Automated Inductive Verification of Parameterized Protocols

Abhik Roychoudhury

National University of Singapore

I.V. Ramakrishnan

State University of New York at Stony Brook

Parameterized Systems

- Infinite family of finite state processes parameterized by a type *e.g.* n process token ring
- Occurs widely in distributed algorithms *e.g.* an algorithm to ensure mutual exclusion of a shared resource among n users.
- Verification of temporal properties of parameterized systems is undecidable. [AK86]

Verifying Safety Properties

$$\frac{\begin{array}{c}\text{I1: } \texttt{init(S)} \Rightarrow \texttt{p1(S)} \\ \text{I2: } \texttt{p1(S)} \land \texttt{trans(S, T)} \Rightarrow \texttt{p1(T)} \\ \text{I3: } \texttt{p1(S)} \Rightarrow \texttt{p(S)}\end{array}}{\text{AG } \texttt{p}}$$

Our method is sound but incomplete which

- Automatically proves I1, I2, I3 without resorting to finitary abstractions. These premises are proved by *induction*.
- Does not provide automatic support for discovering the strengthened invariant `p1`

A machinery for induction

- A model checking like proof rule which can dispense *base case*, finite part of *inductive step*.
- A deductive proof rule for recognizing *induction hypothesis* (with/without strengthening)

---

1. Should allow interleaving of proof rules for tight integration
2. **Let us build such a lightweight prover !**

Inductive proofs via Transformations

1. Encode the temporal property to be verified as well as the parameterized system as a logic program P_0.
2. Convert the verification proof obligation to a predicate implication proof obligation $P_0 \vdash \mathtt{p} \Rightarrow \mathtt{q}$ ($\mathtt{p}, \mathtt{q}$ are predicates)
3. Construct a program transformation sequence $P_0, P_1, \ldots, P_k$ s.t.
   (a) Semantics of P_0 = Semantics of P_k
   (b) from the syntax of P_k we automatically infer $P_k \vdash \mathtt{p} \Rightarrow \mathtt{q}$

Program transformations correspond to Proof rules.

Logic Programming

- A (definite) logic program is a finite set of Horn clauses. Computation via clause resolution.
- $M(P)$ = Least Herbrand Model Semantics of P
  = Gnd. atoms which are logical consequences of P
  = Gnd. atoms which are proved by resolution with clauses of P

$P =$
```
p(a).
p(f(X)) :- p(X).
```

$M(P) = \{\mathtt{p(a)}, \mathtt{p(f(a))}, \mathtt{p(f(f(a)))}, \ldots\}$

$P' =$
```
p(f(X)) :- p(X).
```

$M(P') = \{\}$

Proof Rule: Unfolding

$$
\begin{array}{l}
\texttt{q} :- \texttt{g}, \boxed{\texttt{p}}, \texttt{g}'. \\
\\
\texttt{p} :- Bd_1. \\
\texttt{p} :- Bd_2. \\
\vdots \\
\texttt{p} :- Bd_n.
\end{array}
\quad \xrightarrow{Unfold} \quad
\begin{array}{l}
\texttt{q} :- \texttt{g}, \; Bd_1, \; \texttt{g}'. \\
\texttt{q} :- \texttt{g}, \; Bd_2, \; \texttt{g}'. \\
\vdots \\
\texttt{q} :- \texttt{g}, \; Bd_n, \; \texttt{g}'. \\
\texttt{p} :- Bd_1. \\
\vdots \\
\texttt{p} :- Bd_n.
\end{array}
$$

- A step of Clause Resolution. Corresponds to computation.

Proof Rule: Folding

$$\frac{P_j\ (j \le i):\quad \begin{array}{l} \mathtt{p} :- Bd_1. \\ \vdots \\ \mathtt{p} :- Bd_n. \end{array}}{P_i:\quad \begin{array}{l} \mathtt{q} :- \mathtt{g}, \boxed{\begin{array}{l} Bd_1, \\ Bd_2, \\ \\ Bd_n, \end{array}} \begin{array}{l} \mathtt{g}'. \\ \mathtt{g}'. \\ \\ \mathtt{g}'. \end{array} \\ \vdots \end{array} \xrightarrow{Fold} \mathtt{q} :- \mathtt{g}, \boxed{\mathtt{p}}, \mathtt{g}'.}$$

- Replaces occurrence of clause body (from a previous program in the transformation sequence) by its head.

