<th>Executable</th>
<th>Deduction</th>
<th>Expresiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, Java, Python...</td>
<td>✓</td>
<td>Undecidable</td>
<td>Turing-Complete</td>
</tr>
<tr>
<td>Dafny</td>
<td>✓</td>
<td>Undecidable</td>
<td>Turing-Complete</td>
</tr>
<tr>
<td>SMV</td>
<td>✗</td>
<td>Decidable Verification of Temporal properties</td>
<td>Finite-state</td>
</tr>
<tr>
<td>Ivy</td>
<td>in progress</td>
<td>Decidable (EPR)</td>
<td>Turing-Complete</td>
</tr>
<tr>
<td>Alloy</td>
<td>✗</td>
<td>Undecidable</td>
<td>Turing-Complete</td>
</tr>
<tr>
<td>Coq, Isabelle/HOL</td>
<td>✓</td>
<td>Manual</td>
<td>Turing-Complete</td>
</tr>
<tr>
<td>TLA+</td>
<td>✗</td>
<td>Manual</td>
<td>Turing-Complete</td>
</tr>
</tbody>
</table>
Invariant Inference In Ivy

Inv = \neg \text{Bad}

I can decide inductiveness!
Invariant Inference In Ivy

Inv = \neg \text{Bad}

\sigma_1, \sigma_1' \text{ -- CTI}

\varphi(\sigma_1, \sigma_1')

Generalize from CTI
Invariant Inference In Ivy

Inv = \neg Bad \land \varphi(\sigma_1, \sigma_1')

Check Inductiveness

Counterexample To Induction (CTI)
Invariant Inference In Ivy

\[ \text{Inv} = \neg \text{Bad} \land \phi(\sigma_1, \sigma_1') \]

- \( \sigma_2, \sigma_2' \) – CTI
- Generalize from CTI
- \( \phi(\sigma_2, \sigma_2') \)
Invariant Inference In Ivy

\[ \text{Inv} = \neg \text{Bad} \land \varphi(\sigma_1, \sigma_1') \land \varphi(\sigma_2, \sigma_2') \]

- Key challenge for invariant inference: generalization
- Ivy’s approach: put the user in the loop \textit{interactive generalization}

Generalize from CTI

User \leftrightarrow Automation
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

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

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 Election Protocol (RML)

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

protocol = (send | receive)*

next(a)=b \iff \forall x: Node. x=a \lor x=b \lor btw(a,b,x)

assert I0 = \forall x,y: Node. leader(x) \rightarrow id(y) \leq id(x)
\[ \forall x, y : \text{Node.} \quad \text{leader}(x) \rightarrow \text{id}(y) \leq \text{id}(x) \]
$I \emptyset = \forall x, y: \text{Node.}$

$\text{leader}(x) \rightarrow \text{id}(y) \leq \text{id}(x)$
Inductive Invariant for Leader Election

- $\leq (\text{ID}, \text{ID})$ – total order on node id’s
- $\text{btw} (\text{Node}, \text{Node}, \text{Node})$ – the ring topology
- $\text{id}: \text{Node} \rightarrow \text{ID}$ – relate a node to its id
- $\text{pending}(\text{ID}, \text{Node})$ – pending messages
- $\text{leader}(\text{Node})$ – leader$(n)$ means $n$ is the leader

**Safety property:** $I_0$

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

**Inductive invariant:** $\text{Inv} = I_0 \land I_1$

$I_1 = \forall x, y, z: \text{Node}. \neg (\text{btw}(x, y, z) \land \text{pending}(\text{id}(y), x) \land \text{id}(y) \neq \text{id}(z))$

How can we find an inductive invariant without knowing it?
Ivy: Check Inductiveness (1)

Leader Protocol

Inv = I0

Bad = ¬ I0

“Leader has maximal id”

Check Inductiveness

CTI

I0

rcv(1, id(1))

next

next

pnd
Ivy: Generalize from CTI (1)

