Using Simulated Execution in Verifying Distributed Algorithms

Toh Ne Win, Michael Ernst, Stephen Garland, Dilsun Kirli Kaynar, Nancy Lynch

"How to help a theorem prover with execution data"
Goal: make theorem provers easier to use

• Why do we want to use a prover?
  – To verify general, infinite state systems

• What's hard about using a prover?
  – They get stuck and need human input
What kind of human input?

Program to be verified → Theorem prover → Verified proof
What kind of human input?

Program to be verified

Tactics

Proof structure

Case analysis

Which facts to use

Theorem prover

Verified proof
What kind of human input?

Program to be verified

- Lemmas
  - Human insight and intuition on invariants of reachable states

- Tactics
  - Proof structure
  - Case analysis
  - Which facts to use

Theorem prover

Verified proof
Traditional approaches

Program to be verified

Lemmas

Human insight and intuition on invariants of reachable states

Tactics

Improved tactics to prover

Theorem prover

Verified proof
Using execution data to help provers

• Programs are often tested before verification
  – Testing shows errors quickly
  – Verification is expensive in human time

• Execution data is normally thrown away
  – What information can be kept for proofs?
Generating tactics

Program annotated for testing by execution

Lemmas

Translator from test cases to prover language

Proof structure and case analysis

Tactics

Theorem prover

Verified proof
Generating lemmas

Program annotated for testing by execution

Lemmas

Daikon runtime analysis tool
generalizes over executions

Conjectured invariants

Translator from test cases to prover language

Proof structure and case analysis

Theorem prover

Verified proof
Outline

• Motivation: execution-assisted theorem provers

• **Formal model: IO automaton**

• Case study: Lamport's Paxos protocol

• Lemmas: conjectured invariants

• Tactics: proof outline

• Conclusion
Formal model: IO automaton

• Model for distributed systems [Lynch/Tuttle 89]
  – Labeled (infinite, nondeterministic) state machine
  – First order logic to define transitions
Formal model: IO automaton

- Model for distributed systems [Lynch/Tuttle 89]
  - Labeled (infinite, nondeterministic) state machine
  - First order logic to define transitions
- Multiple levels of abstraction
  - Abstract specification automaton
  - Layered implementation automata
Verification methods

- Simulation relations for refinement
Verification methods

- Simulation relations for refinement
- Invariant assertions for implementations

Diagram:
- Specification
- Abstract Implementation
- Concrete Implementation
- Simulation relations
- Invariants
IOA language and tools

• IOA interpreter
  – Allows simulated execution of one automaton, or of a pair for refinement
  – User-specified scheduling to resolve nondeterminism
• IOA translators to proving languages
  – The Larch Prover
  – Isabelle/HOL
Outline

- Motivation: execution-assisted theorem provers
- IO automaton model
- Case study: Lamport's Paxos protocol
- Lemmas: conjectured invariants
- Tactics: proof outline
- Conclusion
Paxos in IOA

- Specification for consensus

- Globalized implementation using ballots and quorums
Specification for consensus

automaton Consensus

% Inputs and outputs are externally visible.
signature
  input init (i:Node, v:Value)
  input fail (i:Node)
  output decide (i:Node, v:Value)
  internal chooseVal (v:Value)

states
  proposed, chosen : Set[Value] := {}
...

transitions
  internal chooseVal (v)
  pre
    v ∈ proposed;
    chosen = {}
  eff
    chosen := {v}
Implementation by Global1

Automaton Global1

signature
  input init (i:Node, v:Value)
  input fail (i:Node)
  output decide (i:Node, v:Value)
  internal internalDecide (b:Ballot)...

states
  succeeded, createdBallots: Set[Ballot]
  ...

internal internalDecide(b:Ballot)
pre
% The ballot was created.
b ∈ createdBallots;
% There was a quorum that voted on the ballot.
∃ quorum: Set[Node] (quorum ∈ wquorums ∧ ...
Gameplan for proof

- Show that Global1 implements Consensus
  - Simulation relation proof
Gameplan for proof

• Show that Global1 implements Consensus
  – Simulation relation proof
• Need invariants on Global1
Outline

- Motivation: execution-assisted theorem provers
- IO automaton model
- Case study: Lamport's Paxos protocol
- Lemmas: conjectured invariants
- Tactics: proof outline
- Conclusion
Uses of invariants

- Lemmas in proofs
  - Of simulations relations
  - Of other invariant statements
  - Often needed because the induction hypothesis for a proof must be strong enough
How to conjecture invariants

- Execute automaton using test cases
- Use Daikon tool on execution data
  - Analyzes execution data
  - Outputs properties true for observed executions
  - Invariants in first order logic
Issues with conjectured invariants