Folding and Inductive Reasoning

Folding corresponds to recognizing induction hypothesis in an induction proof of a predicate impl. $P_0 \vdash \mathtt{p} \Rightarrow \mathtt{q}$

$$P_0: \quad \begin{array}{l} \mathtt{p(X)\ \text{:-}\ p1(X), p2(X).} \\ \mathtt{q(a).} \\ \mathtt{q(f(X))\ \text{:-}\ q(X).} \end{array}$$

$$P_i: \quad \begin{array}{l} \mathtt{p(a).} \\ \mathtt{p(f(X))\ \text{:-}\ \boxed{\mathtt{p1(X),\ p2(X).}}} \\ \mathtt{q(a).} \\ \mathtt{q(f(X))\ \text{:-}\ q(X).} \end{array} \quad \xrightarrow{\textit{Fold}} \quad \begin{array}{l} \mathtt{p(a).} \\ \mathtt{p(f(X))\ \text{:-}\ p(X).} \\ \mathtt{q(a).} \\ \mathtt{q(f(X))\ \text{:-}\ q(X).} \end{array}$$

Nesting of proofs

- Our semantics preserving unfold/fold transformations can simulate induction proofs.
- Human proofs are *hierarchical* ; an induction proof may use a lemma which is again proved by induction.
- Nested induction proofs simulated by nesting pred. impl. obligations.
- An unfold/fold proof of an impl. $\mathtt{p} \Rightarrow \mathtt{q}$ uses pred. impl. $\mathtt{p}' \Rightarrow \mathtt{q}'$ and $\mathtt{p}'' \Rightarrow \mathtt{q}''$ as lemma. These are again proved by unfold/fold.

Recipe for a lightweight prover

- A proof rule for a step of model checking : Unfolding
- A proof rule to recognize induction hypothesis : Folding
- A proof rule to decompose an assertion $P \vdash \mathtt{p} \Rightarrow \mathtt{q}$ to new assertions $P \vdash \mathtt{p_1} \Rightarrow \mathtt{q_1}, \ldots, P \vdash \mathtt{p_k} \Rightarrow \mathtt{q_k}$

These rules will be arbitrarily interleaved in a proof.

Proof Search of $P \vdash \mathtt{p} \Rightarrow \mathtt{q}$

1. Repeatedly unfold the clauses of p and q without risking non-termination.

2. Fold the unfolded clauses of p, q.

3. (a) Compare the transformed definitions of p and q to compute:

$$\{(\mathtt{p}_1, \mathtt{q}_1), \ldots, (\mathtt{p}_k, \mathtt{q}_k)\} \text{ s.t.}$$

$$\left(\bigwedge_{1 \leq i \leq k} P \vdash \mathtt{p}_i \Rightarrow \mathtt{q}_i\right) \Rightarrow P \vdash \mathtt{p} \Rightarrow \mathtt{q}$$

   (b) Prove $P \vdash \mathtt{p}_1 \Rightarrow \mathtt{q}_1, \ldots, P \vdash \mathtt{p}_k \Rightarrow \mathtt{q}_k$ via program transformations.

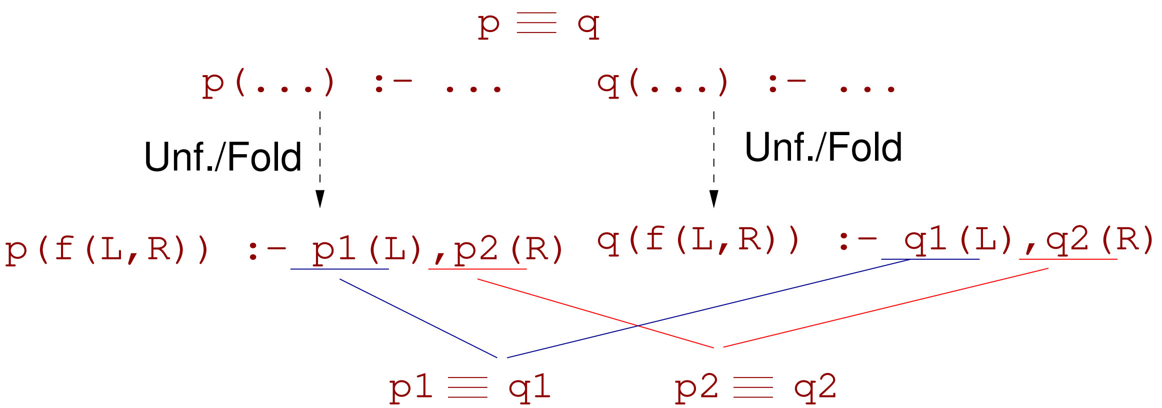
   (c) $P \vdash \mathtt{p} \Rightarrow \mathtt{q}$ now established by syntactic check.