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

Only the highest id can be self pending
Ivy: Generalize from CTI (1)

User’s Generalization

Only the highest id can be self pending
Ivy: Generalize from CTI (1)

Only the highest id can be self pending

User’s Generalization

1
next
2

id

pnd

id

next

id

pnd

id

btw

id

pnd

L
Ivy: Generalize from CTI (1)

Only the highest id can be self pending

User’s Generalization
Ivy: Generalize from CTI (1)

Only the highest id can be self pending

Looks good, add to the invariant as I1
**Ivy: Check Inductiveness (2)**

Leader Protocol

\[ \text{Inv} = I\emptyset \land I1 \]

Bad = \( \neg I\emptyset \)

---

Check Inductiveness

---

CTI

---

\( I\emptyset \land I1 \)

---

rcv(1, id(2))

---

\( \neg I1 \)
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

Ivy: Generalize from CTI (2)

Cannot bypass nodes with higher ids
Ivy: Generalize from CTI (2)

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

Cannot bypass nodes with higher ids

Counterexample Trace
Ivy: Generalize from CTI (2)
Ivy: Generalize from CTI (2)

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

Cannot bypass nodes with higher ids

Proof

BMC(3)
Ivy: Generalize from CTI (2)

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

Cannot bypass nodes with higher ids

This looks good, add to the invariant as I2

UNSAT CORE
Generalization with $btw$

btw("1", "2", "3")

I2
Ivy: Check Inductiveness (3)

Leader Protocol
Inv = I0 ∧ I1 ∧ I2
Bad = ¬ I0

Check Inductiveness

Proof
\[ \text{Init} \subseteq \text{Inv} \text{ (Initiation)} \]

if \( \sigma \in \text{Inv} \) and \( \sigma \rightarrow \sigma' \) then \( \sigma' \in \text{Inv} \) \text{ (Consecution)}

\[ \text{Inv} \cap \text{Bad} = \emptyset \text{ (Safety)} \]
Completeness and Interaction Complexity

• Any generalization from CTI adds one universally quantified clause

• A universally quantified invariant in CNF with $N$ clauses, 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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model Types</th>
<th>Relations &amp; Functions</th>
<th>Property (# Literals)</th>
<th>Invariant (# 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>12</td>
<td>3</td>
</tr>
<tr>
<td>Learning Switch</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>18</td>
<td>3</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>Lock Server 500 Coq lines [Verdi]</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>21</td>
<td>8 (1h)</td>
</tr>
<tr>
<td>Distributed Lock 1 week [IronFleet]</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>26</td>
<td>12 (1h)</td>
</tr>
<tr>
<td>Paxos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Paxos: Work in progress
Expressiveness vs. Automation

<table>
<thead>
<tr>
<th></th>
<th>Coq</th>
<th>Dafny</th>
<th>Ivy</th>
<th>Fully Automatic Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant</td>
<td>User</td>
<td>User</td>
<td>User + System</td>
<td>System</td>
</tr>
<tr>
<td>Deduction</td>
<td>User</td>
<td>System (Z3) + “User”</td>
<td>System (EPR Z3)</td>
<td>System</td>
</tr>
</tbody>
</table>
Summary

• RML – modeling language that makes deduction decidable
  • Many systems can be verified (axioms for orders, trees, rings, ...)
• Interactive generalization for finding inductive invariants
• Application to the domain of distributed protocols
• User intuition and machine heuristics complement each other:
  • User has intuition that leads to better generalizations
  • Machine is better at finding bugs and corner cases
• Interactive process assists user to gain intuition about the protocol

Expressiveness

Coq

Dafny

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

I can decide inductiveness!
4 Lessons Learned from Tom

• Spoonfeed the reader
• Look the other way
• Think deeply
• Dedication/Dedication/Dedication
Thanks

James Cheney

Xavier Rival