- **Unsound**
  - Statistical analysis reduces false positives
  - Use prover to prove conjectured invariants

- **Incomplete**
  - Necessary because search space is infinite

- **Needs test cases**
  - In practice, test cases exist
  - We use randomized scheduling
  - Trial-and-error execution usually enough
Conjectured invariants: example

• Paxos case study
  - Found 4 of 6 invariants needed for simulation relation proof

\[
\begin{align*}
\text{val(nonNull)} & \subseteq \text{proposed} \\
\text{succeeded} & \subseteq \text{createdBallots} \\
0 & = \text{size(succeeded} \cap \text{dead)} \\
0 & = \text{size(voted[aNode] \cap abstained[aNode])}
\end{align*}
\]
What was not found

• Invariants with
  
  – Existential quantifiers
    
    • If a ballot has succeeded, a quorum voted for it
      
      \[(b \in \text{succeeded} \Rightarrow \exists \text{quorum : Set[Node]} (\text{quorum} \in \text{wquorums} ^ \land \forall n : \text{Node} (n \in \text{quorum} \Rightarrow b \in \text{voted}[n])))\]

  – Too many boolean clauses
    
    • If a ballot has non-nil value, it is the same value as all earlier non-dead ballots
      
      \[(\text{val}[b] \neq \text{nil} ^ \land b' < b) \Rightarrow (\text{val}[b'] = \text{val}[b] ^ \land b' \in \text{dead})\]
Outline

• Motivation: execution-assisted theorem provers
• IO automaton model
• Case study: Lamport's Paxos protocol
• Lemmas: conjectured invariants
• Tactics: proof outline
• Conclusion
Gameplan for proof

• Show that Global1 implements Consensus
  – Simulation relation proof
To prove a forward simulation relation

- A implements B if there exists $f$ such that $f$:
  - Is a relation on states[A] and states[B]
  - Satisfies a start condition
  - Satisfies a step condition
To prove a forward simulation relation

- A implements B if there exists $f$ such that:

 Specifications:
- Specification automaton B

 Implementation:
- Implementation automaton A

\[ \text{start condition} \]
- Start state: $a$
- Pre-condition: $b$

\[ \text{step condition} \]
- Reachable state: $a'$
- Post-condition: $b'$

\[ \beta \text{ witness execution} \]
- Action: $\alpha$

Red = proof obligation
Forward sim: interpreter support

- Paired execution mode of IOA interpreter
  - For testing forward simulations
  - User annotates program for witness executions

- Mechanics of paired execution
  - Execute implementation automaton
  - Use annotations to drive execution of specification automaton
  - Check that $f$ holds
Annotation example

for internal internalDecide (b) do
  if (b ∈ Global1.succeeded) then
    ignore
  elseif (Global1.val[b] = nil) then
    ignore
  ...
  else
    fire internal chooseVal
    (Global1.val[b].val)
  fi
Generating prover tactics from tests

- Translate testing annotations into proof scripts
  - For start condition
    - Pick witness start state \( \beta \)
  - For step condition
    - Tactic: structural induction on action data type
    - Use conditionals ('if') in annotations to perform case splits
    - Pick witness execution \( \beta \)
Forward sim: step example

% Annotation
for internal internalDecide (b) do
  if (b ∈ Global1.succeeded) then
    ignore
  elseif (Global1.val[b] = nil) then
    ignore
  ...
  else
    fire internal chooseVal
    (Global1.val[b].val)
  fi

% Proof
prove enabled(internalDecide(b)) => ∃ β ....% Step condition
resume by cases (b ∈ Global1.succeeded)
% case true
resume by specializing β to []
Outline

- Motivation: execution-assisted theorem provers
- IO automaton model
- Case study: Lamport's Paxos protocol
- Lemmas: conjectured invariants
- Tactics: proof outline
- Conclusion
Discussion

• Better theorem proving experience
  – Less human effort
  – Lets designers concentrate on high-level proof
• Designers have concept of high-level proof
  – Theorem provers get stuck in details
• Tactics: provide proof structure (82/150 lines)
  – What remains is rephrasing of facts
• Lemmas: provide invariants (4/6)
  – Missing ones syntactically evident in program code
Research directions

• Better conjectured invariants
  – Analyze IOA code statically for invariant templates
    • Find predicates in code, use as left side of implications

• Better proof tactics, more automation
  – Which lemmas are used in all IOA proofs?
  – What ordering of lemmas?
    • E.g., “apply definition of automaton effects only after inducting on the action type”
Conclusion

• Theorem provers need lemmas and tactics
  - Execution data can provide some of both

• Lemmas
  - Generalize over execution data
    • Conjectured invariants

• Tactics
  - Annotations for paired testing provides
    • Proof outline
    • Existential witnesses

• Contribution: easier to use theorem prover