Design of a Lightweight Prover

- Geared to automate *nested* induction proofs where each induction proceeds without *strengthening* of hypothesis.
- The prover is both automated and programmable. User can set problem-specific information before the proof attempt begins.
- Typical user settings to control proof search: Network topology of the parameterized system, Predicates encoding temporal property
- User settings are exploited by the prover to *resolve nondeterminism* when multiple transformation rules are applicable.

User Settings can alter Proof Search

- Network topology is implicit in the recursive structure of predicates describing the parameterized network.
- Unf./Fold of these predicate occurrences exposes this structure.
- Knowledge of the network topology then exploited by the prover to choose nested proof obligations to be dispensed.



Implementation

- Built an automated prover based on program transformations on top of XSB LP system.
- **Process Structure:** Used to prove invariants for parameterized networks of diff. topologies: uni- and bi-directional chains, rings, trees.
- **Pred. Impl. Proof Obligations:** Given the inv. and the parameterized network, finding the pred. impl. to be proved requires no user ingenuity.
- **Case Studies :**
  Cache Coherence Protocols e.g. Illinois, Berkeley RISC
  Java Meta-Locking Algorithm (Sun MicroSystems)

Case Study: Java Meta-Locking Algorithm

- In Java program execution, any object can be synchronized upon by different threads.
- To maintain mutually exclusive access to an obj., each obj. maintains synchronization data (queue of threads waiting to access obj).
- The meta-locking algorithm is a distributed algorithm observed by each thread and any object in accessing the synch. data.
- Need to prove mutual exclusion in access of synch. data.
- Previous proof attempts include an informal proof sketch [ADG^{+}99] and model checking of instances of the algorithm. [BSW00]

Modeling and Verification

- Parameterized System : $Handoff \parallel Object \parallel Thread \parallel \ldots \parallel Thread$
- Thread : *Finite* state. Acquires/releases meta-lock via fast/slow path.
- Handoff : *Finite* state sys. which resolves contention between acquiring and releasing threads. Eliminates busy waiting.
- Object : *Inf.* state sys. Records # of threads waiting for the meta-lock.
- Mutex strengthened by considering local states of Handoff, Object.
- Generates and proves 39 pred. impl. in the proof.

Experimental Results

Protocol	Invariant	Time	# Unf.	#Ded.
Meta-Lock	$\#\text{owner} + \#\text{handout} < 2$	129.8	1981	311
Mesi	$\#\text{m} + \#\text{e} < 2$	3.2	325	69
	$\#\text{m} + \#\text{e} = 0 \vee \#\text{s} = 0$	2.9	308	63
Illinois	$\#\text{dirty} < 2$	35.7	2501	137
Berkeley RISC	$\#\text{dirty} < 2$	6.8	503	146
Tree-cache	$\#\text{bus\_with\_data} < 2$	9.9	178	18

Related Work

- Developing Decision Procedures for Subclasses: [EN95, ID99] :
- Network Invariants [KM95,CGJ95,LHR97] :
- Rich language Model Checking [KMM$^+$97,EFM99] :
- Invisible Invariants: [PRZ01]
- Theorem Proving [PVS,ACL2,Oyster-CLAM,RRL] :

Comparison to Related Work

- Network Invariants: Uses a specific induction principle.

$$P \models \varphi \qquad\qquad I \models \varphi \Rightarrow I \times P \models \varphi$$

  This induction proof can be constructed using our transformations.

- Rich language model checking:
  - Rich language assertions capture an exact finite description of the state space. In our work, this is captured by recursive predicates.
  - Constructs uniform proofs. We construct induction proofs.

Comparison to Related Work

- Invisible Invariants:
  - Proves inductive assertions by abstraction, not induction.
  - Provides automated support for assertion strengthening.
- Theorem Proving: Integrate model-checking (and other automated techniques) as decision procedures to the theorem prover.

  We enhance the underlying evaluation technique (unfolding) of a model checker (XMC) with deductive capabilities (*e.g.* folding).

Summary

- Our technique allows arbitrary interleaving of unfolding (model checking) and folding (deduction) steps in a verification proof.
- Since deductive steps are applied lazily:
  - deductive steps do not affect efficiency of model checking steps
  - for finite state systems our technique reduces to model checking.
- Need to integrate automated assertion strengthening techniques to the proof technique (via program analysis).
- Need more proof experience with non-linear network topologies.